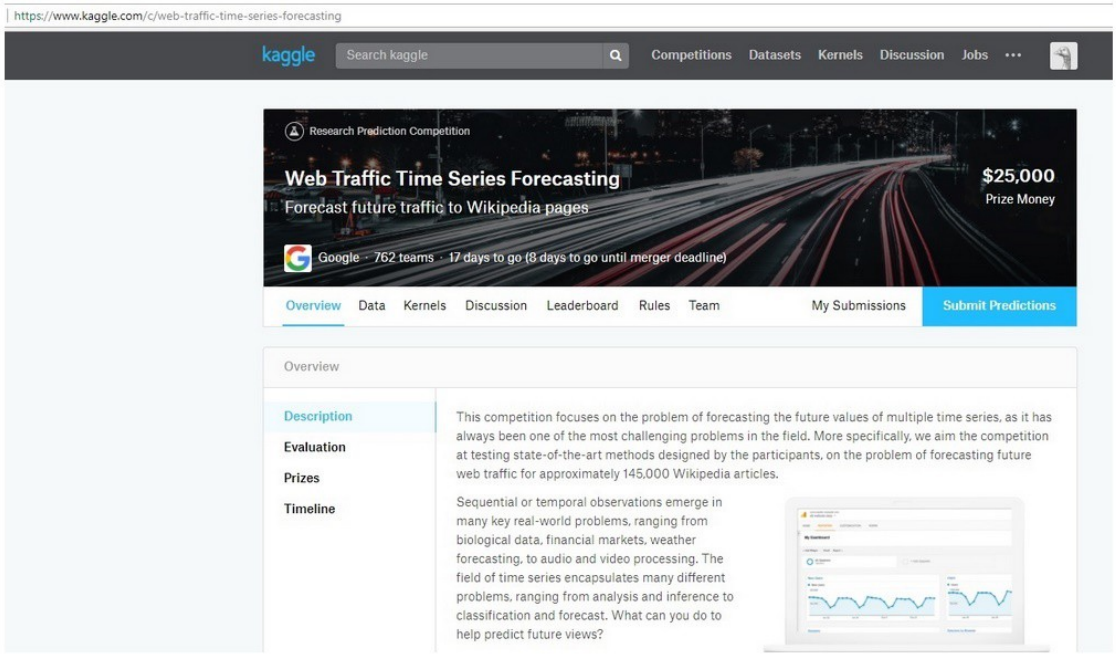
Parcours Data Scientist, projet n°9, Rapport

Christophe Coudé, 2017

1- Présentation

Le projet 9 consiste dans le choix d'une compétition Kaggle.J'ai choisi la compétition Web Traffic Time Series Forecasting :
https://www.kaggle.com/c/web-traffic-time-series-forecasting
car elle traite d'un champ de la data science non encore traitée dans le parcours, celui des séries temporelles.

Cette compétition se concentre sur la question de prévision des futures valeurs d'un ensemble de séries temporelles, sujet faisant partie des problèmes les plus stimulants et difficiles de la data science. Les participants doivent tester les différentes méthodes de prévision du traffic sur près de 145063 articles wikipedia.



Les données consistent donc 145063 lignes contenant chacune le nom de la page ainsi que 18 mois de fréquentation jour par jour allant du 01/07/2015 au 31/12/2016. Ces données sont rassemblées dans le fichier train_1.csv. Le but de la compétition est de prévoir pour chaque article, le traffic qui a eu lieu du 01/01/2017 au 01/03/2017, soit 60 jours, et de les comparer aux valeurs réelles que possède Kaggle. La mesure de cette adéquation se fait à partir de la mesure SMAPE : https://en.wikipedia.org/wiki/Symmetric_mean_absolute_percentage_error
Il s'agit donc de reprendre les données du fichier key_1.csv comportant pour chaque page / date un Id et de créer un fichier associant à chacun de ces Id, la prévision du traffic.
Ce fichier peut alors être soumis à Kaggle qui va calculer le SMAPE et déterminer la place du score dans la hiérarchie.

Pendant tout le rapport, on appellera indifféremment Time Serie, TS ou série temporelle, l'évolution temporelle du nombre de consultation (ou trafic) d'une page Wikipedia.

2- Exploration

Apperçu du fichier train_1.csv

```
In [4]: train.head(2)
```

Out[4]:

	Page	2015-07-01	2015-07-02	2015-07-03	2015-07-04	2015-07-05	2015-07-06	2015-07-07	2015-07-08	2015-07-09	...	201
0	2NE1_zh.wikipedia.org_all-access_spider	18.0	11.0	5.0	13.0	14.0	9.0	9.0	22.0	26.0	...	32.(
1	2PM_zh.wikipedia.org_all-access_spider	11.0	14.0	15.0	18.0	11.0	13.0	22.0	11.0	10.0	...	17.(

2 rows × 551 columns

Apperçu du fichier key_1.csv

```
In [5]: test.head(2)
```

Out[5]:

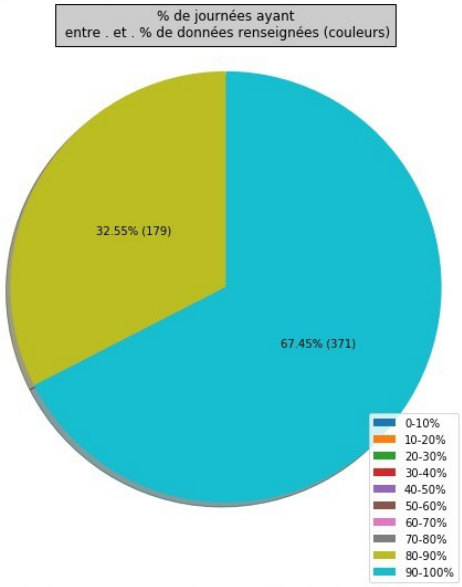
	Page	Id
0	!vote_en.wikipedia.org_all-access_all-agents_2...	bf4edcf969af
1	!vote_en.wikipedia.org_all-access_all-agents_2...	929ed2bf52b9

le fichier train_1.csv contient 145063 enregistrements de 551 colonnes et le fichier key_1.csv 8703780 enregistrements de 2 colonnes (qui consistent en fait aux Id des 60 prévisions des 145063 pages Wikipedia).

Taux de remplissage des journées

In [9]:
```
graphPourcentageNan()
```

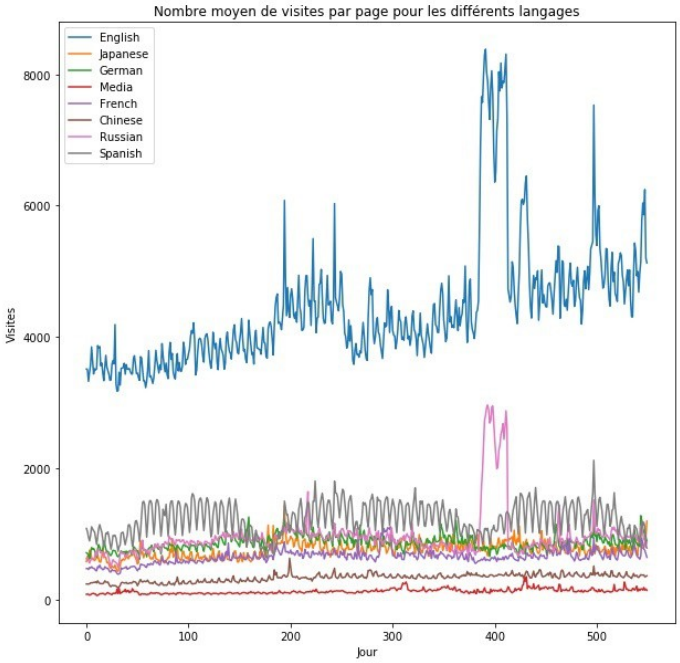
```
<matplotlib.figure.Figure at 0x1b3b0d1d6d8>
```



- Toutes les pages Wikipedia n'ont pas forcément de trafic sur les 18 mois, comme nous l'avons vu dans le notebook d'exploration, il y a une création presque constante de pages au fil des jours même si certaines des pages semblent cesser d'être disponibles au bout d'un moment (le pourcentage de pages sans données au 31 décembre 2016 est d'un peu plus de 2%).
- Comme on peut le voir sur le camembert, le pourcentage de données remplies est important puisqu'on a plus de 67% des journées qui ont un taux de remplissage de plus de 90% et les autres journées sont entre 80 et 90% de remplissage.

Evolution du nombre moyen de visites par page par langue

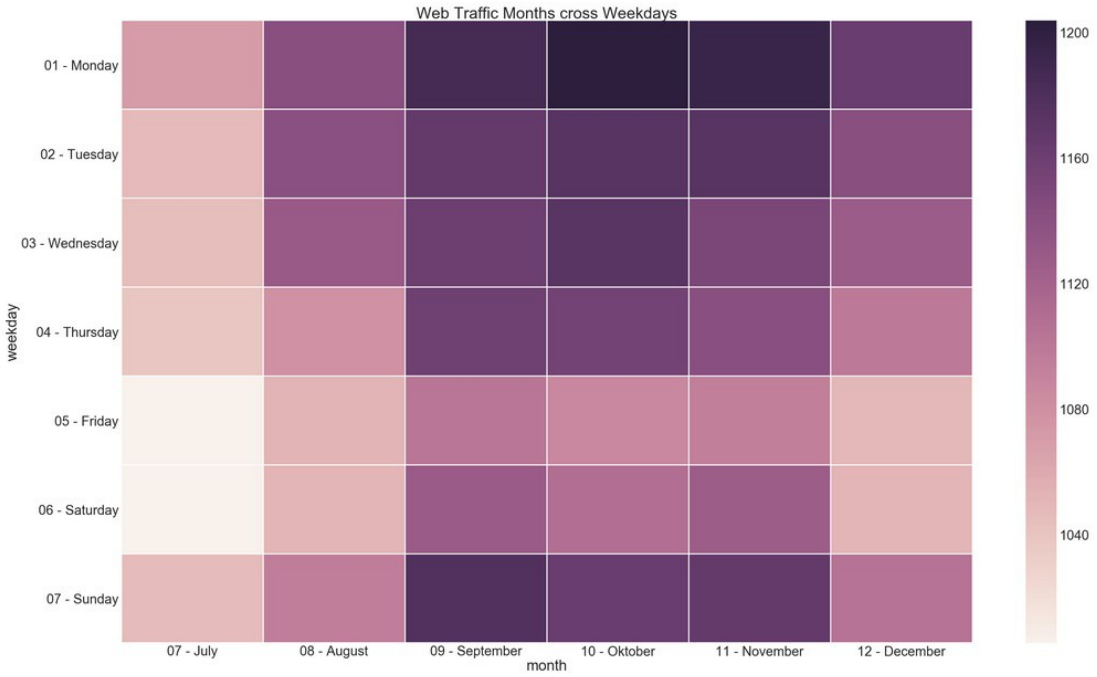
In [13]:
```
influenceLangage()
```



- la fréquentation des pages en anglais est globalement en augmentation constante
- on observe des pics autour de août 2016 et novembre 2016, peut-être pour les JO et les élections américaines
- le trafic des pages en espagnol semble observer des cycles
- le trafic sur les pages des autres langues semble plus régulier

Evolution du trafic global moyen par jour de la semaine et par mois

```
In [17]: crossWeekdaysMonth()
```



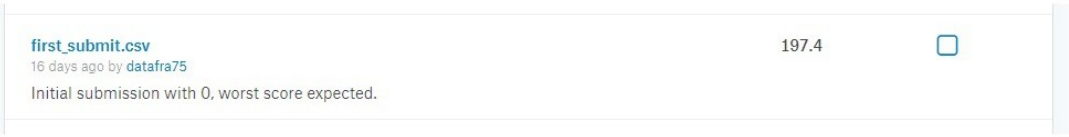
Il semble qu'il y ait une variation hebdomadaire avec un pic le lundi, puis une diminution progressive au cours de la semaine et enfin une remontée le dimanche.

3- Soumissions à Kaggle, modèles

Le principal objectif est donc d'obtenir le meilleur score possible.

Première soumission
Comme base de départ, on peut regarder ce que donne une soumission où on ne donne comme prédiction que la valeur 0:

```
In [ ]: submit_with_zeros()
```



Présentation de la mesure SMAPE (Symmetric mean absolute percentage error) avec At la valeur réelle et Ft la valeur prédite:

$$\text{SMAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \frac{|F_t - A_t|}{(|A_t| + |F_t|)/2}$$

Le SMAPE peut évoluer entre les valeurs 0 et 200 (du moment que les At et Ft ne sont pas égaux à 0):

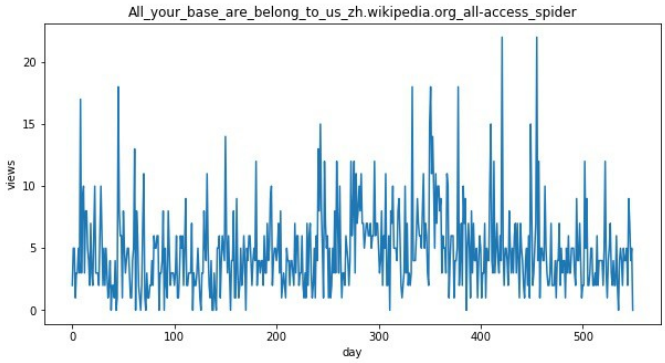
- quand pour toutes les valeurs, la prévision égale la valeur réelle, on obtient un score de 0
- à partir de l'inégalité triangulaire |Ft - At| <= |Ft| + |At|, on majore le SMAPE par 200
- 200 qu'on obtient par exemple si tous les At = 0 et Ft > 0 (ou symétriquement tous les Ft = 0 et At > 0)

Avec 197.4, on est proche du pire score de 200 que peut produire le SMAPE.
Toutefois, on observe que nous sommes a priori dans le cas relaté dans le troisième point, où tous les Ft = 0 et At > 0. Or le score ne vaut pas 200. On peut donc supposer qu'il existe, et en effet c'est bien le cas, des séries où tous les At = 0. En y adjoignant des Ft = 0, posant ainsi un problème de division par 0, on peut supposer que pour ces séries, le calcul du SMAPE n'a pas été fait et qu'on ne tient compte, dans le calcul général du SMAPE, que des séries qui possèdent un SMAPE défini et compris entre 0 et 200.

Soumission avec la moyenne

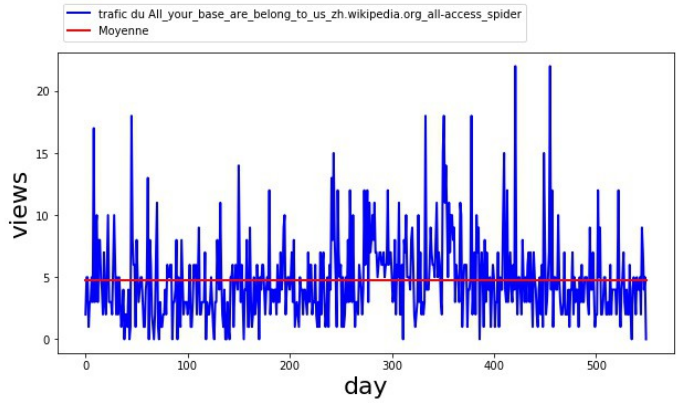
Comme on peut le voir dans l'exemple ci-dessous et dont on retrouve l'allure très régulièrement parmi les 145063 pages, le trafic des pages wikipedia varie énormément.

In [14]:
```
plot_entry(train, 12)
```



On peut alors essayer de limiter l'erreur de prédiction produite par la forte volatibilité du trafic en utilisant la moyenne.

In [76]:
```
plotTrafficAndMean(train, 12)
```



In [ ]:
```
submit_with_mean()
```

mean_submit.csv
15 days ago by datafra75
Mean submission
65.3

Comme on le voit avec le score de cette soumission, on a nettement amélioré le score en le faisant passer de 197.4 à 65.3. Peut-on faire mieux ?

Mais avant cela, peut-on se faire une idée de ce que représente 65.3 ? Le score est calculé sur deux mois, janvier et décembre. On peut regarder ce que cela donne sur les deux derniers mois novembre et décembre d'une série en faisant varier la prédiction en pourcentage des valeurs réelles.

Prenons par exemple comme prédiction, 5% en plus des valeurs réelles pour la série d'index train 12:

In [95]:
```
smape_fast(train.iloc[12,-60:].values, 1.05 * train.iloc[12,-60:].values)
```

Out[95]:
```
4.7154471544715495
```

On obtient un score assez proche de 0. On se rapproche d'un SMAPE de 65 en doublant les valeurs réelles (ou en les divisant par 2, le SMAPE est symétrique vis-à-vis d'un pourcentage au-dessus et en dessous des valeurs réelles).

In [101]:
```
print(smape_fast(train.iloc[12,-60:].values, train.iloc[12,-60:].values * 2))
print(smape_fast(train.iloc[12,-60:].values, train.iloc[12,-60:].values / 2))
```

```
64.4444444444
64.4444444444
```

En fait, lorsqu'on prend comme prédiction un multiple X des valeurs réelles, on élimine les valeurs (At et Ft) pour trouver comme SMAPE un coefficient C qui ne dépend que du multiple, coefficient qui vaut C(X) = 200 *|1-X| / (1+X). (Lorsqu'on remplace X par 1/X, on obtient bien le même coefficient C(1/X) = c(X).)*
On aurait donc dû trouver 66.66 comme SMAPE pour le multiple 2. Toutefois, le SMAPE ne prend pas en compte les couples (At, Ft) = (0, 0), ce qui fait qu'on obtient un score de C(2) (60 - N) / 60 si N est le nombre de valeurs 0. Et comme dans notre exemple, N = 2, on a bien (58/60)*200*(1/3) = 64.44.

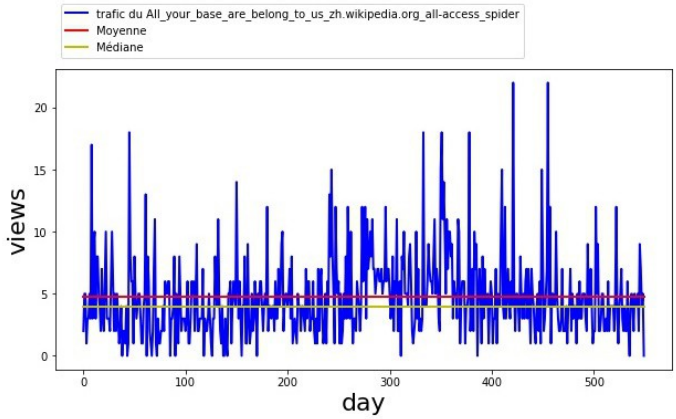
In [112]:
```
print("Nombre de valeurs 0 dans la Time Serie n°12 : {0}".format(np.unique(train.iloc[12,-60:].values, return_counts=True)[1]
[0]))
```

```
Nombre de valeurs 0 dans la Time Serie n°12 : 2
```

On peut donc supposer, qu'avec un score de 65.3 sur janvier/février, on est en moyenne sur un rapport de 2 ou de 1/2 sur les TS. Mais cela reste une hypothèse très générale, même si cela permet de se donner une idée du rapport réalité / prédiction pour ce score.

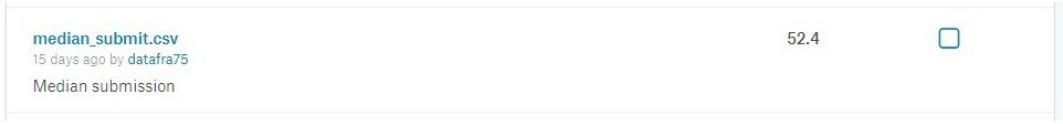
Soumission avec la médiane

In [114]:
```
plotTrafficAndMedian(train, 12)
```



Dans la plupart des pages, il y a pas mal de "sursauts" qui dépassent de beaucoup en valeur la plupart des autres journées, ce qui fait qu'ils ont tendance à élever la moyenne. Toutefois, ces sursauts ne sont pas non plus majoritaires, on a ainsi une médiane plus basse qui semble mieux coller à l'ensemble des données dans cet exemple.

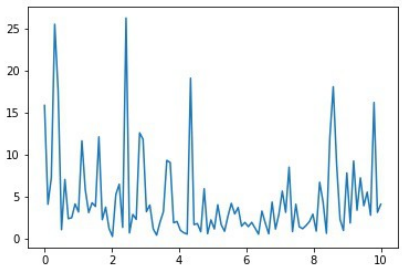
In [ ]:
```
submit_with_median()
```



La soumission donne un meilleur score sur Kaggle avec 52.4 au lieu de 65.3.

Un des participants a fait quelques tests d'étude du SMAPE, et il note qu'en effet, la médiane est souvent proche du meilleur résultat:
https://www.kaggle.com/cpmpml/smape-weirdness
Par exemple, en simulant une loi log normale qui pourrait très bien passer pour une distribution du trafic d'une page wikipedia:

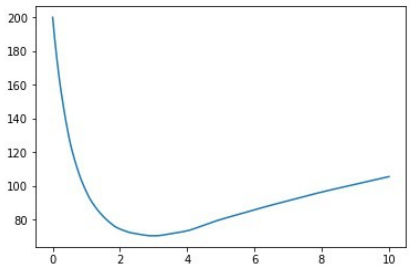
In [139]:
```
grapheLogNormal()
```



il a tracé l'évolution du SMAPE

In [140]:
```
studySMAPE()
```

```
SMAPE min:70.46  at 3.07
SMAPE is :70.49  at median 2.99
SMAPE is :78.37  at mean 4.71
```

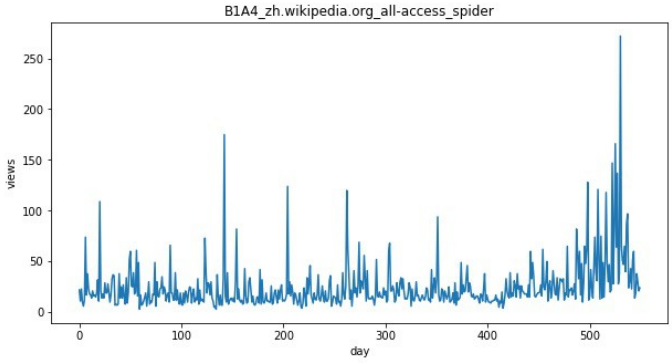


On voit en effet que le SMAPE au niveau de la médiane est proche du minimum, contrairement au SMAPE pour la moyenne. (Cela nécessiterait toutefois une étude plus approfondie qui ne se base que sur quelques exemples.)

Soumission avec la médiane restreinte aux 56 derniers jours de novembre/décembre

Il arrive que dans certains cas il y ait un changement important dans les derniers mois, par exemple une hausse substantielle du trafic comme avec l'exemple suivant. Prendre la médiane sur la totalité des 18 mois ne semble alors pas un bon calcul.

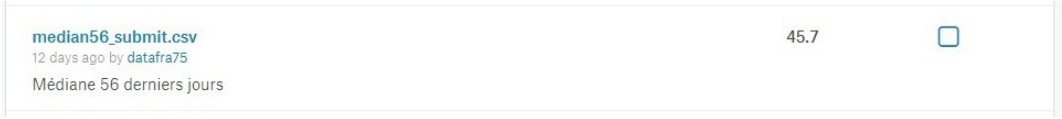
In [144]:
```
plot_entry(train, 20)
```



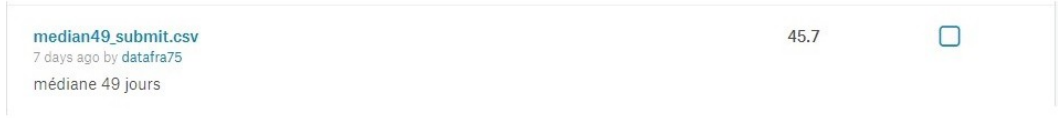
On peut alors expérimenter si en ne prenant la médiane que sur les X derniers jours du jeu de données, on n'obtient pas de meilleurs résultats, ce qui indiquerait que cette période est plus significative que le reste de l'année pour la prévision sur janvier/février.

En faisant varier X, on obtient le meilleur score pour X = 56 et X = 49, soit un score de 45.7.

In [ ]:
```
submit_with_median_56()
```



(son équivalent avec 49 jours)



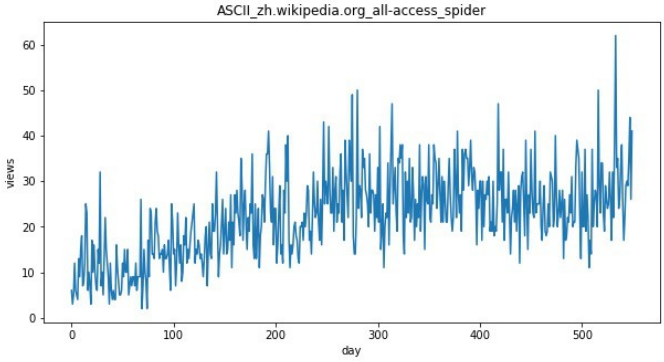
Il faut toutefois remarquer qu'il s'agit d'une recherche ad hoc se basant sur la connaissance du résultat donné par Kaggle. Que pour janvier/février, c'est la médiane sur les 56 (ou 49) derniers jours qui fonctionne le mieux ne signifie pas que cela soit encore le cas pour mars/avril 2017.

Alerte ! On peut alors se demander si le fait de jouer sur le score donné au fur et à mesure par Kaggle lors des différentes soumissions, ne consiste pas en une forme de **leakage** si jamais on choisissait de ne plus utiliser que les 56 derniers jours de données. Car on intègrerait dans le choix du modèle une donnée qu normalement on ne devrait pas connaître, celle du score de certains modèles sur les données à prédire.
Toutefois, Kaggle ne produit le score sur les soumissions en ne tenant compte que de 30% des données à prédire. La robustesse des modèles se découvrira alors lors du calcul sur les 70% restant.

Graphes avec une tendance marquée

On peut observer dans l'évolution du trafic de certaines pages Wikipedia des tendances marquées, contrairement à la page précédente. Par exemple la page ASCII_zh.wikipedia.org_all-access_spider:

In [13]:
```
plot_entry(train, 9)
```



Pour ce genre de mouvement, si jamais la tendance à la hausse, qui se dessine à la fin, se poursuit, utiliser la médiane ne semble pas une bonne idée. On peut supposer que la tendance à la hausse va se poursuivre puisqu'elle s'est déjà poursuivie sur les 280 premiers jours. Cependant, prévoir une continuité de la hausse sur les mois de janvier/février comporte un risque si le trafic se stabilise ou commence à régresser. Mais c'était également le cas même avec la médiane, car il suffisait que pour janvier et février il y ait une brusque croissance du trafic ou symétriquement, une chute vertigineuse, pour que la médiane donne un SMAPE très mauvais. En fait, de manière générale, logiquement, toute méthode qui poursuit une tendance obtiendra de bons résultats si la tendance est poursuivie, et de mauvais s'il y a un changement net de tendance. La moyenne, la médiane et Prophet (que nous allons voir) sont trois méthodes qui suivent la tendance.

Dans le cas où on supposerait que la "courbe" du trafic va continuer crescendo, il existe une méthode pour simuler la suite de la hausse, il s'agit de la méthode utilisant la librairie fbprophet (utilisée par exemple par Facebook).

Il ne sera toutefois donné qu'une illustration car si cette méthode est intéressante pour quelques séries temporelles, son temps d'exécution est suffisamment élevé pour qu'avec mon ordinateur je ne puisse pas produire facilement le fichier à soumettre à Kaggle (près de 10 heures). De toute manière, il semble, d'après certains échos que j'en ai eus, qu'elle ne fonctionne pas bien pour la totalité des 145063 time series. C'est une méthode utile si la très grande majorité des TS poursuit une tendance déjà engagée. Or quand on regarde les TS, il ne semble pas que cela soit le cas. J'ai toutefois fait quelques tests.

Pour cette illustration, j'ai choisi de prendre la TS ayant la moyenne de trafic sur les 18 mois la plus élevée de toutes les TS. Toutefois, cela ne signifie pas que ce haut volume de trafic impliquera un éventuel plus gros SMAPE relativement aux autres TS puisque si on veut comparer les TS entre elles, il faut prendre les erreurs en pourcentage des sérries et non en volume. Or si on prend les pourcentages, comme nous l'avons vu, la fonction |Ft - At| / (Ft + At) fait que la série est éliminée au profit du pourcentage seul. Ainsi, qu'on ait un gros volume de donnée ou non, ce qui compte c'est le pourcentage d'erreur vis-à-vis des valeurs réelles et non le volume de l'erreur.

In [4]:
```
print(getHighestCrowd(train))
```

```
(38573, 21930840.730418943)
```

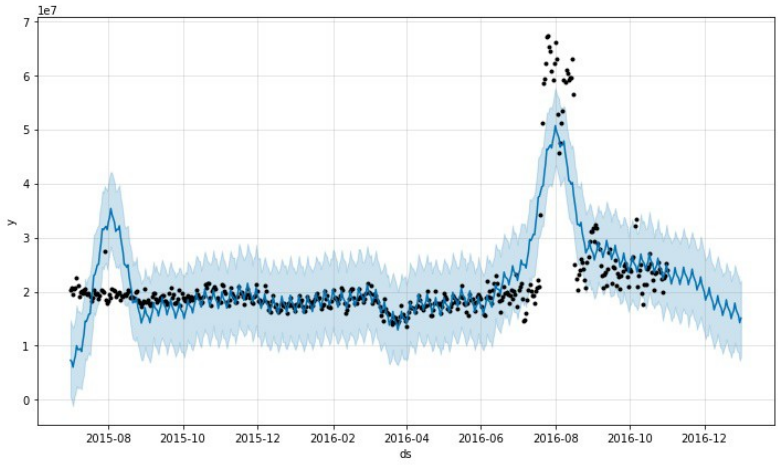
Prophet est une procédure de prévision des données d'une TS. Elle est basée sur un modèle où des tendances non linéaires prennent en compte des périodicités hebdomadaire et annuelle de même que les jours non ouvrés. Elle est normalement censée être robuste vis-à-vis des valeurs extrêmes.

Toutefois, comme nous allons le voir, il est préférable de gérer soi-même les valeurs extrêmes car elles influent beaucoup sur la tendance. (Ici est défini comme valeur extrême toute valeur Xi telle que |Xi- median({Xi}, i)| >= coeff * ecart-type({Xi}, i)

- Dans le premier cas, on laisse les valeurs extrêmes libres, on obtient un piètre score vis-à-vis des autres gestions
- Dans le deuxième cas, on remplace les valeurs extrêmes par la médiane sur toutes les valeurs du jeu de donnée
- Dans le troisième cas, on remplace les valeurs extrêmes par une médiane roulante paramétrable

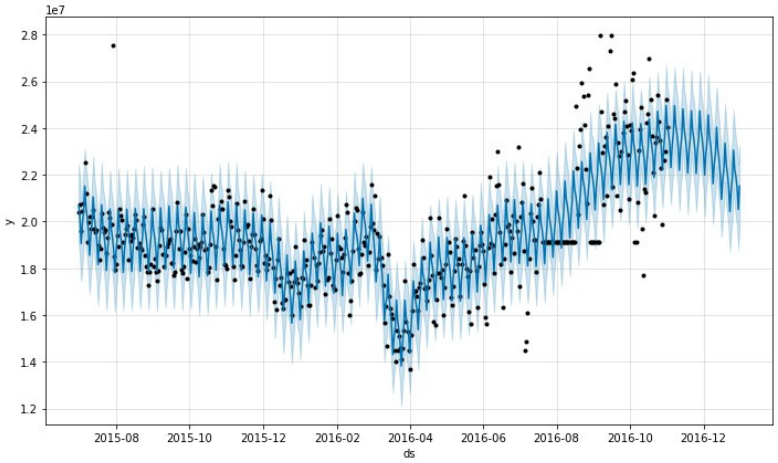
In [29]:
```
runProphet(38573, 0, 60, train, "let extreme live", 0, 1, True)
```

Out[29]:
```
20.963664128210439
```



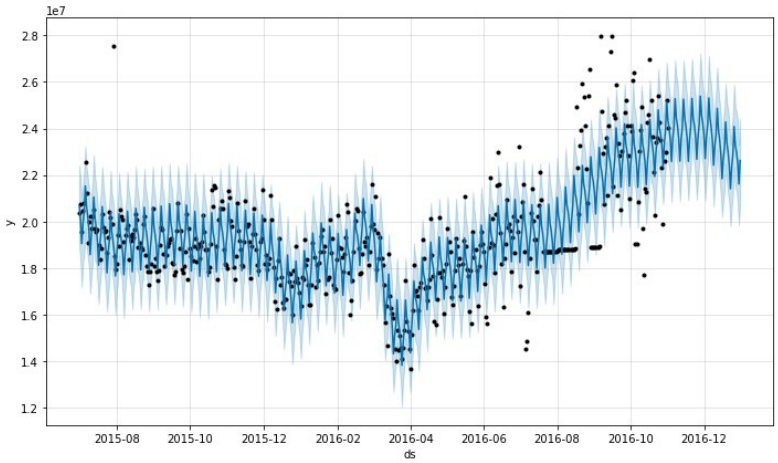
In [30]:
```
runProphet(38573, 0, 60, train, "median", 0, 1, True)
```

Out[30]: 7.8712355028308574



In [31]:
```
runProphet(38573 , 0, 60, train, "rolling_median", len(train.iloc[0,:]), 1, True)
```

Out[31]: 7.1363651935688956



In [32]:
```
for roll in [3, 10, 20, 40, 56, 70, 100, 300, len(train.iloc[0,:])]:
    print("Nombre jour de roulement : {0}, SMAPE : {1}".format(roll, runProphet(38573 , 0, 60, train, "rolling_median", roll,
1, False)))
```

```
Nombre jour de roulement : 3, SMAPE : 20.151776491284114
Nombre jour de roulement : 10, SMAPE : 8.272010692584516
Nombre jour de roulement : 20, SMAPE : 9.403986918664852
Nombre jour de roulement : 40, SMAPE : 8.291076274838952
Nombre jour de roulement : 56, SMAPE : 7.410111954048204
Nombre jour de roulement : 70, SMAPE : 7.858356741273077
Nombre jour de roulement : 100, SMAPE : 8.450225640092773
Nombre jour de roulement : 300, SMAPE : 7.469228948483646
Nombre jour de roulement : 551, SMAPE : 7.136365193568896
```

Comme on peut le voir sur l'exemple de cette TS, il est préférable de gérer les valeurs extrêmes car cela améliore nettement le score. (Il faudrait toutefois étudier cette influence sur la totalité des TS.) La méthode de la médiane roulante donne de meilleurs résultats du moment qu'on prend en compte un grand nombre de données. Mais ce n'est qu'un cas particulier, pour la TS n°10, c'est le contraire, on obtient de meilleurs résultats avec une médiane roulante sur 3 jours. Finalement, cela dépend de la configuration de la TS et puisqu'on ne peut pas faire l'optimisation pour chaque TS, surtout qu'on ne sait pas s'il n'y aura pas un changement abrupt de tendance en janvier/février, on pourra préférer non pas une médiane roulante, mais la médiane générale qui ne nécessite pas d'adaptation à chaque TS.

On peut également faire varier d'autres paramètres.

- Par exemple à quel degré considère-t-on qu'on a une valeur extrême ? au-delà de 1 std, 1.5 std, 3 std ?

In [41]:
```
for coeff in [1, 1.5, 3, 5]:
    print("Coeff : {0}, SMAPE : {1}".format(coeff, runProphet(38573 , 0, 60, train, "median", 0, coeff, False)))
```

```
Coeff : 1, SMAPE : 7.871235502830857
Coeff : 1.5, SMAPE : 7.6561852930366925
Coeff : 3, SMAPE : 7.152675565161452
Coeff : 5, SMAPE : 10.929719769803079
```

In [44]:
```
for coeff in [1, 1.5, 3, 5]:
    print("Coeff : {0}, SMAPE : {1}".format(coeff, runProphet(100 , 0, 60, train, "median", 0, coeff, False)))
```

```
Coeff : 1, SMAPE : 46.37612188069574
Coeff : 1.5, SMAPE : 46.921588249787405
Coeff : 3, SMAPE : 47.148160001231766
Coeff : 5, SMAPE : 48.375796906136614
```

Si pour la TS 38573, il vaut mieux prendre un coeff de 3, pour la TS 100, c'est 1. Cela varie en fonction de l'allure des TS.

- On peut également jouer sur la "frontière" entre données d'entraînement et données de test. En effet, pour vérifier l'adéquation entre la prévision de Prophet et les données réelles, on ne peut qu'utiliser des données connues. Or puisque Prophet perpétue une tendance, on ne doit pas prendre comme données d'entraînement et de test des données éparses sur tout le jeu de données. Il faut donc choisir une "frontière" entre données d'entraînement et données de test. Comme on sera amené à prévoir sur les deux mois de janvier et février, j'ai choisi de faire de même en prenant comme frontière 60 jours avant la fin du jeu de données.

- De même, on peut jouer sur le début des données à prendre en compte. Prend-on la totalité des 18 mois de données, ou simplement la moitiée, les 2 derniers mois ? En fait, il faudrait quasiment évaluer pour chaque série l'allure générale de la courbe, et choisir en connaissance de cause les bons paramètres. Mais c'est un algorithme qui prendrait trop de temps sur ma machine pour la totalité des TS. Puisque Prophet possède deux paramètres de "saisonnalité" hebdomadaire et annuelle, on peut jouer sur ces paramètres.

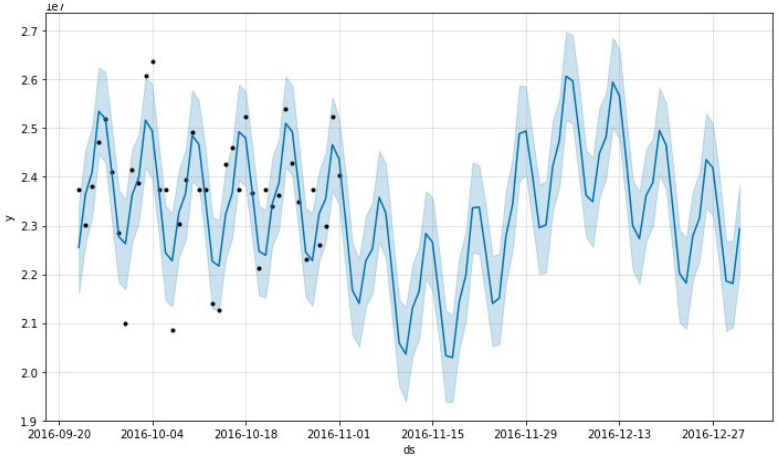
In [47]:
```
for start in [0, 100, 200, 450]:
    print("Start : {0}, SMAPE : {1}".format(start, runProphet(38573 , start, 60, train, "median", 0, 1, False)))
```

```
Start : 0, SMAPE : 7.871235502830857
Start : 100, SMAPE : 9.9793061843766
Start : 200, SMAPE : 9.01824883549624
Start : 450, SMAPE : 7.349719091095936
```

Il est ici préférable de ne prendre que les 100 derniers jours

In [50]:
```
runProphet(38573 , 450, 60, train, "median", 0, 1, True)
```

Out[50]: 7.3497190910959356



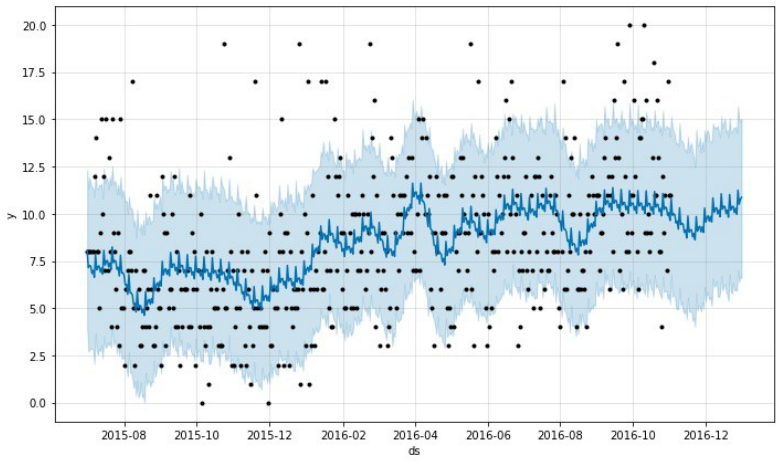
In [49]:
```
for start in [0, 100, 200, 450]:
    print("Start : {0}, SMAPE : {1}".format(start, runProphet(200 , start, 60, train, "median", 0, 1, False)))
```

```
Start : 0, SMAPE : 37.55561395878672
Start : 100, SMAPE : 40.127142900696164
Start : 200, SMAPE : 49.604432715416706
Start : 450, SMAPE : 66.44166002003969
```

Alors qu'ici il vaut nettement mieux prendre en compte le tout début.

In [52]:
```
runProphet(200 , 0, 60, train, "median", 0, 1, True)
```

Out[52]: 37.555613958786722



Finalement, on observe donc que Prophet peut s'adapter à un grand nombre de paramètres, mais que ces paramètres doivent être finement déterminés grâce à l'allure générale de la TS. Cela nécessite donc un travail précis sur chacune des 145063 TS et donc (comme d'habitude) un compromis temps d'exécution / précision. Si on veut aller au moins lent, on peut choisir d'utiliser les mêmes paramètres pour la totalité des TS, par exemple:

- coeff : 1
- start : 0
- weekly_seasonality : True
- yearly_seasonality : True
- valeurs extrêmes : médiane

Je n'ai pas pu faire d'essai sur la totalité des TS en soumission à Kaggle. Mais j'ai lu sur les commentaires que Prophet ne donnait pas de bons résultats. Il n'était néanmoins pas indiqué quel paramétrage avait été effectué.

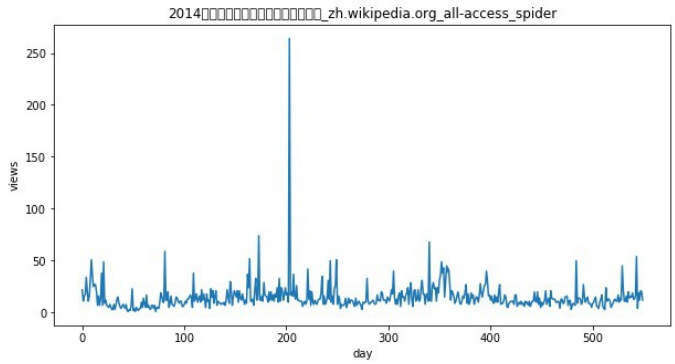
Une librairie tendance : statsmodels.api

Nous avons vu que Prophet se basait sur les tendances des TS. On peut alors essayer de dégager ces tendances en utilisant la librairie statsmodels.api.

L'idée est que, puisque la médiane donne de meilleurs résultats, on peut "l'aider un peu" en lui ajoutant une ou des valeurs dérivées de la dernière tendance que suit la TS.

Prenons par exemple la TS 750:

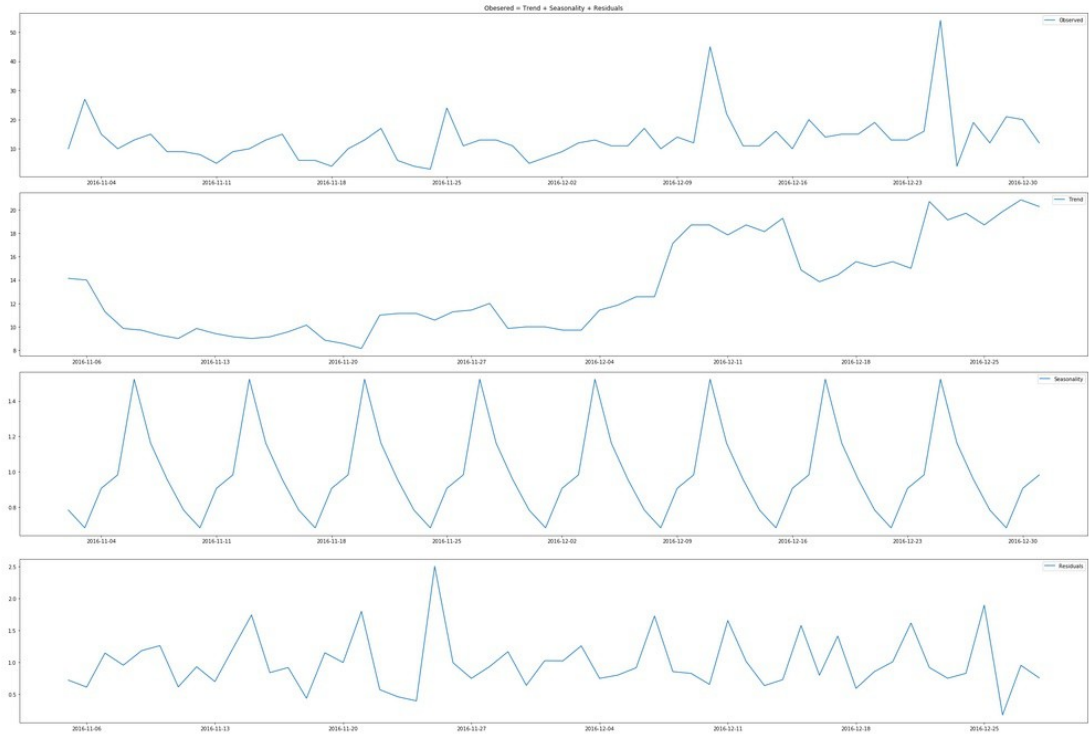
In [53]:
```
plot_entry(train, 750)
```



On note sur la fin une légère tendance à l'augmentation, statsmodels.api pourra-t-elle la déceler ?

Puisqu'il s'agit de repérer la dernière tendance, il n'est pas nécessaire d'utiliser pour ce point-ci la totalité des données. On peut prendre par exemple les 60 dernières journées.

In [61]:
```
trend = displayTrendSeasonalResidual(750)
```

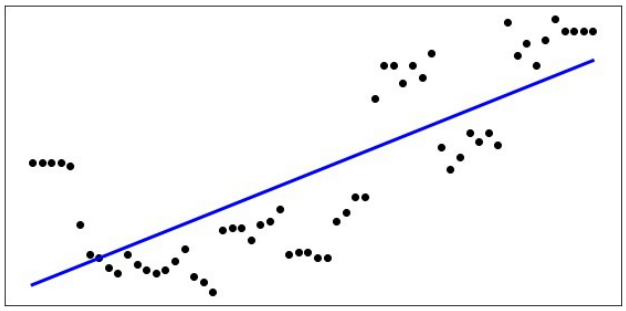


On peut appliquer deux modèles pour cette méthode, l'additive pour les cas linéaire et la multiplicative pour les cas non linéaire. Dans notre cas, c'est donc le modèle multiplicatif qui est choisi (cf. code)
Si on appelle T la tendance, S la saisonnalité et e l'erreur ou le résidu, on décompose la TS Y comme suit :

- Modèle additif : Y[t] = T[t] + S[t] + e[t]
- Modèle multiplicatif : Y[t] = T[t] x S[t] x e[t]

On voit ainsi une tendance à la hausse se démarquer sur le dernier mois. Il faudrait maintenant pouvoir la quantifier.

In [67]:
```
coefTrend = getRegCoef(trend)
```



```
On obtient un coefficient de régression linéaire de 0.17719604652085902
```

Il y a alors deux possibilités. Soit utiliser le coefficient de régression linéaire sur la tendance elle-même, soit l'établir directement sur les données. Dans le premier cas, j'ai testé comme prévision:

- médiane sur les 56 dernier jours + 30 x le coefficient sur la tendance

(dans un premier temps, puisqu'il y a 60 jours à prévoir, j'ai pris une prévision moyenne à 30 jours)

In [ ]:
```
submitWithMedianAndTrend30()
```

median_trend_submit.csv 10 days ago by datafra75 médiane avec 30 * moitié tendance	54.0

On n'obtient pas d'amélioration. Mais on a fait une régression sur la tendance et non sur les données elles-mêmes. Toutefois, dans les faits, on obtient exactement le même résultat.

In [ ]:
```
submitWithMedianAndPureTrend30()
```

median_pure_trend_submit.csv 9 days ago by datafra75 Médiane avec tendance des données directe, incrémentée de la tendance à 30 jours	54.0	☐

J'ai alors fait d'autres essais :

- On incrémente/décrémente de la tendance progressivement, jour par jour à partir de la médiane :

In [ ]:
```
submitWithMedianAndPureTrendIncremental()
```

median_pure_trend_increm_submit.csv 9 days ago by datafra75 median_pure_trend_increm 2	55.1	☐

- j'ai augmenté la médiane d'un certain pourcentage en fonction de la tendance. Si la tendance était à la hausse, j'ai par exemple augmenté de 5% la médiane pour les prévisions, et si à la baisse, diminué de 5% la médiane pour les prévisions

In [ ]:
```
submitWithMedianAndPercentagePureTrend5pc()
```

median_trend_5pc_submit.csv 9 days ago by datafra75 median_trend_5pc	45.7	☐

On est revenu au meilleur score.

Diverses autres tentatives:

- 2% en suivant la tendance
- 5% en allant contre la tendance (ajout de 5% quand tendance à la baisse, retrait de 5% quand à la hausse)
- 10 % en suivant la tendance
- 20% en suivant la tendance

median_trend_2pc_submit.csv 8 days ago by datafra75 median_trend_2pc	45.7	☐
median_trend_moins_5pc_submit.csv 8 days ago by datafra75 median_trend_moins_5pc	46.3	☐
median_trend_10pc_submit.csv 8 days ago by datafra75 median_trend_10pc_2	46.2	☐
median_trend_20pc_submit.csv 8 days ago by datafra75 median_trend_20pc	48.3	☐

Le score n'a pas été amélioré.

Essai de blending

On a vu que jusqu'à présent, la médiane à 56 jours était le meilleur modèle avec un score à 45.7 (à égalité avec d'autres modèles). Mais c'est une tendance moyenne qui concerne la totalité des 145063 pages. Il est donc probable que pour certaines pages, d'autres modèles soient localement plus performants, comme peut-être la moyenne ou Prophet.

J'ai donc lancé la méthode runBlending3Models(_id, 56, train) sur les 1000 premières TS (car sur la totalité, cela était trop long pour mon ordinateur) et voici ce qu'on obtient avec les modèles suivants:

- Modèle 1 : médiane des 56 derniers jours
- Modèle 2 : moyenne des 56 derniers jours
- Modèle 3 : Prophet avec prévisions sur les 60 derniers jours

Remarque 1: il serait préférable d'éviter de prendre la médiane ou la moyenne sur les 56 derniers jours pour le problème de leakage décrit précédemment, on pourrait prendre par exemple 60.
Remarque 2: le SMAPE est calculé sur les 56 derniers jours de novembre/décembre pour la médiane et la moyenne, et pour les 60 derniers jours pour Prophet, puisque ce sont ces données dont nous disposons (et non pas celles de janvier/février de Kaggle).

996	24,78191034	25,31608189	24,99691295	24,7819103	1	0	0	0	0
997	51,54875173	64,81688634	51,23299541	51,2329954	0	0	1	0,31575632	0
998	43,75038545	55,6843793	43,677416	43,677416	0	0	1	0,07296945	0
999	35,89941437	43,85884553	36,72330219	35,8994144	1	0	0	0	0
1000	45,7658785	47,92200395	46,80825034	45,7658785	1	0	0	0	0
1001	médiane	moyenne	prophet						
1002	40,43670042	45,3132914	46,84197293	40,1516839	860	26	111	5,95157356	2,72119063
1003									
1004								0,32587947	0,27059506
1005									

Résultats:

- Le modèle 1 de la médiane arrive en tête à 860 reprises
- Le modèle 2 de la moyenne arrive en tête à 26 reprises
- Le modèle 3 du Prophet arrive en tête à 111 reprises

(le total ne fait pas 1000 car il y a parmi les 1000 premières TS des TS sans aucune donnée)

Remarques:

- le SMAPE moyen des 1000 premières TS (on ne garde que le SMAPE de la meilleure méthode pour chaque TS) est de 40.15
- le SMAPE moyen de la médiane est de 40.43, celui de la moyenne est de 45.31 et pour prophet, 46.84
- quand la moyenne arrive en tête, le gain moyen est en moyenne de 0.33
- quand Prophet arrive en tête, le gain moyen est de 0.27

On peut donc en conclure que pour ces 1000 premières séries :

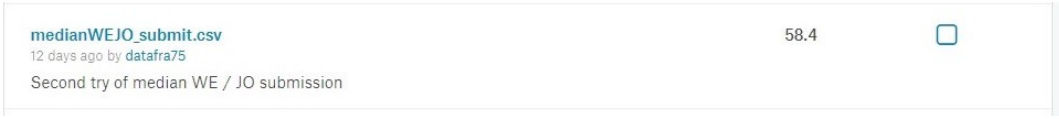
- le modèle de la médiane est nettement le meilleur puisqu'il arrive en tête pour 86% des TS et avec un SMAPE bien inférieur en moyenne
- moyenne et prophet sont assez proches, même si en moyenne prophet est un peu moins bon que le modèle moyenne, toutefois prophet est meilleur que la moyenne (et que la médiane) à 111 reprises pour seulement 26 pour la moyenne
- les gains de SMAPE apportés par les deux modèles (0.33 pour la moyenne et 0.27 pour prophet) ne sont pas très importants eu égard au temps supplémentaire très important que nécessite l'exécution de la méthode de blending. Cependant, avec une puissance de calcul suffisante, cette méthode devrait améliorer le SMAPE général sur les 145063 TS, sous réserve que ce qui se passe sur novembre/décembre se reproduise sur janvier/février. Or ça, rien ne le garantit. Nous verrons d'ailleurs plus loin que pour juste les deux méthodes moyenne et médiane, le blending n'améliore pas le score.

Gestion des jours ouvrés / WE

L'exploration a permis de voir qu'il y avait une variation de la fréquentation des sites suivant le jour de la semaine, et donc suivant qu'on est en WE ou non. On peut donc essayer de voir si la distinction WE et jour ouvré ne permettrait pas de mieux prévoir le trafic.

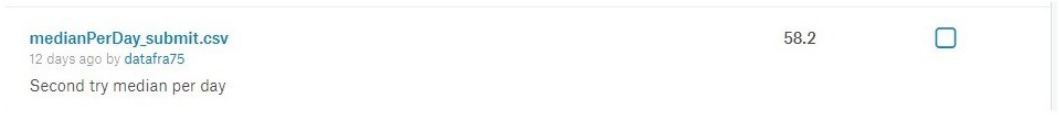
- Essai avec, sur la totalité des 18 mois de données, la distinction entre WE et jours ouvrés et application de la médiane respective sur chacun de ces 2 groupes (médiane sur les WE et médiane sur les jours ouvrés)

In [ ]:
```
submitWithMedianWeJo()
```



- Essai avec une médiane par jour de la semaine (médiane du lundi, médiane du mardi etc.) sur les 18 mois de données

In [ ]:
```
submitWithMedianPerDay()
```



- Essai avec une médiane par WE et par jours ouvrés, mais seulement sur les 56 derniers jours

In [ ]:
```
submitWithMedianWeJo56()
```

- Essai avec une médiane par jour de la semaine, mais seulement sur les 56 derniers jours

In [ ]:
```
submitWithMedianPerDay56()
```

Résultats:

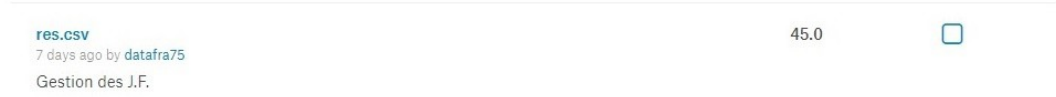
- la gestion des WE / jours ouvrés n'a pas permis d'améliorer le score, tout au plus de l'égaler, le fait de prendre uniquement les 56 derniers jours étant prioritaires.

Gestion des jours fériés

Puisque la gestion des WE / jours ouvrés n'a rien donné, on peut tenter de voir si les jours fériés propices au divertissement n'a pas une influence sur le trafic des pages Wikipedia.

Puisque les pages sont écrites en différents langages, anglais, américain, japonais, allemand, français, chinois, russe, espagnol etc. et que les jours fériés diffèrent entre les pays de ces différentes langues, il est nécessaire de bien les séparer et de leur attribuer des jours fériés particuliers (cf. code).

In [ ]:
```
submitWithMedianJoursFeries()
```

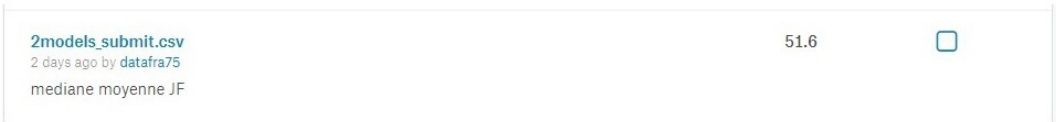


En ajoutant la gestion des jours fériés, on améliore le SMAPE en le passant de 45.7 à 45.0.

Soumission complète d'un blending moyenne / médiane

In [ ]:
```
submit2Models()
```

(En réalité, j'ai dû découper la fonction submit2Models par tranches de 2000 lignes du fichier train et faire tourner 4 notebooks en parallèle (au-delà de 4, je suis tombé en error memory) car exécuter submit2Models en une fois aurait pris près de 3 jours. En la découpant, le temps d'exécution a nettement diminué, même s'il a quand même atteint près de 10 heures.)



Malheureusement, cela n'a pas amélioré le score.

Quelques méthodes qui n'ont pas pu être soumises à Kaggle du fait d'un temps d'exécution trop long

ARIMA

Pour comprendre ce qu'est ARIMA:

Avant tout, on dit qu'une série temporaire est stationnaire (au sens faible) si ses propriétés statistiques ne varient pas dans le temps (espérance, variance, auto-corrélation).

La classe des modèles ARIMA (développés par Box et Jenkins, 1976) a été introduite pour reconstituer le comportement de processus soumis à des chocs aléatoires au cours du temps. Entre deux observations successives d'une série de mesures portant sur l'activité du processus, un événement aléatoire appelé perturbation vient affecter le comportement temporel de ce processus et ainsi modifier les valeurs de la série chronologique des observations.
Dans notre cas, cela peut être des événements comme les JO qui donnent envie aux gens de se renseigner grâce à Wikipedia.
Les modèles ARIMA permettent de combiner trois types de processus temporels :

- les processus autorégressifs (AR-AutoRegressive)
- les processus intégrés (I-Integrated),
- et les moyennes mobiles (MA-Moving Average).

Dans le cas le plus général, un modèle ARIMA combine les trois types de processus aléatoires à différents degrés.

- Les processus auto-régressifs:

Pour un processus autorégressif, chaque valeur de la série est une combinaison linéaire des valeurs précédentes de la série.
Si la valeur de la série à l'instant t, Y(t) ne dépend que de la valeur précédente Y(t−1) à une perturbation aléatoire près ε(t), le processus est dit autorégressif du premier ordre et noté AR(1) :
$Y(t) = \varphi 1 \times Y(t-1) + \varepsilon(t)$

Le coefficient φ exprime la force de la liaison linéaire entre deux valeurs successives.
Ainsi, si un processus autorégressif où la valeur de la série à l'instant t, Y(t), dépend des p précédentes valeurs, ce processus est dit d'ordre p et est noté AR(p). Ainsi un processus AR(2) s'écrit :
$Y(t) = \varphi 1 \times Y(t-1) + \varphi 2 \times Y(t-2) + \varepsilon(t)$
Un processus autorégressif possède donc une « mémoire » au sens où chaque valeur est corrélée à l'ensemble des valeurs qui la précède. Un processus autorégressif d'ordre p, AR(p), pourra être noté comme un modèle ARIMA(p,0,0).

- Les processus intégrés

Le comportement des séries chronologiques peut être affecté par l'effet cumulatif de certains processus. Une série chronologique déterminée par l'effet cumulatif d'une activité appartient à la classe des processus intégrés. Même si le comportement d'une série est erratique, les différences d'une observation à la prochaine peuvent être relativement faibles voire osciller autour d'une valeur constante pour un processus observé à différents intervalles de temps. Cette stationnarité de la série des différences pour un processus intégré est une caractéristique importante du point de vue de l'analyse statistique des séries chronologiques. Les processus intégrés constituent l'archétype des séries non stationnaires.
Un exemple de processus I(1), intégré d'ordre 1, est la marche aléatoire définie par : $Y(t) = Y(t-1) + \varepsilon(t)$
où la perturbation aléatoire ε(t) est un bruit blanc. On utilise le terme de marche aléatoire car la valeur courante est définie comme une étape aléatoire à partir de la valeur précédente. La marche aléatoire est également un processus autorégressif d'ordre 1, AR(1), dont le coefficient de régression φ1 est égal à 1. Ainsi, la marche aléatoire possède une « mémoire parfaite » mais limitée à l'observation précédente. Un processus est intégré d'ordre 1, noté I(1), si la série des différences premières est stationnaire. De même un processus est intégré d'ordre 2, noté I(2), si la série des différences secondes (les différences des différences) est stationnaire.
Un processus intégré d'ordre d, I(d), pourra être noté comme processus ARIMA(0,d,0).

- Les moyennes mobiles

La valeur courante d'un processus de moyenne mobile est définie comme une combinaison linéaire de la perturbation courante avec une ou plusieurs perturbations précédentes. L'ordre de la moyenne mobile indique le nombre de périodes précédentes incorporées dans la valeur courante. Ainsi, une moyenne mobile d'ordre 1, MA(1), est définie par l'équation suivante : $Y(t) = \varepsilon(t) - \theta 1 \times \varepsilon(t-1)$

Pour une moyenne mobile, chaque valeur est une moyenne pondérée des plus récentes perturbations tandis que pour un processus autorégressif c'est une moyenne pondérée des valeurs précédentes. L'effet d'une perturbation aléatoire décroît tout au long de la série au fur et à mesure que le temps s'écoule dans un processus autorégressif tandis que dans une moyenne mobile la perturbation aléatoire affecte la série temporelle pour un nombre fini d'observations (l'ordre de la moyenne mobile) puis au-delà cesse brutalement d'exercer une quelconque influence.

Pour utiliser le modèle ARIMA, il n'est donc pas nécessaire qu'une série soit stationnaire. Si on utilisait un modèle ARMA, cela serait nécessaire. On peut par exemple utiliser le test de Dickey-Fuller augmenté pour vérifier si une sétie est stationnaire ou non.

Reprenons pas exemple la TS 38573 qu'on va scinder en ses données d'entraînement (toutes les données sauf les 60 dernières) et ses données de test (les 60 derniers jours).

In [123]:
```
data_train38573 = headTSTrain(38573)
data_train38573.head()
```

Out[123]:

	Visits	date
0	20381245.0	2015-07-01
1	20752194.0	2015-07-02
2	19573967.0	2015-07-03
3	20439645.0	2015-07-04
4	20772109.0	2015-07-05

In [124]:
```
dickeyFullerAugmentedTest(data_train38573.Visits)
```

```
Test Statistic                  -2.821408
p-value                          0.055280
#Lags Used                      15.000000
Number of Observations Used    474.000000
Critical Value (1%)             -3.444221
Critical Value (5%)             -2.867657
Critical Value (10%)            -2.570028
dtype: float64
```

Ce test permet de décider s'il faut ou non rejeter une hypothèse. Dans ce test, on a l'hypothèse nulle:

- Hypothèse nulle (H0) : Si elle est acceptée, cela suggère que la TS a été générée par un processus présentant une racine unitaire, et donc, qu'elle n'est pas stationnaire, elle possède une dépendance au temps.
- Hypothèse alternative (H1): L'hypothèse nulle est rejetée; cela suggère que la TS n'a pas de racine unitaire et donc que la série est stationnaire. Elle n'a pas de structure dépendant du temps.

Comme on le voit dans les résultats, la p-value est supérieure à 0.05 et la valeur du test (-2.82) se situe entre les valeurs critiques de 5% et 10%. On peut donc accepter l'hypothèse nulle, la TS 38573 est non stationnaire.

On pourrait alors essayer de rendre la série stationnaire en utilisant par exemple le logarithme ou la racine carée...

In [125]:
```
logdata = np.sqrt(data_train38573.Visits)
dickeyFullerAugmentedTest(logdata)
```

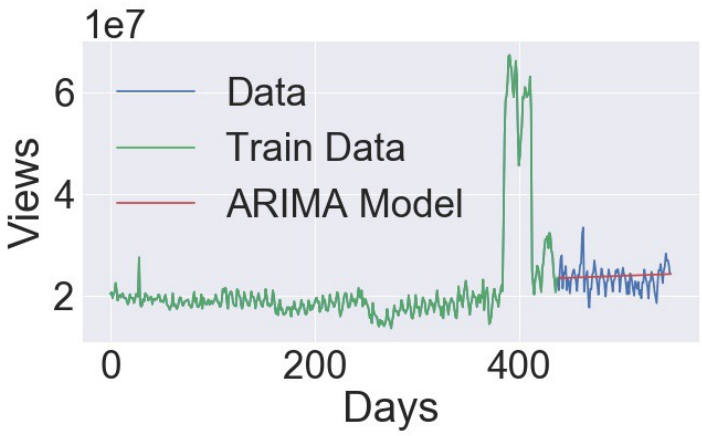
```
Test Statistic                  -2.745362
p-value                          0.066512
#Lags Used                      15.000000
Number of Observations Used    474.000000
Critical Value (1%)             -3.444221
Critical Value (5%)             -2.867657
Critical Value (10%)            -2.570028
dtype: float64
```

Cela ne rend pas la série pour autant stationnaire.

Mais cela n'a pas trop d'importance puisque avec ARIMA, il n'est pas nécessaire que la série le soit.
D'ailleur, comme on peut le voir, cette méthode donne un bon résultat pour la TS 38573:

In [131]:
```
arimarima(train, 38573)
```

```
smape =  7.853965239724074
```



On est avec un SMAPE de 7.85, à peu près dans les mêmes eaux que pour Prophet (7.87 pour la médiane et 7.13 pour la médiane mobile). Mais il ne s'agit que d'un cas, il faudrait tester sur la totalité des TS.

Réseaux de neurones

Je détaillerai plus en quoi consiste l'utilisation des réseaux de neurones lors de la soutenance.

In [137]:
```
trainPredict, testPredict, testId = runKeras(38573, 1)
```

In [138]:
```
smape_fast(testId[:-2].Visits.values, testPredict)
```

Out[138]:
```
array([ 0.76931655], dtype=float32)
```

Le SMAPE est assez exceptionnel (comparaison avec ARIMA et Prophet entre 7 et 8). Il faudrait approfondir sur les autres séries pour vérifier ce bon score vis-à-vis des autres méthodes.

Approche machine learning

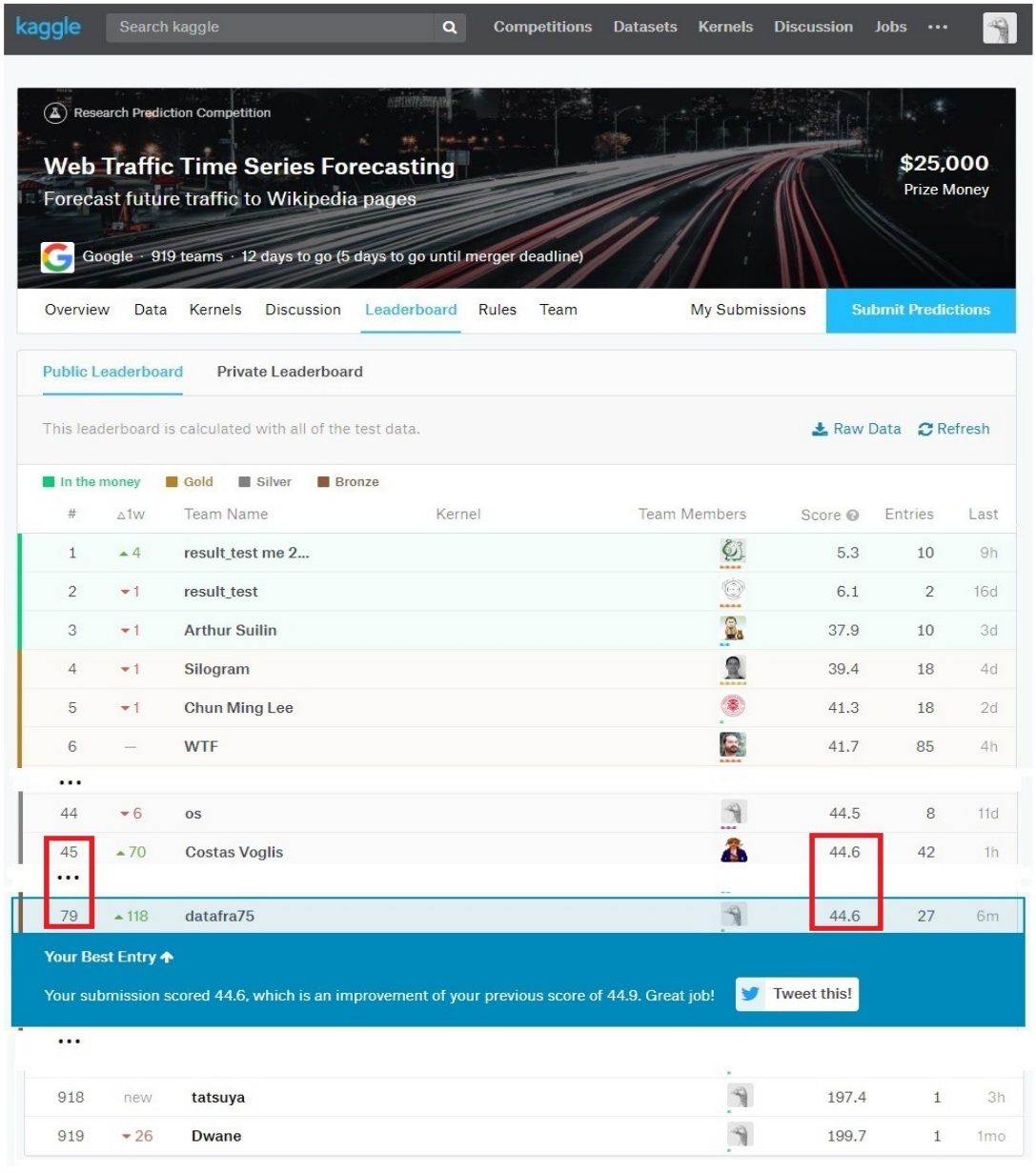
Il existe également une approche de machine learning qui consiste à prédire la différence de trafic entre deux journées successives. On génère ensuite un décalage pour que la valeur pour un certain jour se retrouve dans la colonne suivante mais décalée d'un certain nombre de jours. On a vu pendant l'exploration qu'il semblait y avoir un cycle de 7 jours (plus haut le lundi, baisse jusqu'au samedi, puis remontée le dimanche). On pourrait donc prendre un décalage de 1 jour et le répéter 7 fois. On peut ainsi essayer de voir si des relations apparaissent, par exemple en utilisant des modèles de régression linéaire, afin de prédire le futur comportement de la série, jour après jour.

4- Résultats

(voir la conclusion sous l'image du classement)

Finalement, voici la liste des soumissions de natures différentes qui ont été faites. Avec un score de 45.0, cela a situé DATAFRA75 (c'est moi) à la 138ème place (au moment où j'ai fait la copie écran), à 0.1 SMAPE de la 59ème place. Toutefois, ce classement continue d'évoluer car la compétition n'est pas encore terminée, et les scores ne sont actuellement pas définitifs car ils ne sont calculés que sur 30% des données afin d'éviter toute fuite de données. Un des partipants (parmi les 3 premiers) a d'ailleurs indiqué qu'il avait involontairement utilisé des informations concernant les données de janvier/février dans son modèle, ce qui avait bien fait améliorer son score.

Score soumission Kaggle	Modèle
197.4	que des 0
65.3	moyenne sur la totalité des jours
58.4	distinction médiane WE / jours ouvrés sur la totalité des jours
58.2	distinction médiane par jour de la semaine sur la totalité des jours
55.1	médiane incrémentée chaque jour par la tendance sur les données initiales
54.0	médiane avec tendance moyenne à 30 jours sur tendance calculée
54.0	médiane avec tendance moyenne à 30 jours sur données initiales
52.4	médiane sur la totalité des jours
51.6	blending des deux modèles médiane et moyenne
48.3	médiane sur les 56 derniers jours avec + / - 20 % suivant orientation de la tendance
46.6	médiane par jour de la semaine sur les 56 derniers jours
46.3	médiane sur les 56 derniers jours avec + / - 5% suivant l'inverse de l'orientation de la tendance
46.2	médiane sur les 56 derniers jours avec + / - 10 % suivant orientation de la tendance
45.7	médiane sur les 56 derniers jours avec distinction WE / jours ouvrés
45.7	médiane sur les 56 derniers jours avec + / - 5 % suivant orientation de la tendance
45.7	médiane sur les 56 derniers jours avec + / - 2 % suivant orientation de la tendance
45.7	médiane sur les 56 derniers jours
45.7	médiane sur les 49 derniers jours
45.0	médiane avec gestion des jours fériés



5- Conclusion

C'est donc avec une médiane des 49 derniers jours et la gestion des jours fériés / WE / jours ouvrés que j'ai obtenu mon meilleur score lors des différentes submissions sur Kaggle.

On peut faire quelques remarques:

- le manque de puissance de calcul de mon ordinateur m'a empêché de pouvoir tester différents algorithmes et méthodes existantes. J'ai tenté de lancer certains calculs (avec Prophet) directement sur le site Kaggle mais c'était encore plus lent. Il faudrait alors avoir recours à des serveurs à distance permettant de paralléliser les calculs (la décomposition de certaines méthodes afin de les faire marcher sur 4 notebooks Jupyter a bien divisé le temps d'exécution par un peu moins de 4 mais cela représentait à chaque fois une dizaine d'heure d'exécution...) Il faudrait donc passer des CPU aux GPU.
- nous avons vu lors d'une première approche qu'il faudrait approfondir, que la médiane semblait convenir à la configuration du SMAPE. Il faudrait alors étudier si à tout autre critère de calcul de correspondance entre prévision et réalité, d'autres modèles ne serait pas préférables.

Quelques pistes de recherche:

- pour améliorer les TS qui se terminent par des "plateaux" et dont on souhaite utiliser la médiane, il pourrait être intéressant de déterminer statistiquement le breakpoint afin de ne pas prendre en compte les données précédant le plateau
- les événements importants, comme les JO ou une élection, peuvent avoir une répercussion très forte sur le trafic. Il faudrait réussir à mettre en relation certains pics avec ces événements, voir dans quelle mesure ils interviennent, et intégrer dans les prévisions les dates d'événements prévus (coupe du monde, grande réunion internationale, évenement économique à venir etc.)
- on a vu dans l'exploration que l'évolution du trafic des pages différait en fonction des langages. Il pourrait être intéressant d'incorporer une gestion de la composante langue pour chaque page
- de même, on a vu dans l'exploration qu'il semblait y avoir une tendance générale de l'évolution du trafic le long de la semaine. L'intégration des jours fériés et des WE / jours ouvrés a permis une amélioration du score, mais il serait intéressant de prendre en partie en compte cette évolution générale dans le particulier (décroissance du lundi au samedi et remontée le dimanche)

Code

In [140]:
```
import pandas as pd
import numpy as np
from fbprophet import Prophet
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
from numba import jit
import math
import statsmodels.api as sm
from pylab import rcParams
from sklearn import linear_model
import re
import datetime
from matplotlib.gridspec import GridSpec
import seaborn as sns
from statsmodels.tsa.arima_model import ARIMA
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from keras.models import Sequential
from keras.layers import Dense

test = pd.read_csv("key_1.csv")
train = pd.read_csv("train_1.csv")
```

In [141]:
```
@jit
def smape_fast(y_true, y_pred):
    """ Méthode qui calcule rapidement le SMAPE
    Arguments:
    y_true -- les données réelles
    y_pred -- les données prévues
    Retour:
    out -- le score SMAPE
    """
    out = 0
    for i in range(y_true.shape[0]):
        a = y_true[i]
        b = y_pred[i]
        c = a+b
        if c == 0:
            continue
        out += math.fabs(a - b) / c
    out *= (200.0 / y_true.shape[0])
    return out

def submit_with_zeros():
    """ Création du fichier à soumettre à Kaggle avec que des 0
    Retour:
    first_submit.csv -- le fichier créé à uploader sur Kaggle
    """
    zero_serie = 0*np.ones(test.shape[0])
    dfTestZeros = pd.DataFrame({'Id': test.Id.values, 'Visits': zero_serie})
    dfTestZeros[['Id','Visits']].to_csv('first_submit.csv', index=False)
```

```python
def plot_entry(_data, _id):
    """ Méthode qui affiche le graphe de l'évolution du trafic pour une TS
    Arguments:
    _data -- l'ensemble des TS
    _id -- le numéro de la TS à afficher
    """
    days = [r for r in range(len(_data.iloc[_id,1:]))]
    fig = plt.figure(1,figsize=(10,5))
    plt.plot(days,_data.iloc[_id,1:])
    plt.xlabel('day')
    plt.ylabel('views')
    plt.title(train.iloc[_id,0])
    plt.show()

def plotTrafficAndMean(_data, _id):
    """ Méthode qui affiche le trafic d'une TS ainsi que la ligne représentant sa moyenne
    Arguments:
    _data -- l'ensemble des données
    _id -- l'index de la TS à afficher
    """
    x = [r for r in range(len(_data.iloc[_id,1:]))]
    y1 = _data.iloc[_id,1:]
    moy = y1.mean()
    y2 = moy * np.ones(len(_data.iloc[_id,1:]))

    fig = plt.figure(1,figsize=(10,5))
    ax = fig.add_subplot(111)
    ax.plot(x, y1,'-b', lw=2, label='trafic du '+_data.iloc[_id,0])
    ax.plot(x, y2,'-r', lw=2, label='Moyenne')
    ax.set_xlabel('day', size=22)
    ax.set_ylabel('views', size=22)
    ax.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3, borderaxespad=0.5)

    plt.show()

def submit_with_mean():
    """ Méthode qui crée le fichier à soumettre à Kaggle à partir du modèle de la moyenne
    """
    dfTest2 = test.copy()
    dfTrain2 = train.copy()
    dfTest2['Page'] = dfTest2.Page.apply(lambda a: a[:-11])
    dfTrain2['Visits'] = dfTrain2.drop('Page', axis=1).mean(axis=1, skipna=True)
    dfTest2 = dfTest2.merge(dfTrain2[['Page','Visits']], how='left')
    dfTest2.loc[dfTest2.Visits.isnull(), 'Visits'] = 0
    dfTest2.drop('Page', axis=1)
    dfTest2[['Id','Visits']].to_csv('mean_submit.csv', index=False)

def plotTrafficAndMedian(_data, _id):
    """ Méthode qui affiche le trafic d'une TS ainsi que la ligne représentant sa médiane
    Arguments:
    _data -- l'ensemble des données
    _id -- l'index de la TS à afficher
    """
    x = [r for r in range(len(_data.iloc[_id,1:]))]
    y1 = _data.iloc[_id,1:]
    moy = y1.mean()
    y2 = moy * np.ones(len(_data.iloc[_id,1:]))
    med = y1.median()
    y3 = med * np.ones(len(_data.iloc[_id,1:]))

    fig = plt.figure(1,figsize=(10,5))
    ax = fig.add_subplot(111)
    ax.plot(x, y1,'-b', lw=2, label='trafic du '+_data.iloc[_id,0])
    ax.plot(x, y2,'-r', lw=2, label='Moyenne')
    ax.plot(x, y3,'-y', lw=2, label='Médiane')
    ax.set_xlabel('day', size=22)
    ax.set_ylabel('views', size=22)
    ax.legend(bbox_to_anchor=(0., 1.02, 1., .102), loc=3, borderaxespad=0.5)

    plt.show()
def submit_with_median():
    """ Méthode qui crée le fichier à soumettre à Kaggle à partir du modèle de la médiane sur la totalité des jours
    """
    dfTest3 = test.copy()
    dfTrain3 = train.copy()
    dfTest3['Page'] = dfTest3.Page.apply(lambda a: a[:-11])
    dfTrain3['Visits'] = dfTrain3.drop('Page', axis=1).median(axis=1, skipna=True)
    dfTest3 = dfTest3.merge(dfTrain3[['Page','Visits']], how='left')
    dfTest3.loc[dfTest3.Visits.isnull(), 'Visits'] = 0
    dfTest3.drop('Page', axis=1)
    dfTest3[['Id','Visits']].to_csv('median_submit.csv', index=False)

def grapheLogNormal():
    """ Graphe de la loi log normale
    """
    np.random.seed(0)
    x = np.linspace(0,10,100)
    y = np.random.lognormal(1, 1, 100)
    plt.plot(x, y)
    plt.show()
```

```
def studySMAPE():
    """ Méthode d'étude du SMAPE
    """
    np.random.seed(0)
    y_true = np.random.lognormal(1, 1, 100)
    y_pred = np.ones(len(y_true))
    x = np.linspace(0,10,1000)
    res = [smape_fast(y_true, i * y_pred) for i in x]
    plt.plot(x, res)
    print('SMAPE min:%0.2f' % np.min(res), ' at %0.2f' % x[np.argmin(res)])
    print('SMAPE is :%0.2f' % smape_fast(y_true, y_pred*np.nanmedian(y_true)), ' at median %0.2f' % np.nanmedian(y_true))
    print('SMAPE is :%0.2f' % smape_fast(y_true, y_pred*np.nanmean(y_true)), ' at mean %0.2f' % np.nanmean(y_true))

def submit_with_median_56():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane sur les 56 derniers jours
    """
    dfTest4 = test.copy()
    dfTrain4 = train.copy()
    dfTest4['Page'] = dfTest4.Page.apply(lambda a: a[:-11])
    dfTrain4['Visits'] = np.nan_to_num(np.round(np.nanmedian(dfTrain4.drop('Page', axis=1).values[:, -56:], axis=1)))
    dfTest4 = dfTest4.merge(dfTrain4[['Page','Visits']], how='left')
    dfTest4.loc[dfTest4.Visits.isnull(), 'Visits'] = 0
    dfTest4.drop('Page', axis=1)
    dfTest4[['Id','Visits']].to_csv('median56_submit.csv', index=False)
                                    _           _

def runProphet(_id, _start, _frontier, _data, _val_ext, _per_roll_med, _std_mult, _plot):
    """ Méthode qui lance Prophet pour une TS
    Arguments:
    _id -- l'index de la TS
    _start -- index du jour de départ des données d'entraînement
    _frontier -- limite entre le données d'entraînement et les données de test (souvent 60)
    _data -- l'ensemble des données
    _val_ext -- choix de la gestion des valeurs extrêmes
    _per_roll_med -- période de la médiane mobile
    _std_mult -- coefficient muliplicateur appliqué à l'écart-type
    _plot -- choix de l'affichage ou non du graphe
    Retour:
    score -- retourne le score de Prophet
    """
    _data=_data.iloc[_id,:]
    data_train, data_test = _data.iloc[_start:-_frontier], _data.iloc[-_frontier:]
    test_median = data_test.median()
    test_cleaned = data_test.T.fillna(test_median).T
    train_median = data_train.iloc[1:].median()
    train_cleaned = data_train.T.iloc[1:].fillna(train_median).T
    data=train_cleaned.iloc[:].to_frame()
    data.columns = ['visits']
    if _val_ext != "let extreme live":
        if _val_ext == "median":
            #fill outliers that are out of 1*std with median
            data['median'] = data.visits.median()
        if _val_ext == "rolling_median":
            #fill outliers that are out of 1*std with rolling median of 56 days
            data['median'] = pd.rolling_median(data.visits, _per_roll_med, min_periods=1)
        data.ix[np.abs(data.visits-data.visits.median())>=(_std_mult*data.visits.std()),'visits'] = data.ix[np.abs(data.visit
s-data.visits.median())>=(_std_mult*data.visits.std()),'median']
        data.index = pd.to_datetime(data.index)

    X = pd.DataFrame(index=range(0,len(data)))
    X['ds'] = data.index
    X['y'] = data['visits'].values
    m = Prophet(weekly_seasonality=True, yearly_seasonality=True)
    m.fit(X)
    future = m.make_future_dataframe(periods=_frontier)
    forecast = m.predict(future)
    if _plot:
        m.plot(forecast);
    y_truth = test_cleaned
    y_forecasted = forecast.iloc[-_frontier:,2].values
    score = smape_fast(y_truth, y_forecasted)
    return score

def getHighestCrowd(_df):
    """ Méthode qui recherche l'index de la TS ayant la moyenne de trafic la plus élevée
    Arguments:
    _dfTrain -- l'ensemble des données
    Retour:
    itop -- index de la TS ayant la moyenne de trafic la plus élevée
    top -- moyenne la plus élevée
    """
    itop = 0
    top = 0
    for i in range(_df.shape[0]):
        datai = _df.iloc[i,1:-1].values
        moy = np.sum(datai)/len(datai)
        if moy > top:
            top = moy
            itop = i
    return itop, top
```

```
def displayTrendSeasonalResidual(_id):
    """ Méthode qui affiche la décomposition en tenance / saisonnalité / résidus d'une TS
    Arguments:
    _id -- index de la TS
    Retour:
    trend -- retourne la tendance
    """
    data =train.iloc[_id,-60:].to_frame()
    data.reset_index(drop=False,inplace=True)
    data.columns = ['date', 'visits']
    data['date'] = pd.to_datetime(data['date'])
    data.set_index('date', inplace=True)
    data['visits'] = data['visits'].astype(float)

    # décomposition des données en tendance, saisonnalité et résidus
    #import statsmodels.api as sm
    decomposition = sm.tsa.seasonal_decompose(data, model='multiplicative',freq = 7)
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    #from pylab import rcParams
    rcParams['figure.figsize'] = 30, 20

    plt.subplot(411)
    plt.title('Obesered = Trend + Seasonality + Residuals')
    plt.plot(data, label='Observed')
    plt.legend(loc='best')
    plt.subplot(412)
    plt.plot(trend, label='Trend')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal,label='Seasonality')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residuals')
    plt.legend(loc='best')
    plt.tight_layout()
    plt.show()
    return trend

def getRegCoef(_trend):
    """ Méthode qui détermine le coefficient de régression de la tendance
    Arguments:
    _trend -- les données de la tendance
    Retour:
    print -- affiche le coefficient
    """
    _trend = _trend.fillna(method='bfill').fillna(method='ffill')
    X = np.arange(len(_trend))
    Y = _trend['visits'].values
    X = X.reshape(len(_trend), 1)
    Y = Y.reshape(len(_trend), 1)
    #from sklearn import linear_model
    regr = linear_model.LinearRegression()
    regr.fit(X, Y)
    fig = plt.figure(1,figsize=(10,5))
    plt.scatter(X, Y,  color='black')
    plt.plot(X, regr.predict(X), color='blue', linewidth=3)
    plt.xticks(())
    plt.yticks(())
    plt.show()
    print("On obtient un coefficient de régression linéaire de {0}".format(regr.coef_[0][0]))

def getTrendCoeff(row):
    """ Méthode qui détermine le coefficient de régression de la tendance, applicable à chaque TS
    Arguments:
    row -- toutes les données de la TS
    Retour:
    coef_ -- le coefficient de la régression de la tendance
    """
    trainId =row[-56:]
    if trainId.count() > 0:
        trainIdMedian = trainId.median()
        trainId.ix[np.abs(trainId-trainIdMedian)>=1.5*trainId.std()] = trainIdMedian
        trainId =trainId.to_frame()
        trainId = trainId + 1
        trainId.reset_index(drop=False,inplace=True)
        trainId.columns = ['date', 'visits']
        trainId['date'] = pd.to_datetime(trainId['date'])
        trainId.set_index('date', inplace=True)
        trainId['visits'] = trainId['visits'].astype(float)
        trainIdMedian = trainId.median()
        trainId.fillna(trainIdMedian, inplace=True)
        decomposition = sm.tsa.seasonal_decompose(trainId, model='multiplicative',freq = 7)
        trend = decomposition.trend
        trend = trend.fillna(method='bfill').fillna(method='ffill')
        X = np.arange(len(trend))
        Y = trend['visits'].values
        X = X.reshape(len(trend), 1)
        Y = Y.reshape(len(trend), 1)
        regr = linear_model.LinearRegression()
        regr.fit(X, Y)
        return regr.coef_[0][0]
    else:
        return 0
```

```
def submitWithMedianAndTrend30():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane à laquelle on ajoute 30* la valeur de l
a tendance
    """
    dfTest9 = test.copy()
    dfTrain9 = train.copy()
    dfTest9['Page'] = dfTest9.Page.apply(lambda a: a[:-11])
    dfTrain9['trend'] = dfTrain9.apply(getTrendCoeff, axis = 1)
    dfTrain9['Visits'] = np.nan_to_num(np.round(np.nanmedian(dfTrain9.drop('Page', axis=1).values[:, -56:], axis=1)))+30*dfTr
ain9['trend']
    dfTest9 = dfTest9.merge(dfTrain9[['Page','Visits']], how='left')
    dfTest9.loc[dfTest9.Visits.isnull(), 'Visits'] = 0
    dfTest9.drop('Page', axis=1)
    dfTest9[['Id','Visits']].to_csv('median_trend_submit.csv', index=False)

def getPureTrendCoeff(row):
    """ Méthode qui détermine le coefficient de régression des données directement, applicable à chaque TS
    Arguments:
    row -- toutes les données de la TS
    Retour:
    coef_ -- le coefficient de la régression des données directement
    """
    trainId =row[-56:]
    if trainId.count() > 0:
        trainIdMedian = trainId.median()
        trainId.ix[np.abs(trainId-trainIdMedian)>=1.5*trainId.std()] = trainIdMedian
        trainId =trainId.to_frame()
        trainId = trainId + 1
        trainId.reset_index(drop=False,inplace=True)
        trainId.columns = ['date', 'visits']
        trainId['date'] = pd.to_datetime(trainId['date'])
        trainId.set_index('date', inplace=True)
        trainId['visits'] = trainId['visits'].astype(float)
        trainIdMedian = trainId.median()
        trainId.fillna(trainIdMedian, inplace=True)
        lenTrainId = len(trainId)
        X = np.arange(lenTrainId)
        Y = trainId['visits'].values
        X = X.reshape(lenTrainId, 1)
        Y = Y.reshape(lenTrainId, 1)
        regr = linear_model.LinearRegression()
        regr.fit(X, Y)
        return regr.coef_[0][0]
    else:
        return 0

def submitWithMedianAndPureTrend30():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane à laquelle on ajoute 30 fois la tendanc
e des données pures
    """
    dfTest10 = test.copy()
    dfTrain10 = train.copy()
    dfTest10['Page'] = dfTest10.Page.apply(lambda a: a[:-11])
    dfTrain10['trend'] = dfTrain10.apply(getPureTrendCoeff, axis = 1)
    dfTrain10['Visits'] = np.nan_to_num(np.round(np.nanmedian(dfTrain10.drop('Page', axis=1).values[:, -56:], axis=1)))+30*df
Train10['trend']
    dfTest10 = dfTest10.merge(dfTrain10[['Page','Visits']], how='left')
    dfTest10.loc[dfTest10.Visits.isnull(), 'Visits'] = 0
    dfTest10.drop('Page', axis=1)
    dfTest10[['Id','Visits']].to_csv('median_pure_trend_submit.csv', index=False)

def submitWithMedianAndPureTrendIncremental():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane à laquelle on ajoute un incrément par j
our
    """
    #import math
    dfTest11 = test.copy()
    dfTrain11 = train.copy()
    dfTest11['date'] = dfTest11.Page.apply(lambda a: a[-10:])
    dfTest11['Page'] = dfTest11.Page.apply(lambda a: a[:-11])
    dfTrain11['trend'] = dfTrain11.apply(getPureTrendCoeff, axis = 1)
    dfTrain11['Visits'] = np.nan_to_num(np.round(np.nanmedian(dfTrain11.drop('Page', axis=1).values[:, -56:], axis=1)))
    dfTest11 = dfTest11.merge(dfTrain11[['Page','Visits', 'trend']], how='left')
    dfTest11.loc[dfTest11.Visits.isnull(), 'Visits'] = 0
    dfTest11 = dfTest11.reset_index(drop=False)
    dfTest11['pos'] = (dfTest11['index']%60)+1
    dfTest11['Visits'] = dfTest11['Visits'] + dfTest11['pos']*dfTest11['trend']
    dfTest11['Visits'] = dfTest11.Visits.apply(lambda a: (math.floor(a * 1000))/1000) # pour ne pas que le fichier soit trop
lourd
    dfTest11.drop('Page', axis=1)
    dfTest11[['Id','Visits']].to_csv('median_pure_trend_increm_submit.csv', index=False)
```

```
def applyCoeff(row):
    """ Méthode qui applique à une TS un coefficient à la médiane
    Arguments:
    row -- toutes les données de la TS
    Retour:
    coeff*median -- la médiane multipliée par un coefficient
    """
    part = row.values[-57:-1]
    part = part.astype(float)
    median = np.nan_to_num(np.round(np.nanmedian(part)))
    if row['trend'] > 0:
        return 1.05*median
    elif row['trend'] < 0:
        return 0.95*median
    else:
        return median

def submitWithMedianAndPercentagePureTrend5pc():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane et une variation de 5% suivant la tenda
nce
    """
    dfTest12 = test.copy()
    dfTrain12 = train.copy()
    dfTest12['Page'] = dfTest12.Page.apply(lambda a: a[:-11])
    dfTrain12['trend'] = dfTrain12.apply(getPureTrendCoeff, axis = 1)
    dfTrain12['Visits'] = dfTrain12.apply(applyCoeff, axis = 1)
    dfTest12 = dfTest12.merge(dfTrain12[['Page','Visits']], how='left')
    dfTest12['Visits'] = dfTest12.Visits.apply(lambda a: (math.floor(a * 1000))/1000)
    dfTest12.loc[dfTest12.Visits.isnull(), 'Visits'] = 0
    dfTest12.drop('Page', axis=1)
    dfTest12[['Id','Visits']].to_csv('median_trend_5pc_submit.csv', index=False)

def getDataForProphet(_data):
    """ Méthode qui transforme les données pour Prophet
    Arguments:
    _data -- les données
    Retour:
     --
    """
    data=_data.iloc[:].to_frame()
    data.columns = ['visits']
    data['median'] = data.visits.median()
    std_mult = 1.0
    data.ix[np.abs(data.visits-data.visits.median())>=(std_mult*data.visits.std()),'visits'] = data.ix[np.abs(data.visits-dat
a.visits.median())>=(std_mult*data.visits.std()),'median']
    data.index = pd.to_datetime(data.index)
    X = pd.DataFrame(index=range(0,len(data)))
    X['ds'] = data.index
    X['y'] = data['visits'].values
    return X

def runBlending3Models(_id, _frontier, _data):
    """ Méthode qui détermine le meilleur modèle suivant son SMAPE
    Arguments:
    _id -- l'index de la TS
    _frontier -- frontière entre les données d'entrafnement et de test pour Prophet (souvent 60)
    _data -- les données
    Retour:
     --
    """
    _data=_data.iloc[_id,:]
    # séparation train / test
    data_train = _data.iloc[0:-_frontier]
    data_test = _data.iloc[-_frontier:]
    test_median = data_test.median()
    test_mean = data_test.mean()
    test_cleaned = data_test.T.fillna(test_median).T
    y_truth = test_cleaned

    # cela améliore légèrement le smape de la médiane
    data_test.ix[np.abs(data_test-data_test.median())>=(1.5*data_test.std())] = test_median
    test_median = data_test.median()

    # calcul moyenne
    forecast_mean = test_mean*np.ones(_frontier)
    smape_fast_mean = smape_fast(y_truth, forecast_mean)

    # calcul médiane
    forecast_median =  test_median*np.ones(_frontier)
    smape_fast_median = smape_fast(y_truth, forecast_median)

    if _data.count() > _frontier:

        train_median = data_train.iloc[1:].median()
        train_cleaned = data_train.T.iloc[1:].fillna(train_median).T

        # 1er prophet
        X = getDataForProphet(train_cleaned)
        m = Prophet(weekly_seasonality=True, yearly_seasonality=True)
        m.fit(X)
        future = m.make_future_dataframe(periods=_frontier)
        forecast = m.predict(future)
        y_forecasted = forecast.iloc[-_frontier:,2].values
        smape_fast_prophet = smape_fast(y_truth, y_forecasted)
```

```
        max_sf = min(smape_fast_mean, smape_fast_median, smape_fast_prophet)
        if max_sf == smape_fast_mean:
            return "mean", smape_fast_mean#, forecast_mean
        if max_sf == smape_fast_median:
            return "median", smape_fast_median#, forecast_median
        if max_sf == smape_fast_prophet:
            data_cleaned = _data.T.iloc[1:].fillna(test_median).T
            X = getDataForProphet(data_cleaned)
            m = Prophet(weekly_seasonality=True, yearly_seasonality=True)
            m.fit(X)
            future = m.make_future_dataframe(periods=60)
            forecast = m.predict(future)
            y_forecasted = forecast.iloc[-60:,2].values
            return "proph", smape_fast_prophet#, y_forecasted
    else:
        if _data.count() > 1:
            if smape_fast_mean > smape_fast_median:
                return "mean", smape_fast_mean#, forecast_mean
            else:
                return "median", smape_fast_median#, forecast_median

        else: # que des NaN sauf le nom de la page
            forecast_zero = 0*np.ones(_frontier)
            return "zero", 0, forecast_zero

def submitWithMedianWeJo():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane et la gestion du WE et des jours ouvrés
    """
    dfTest5 = test.copy()
    dfTrain5 = train.copy()
    dfTest5['date'] = dfTest5.Page.apply(lambda a: a[-10:])
    dfTest5['Page'] = dfTest5.Page.apply(lambda a: a[:-11])
    dfTest5['date'] = dfTest5['date'].astype('datetime64[ns]')
    dfTest5['weekend'] = ((dfTest5.date.dt.dayofweek) // 5 == 1).astype(float)
    train_flattened = pd.melt(dfTrain5[list(dfTrain5.columns[1:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    train_flattened['weekend'] = ((train_flattened.date.dt.dayofweek) // 5 == 1).astype(float)
    train_flattened['Visits'].fillna(0, inplace=True)
    train_page_per_dow = train_flattened.groupby(['Page','weekend']).median().reset_index()
    dfTest5 = dfTest5.merge(train_page_per_dow, how='left')
    dfTest5[['Id','Visits']].to_csv('medianWEJO_submit.csv', index=False)

def submitWithMedianPerDay():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane par jour de la semaine
    """
    dfTest6 = test.copy()
    dfTrain6 = train.copy()
    dfTest6['date'] = dfTest6.Page.apply(lambda a: a[-10:])
    dfTest6['Page'] = dfTest6.Page.apply(lambda a: a[:-11])
    dfTest6['date'] = dfTest6['date'].astype('datetime64[ns]')
    dfTest6['dayofweek'] = dfTest6.date.dt.dayofweek
    train_flattened = pd.melt(dfTrain6[list(dfTrain6.columns[1:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    train_flattened['dayofweek'] = train_flattened.date.dt.dayofweek
    train_flattened['Visits'].fillna(0, inplace=True)
    train_page_per_dow = train_flattened.groupby(['Page','dayofweek']).median().reset_index()
    dfTest6 = dfTest6.merge(train_page_per_dow, how='left')
    dfTest6[['Id','Visits']].to_csv('medianPerDay_submit.csv', index=False)

def submitWithMedianWeJo56():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la dictinction de la médiane WE / JO pour les 56 derniers jours
    """
    dfTest7 = test.copy()
    dfTrain7 = train.copy()
    dfTest7['date'] = dfTest7.Page.apply(lambda a: a[-10:])
    dfTest7['Page'] = dfTest7.Page.apply(lambda a: a[:-11])
    dfTest7['date'] = dfTest7['date'].astype('datetime64[ns]')
    dfTest7['weekend'] = ((dfTest7.date.dt.dayofweek) // 5 == 1).astype(float)
    train_flattened = pd.melt(dfTrain7[list(dfTrain7.columns[-56:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    train_flattened['weekend'] = ((train_flattened.date.dt.dayofweek) // 5 == 1).astype(float)
    train_flattened['Visits'].fillna(0, inplace=True)
    train_page_per_dow = train_flattened.groupby(['Page','weekend']).median().reset_index()
    dfTest7 = dfTest7.merge(train_page_per_dow, how='left')
    dfTest7[['Id','Visits']].to_csv('medianWEJO56_submit.csv', index=False)
def submitWithMedianPerDay56():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane par jour de la semaine pour les 56 derniers jours
    """
    dfTest8 = test.copy()
    dfTrain8 = train.copy()
    dfTest8['date'] = dfTest8.Page.apply(lambda a: a[-10:])
    dfTest8['Page'] = dfTest8.Page.apply(lambda a: a[:-11])
    dfTest8['date'] = dfTest8['date'].astype('datetime64[ns]')
    dfTest8['dayofweek'] = dfTest8.date.dt.dayofweek
    train_flattened = pd.melt(dfTrain8[list(dfTrain8.columns[-56:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    train_flattened['dayofweek'] = train_flattened.date.dt.dayofweek
    train_flattened['Visits'].fillna(0, inplace=True)
    train_page_per_dow = train_flattened.groupby(['Page','dayofweek']).median().reset_index()
    dfTest8 = dfTest8.merge(train_page_per_dow, how='left')
    dfTest8[['Id','Visits']].to_csv('medianPerDay56_submit.csv', index=False)
```

```
def submitWithMedianJoursFeries():
    """ Méthode qui crée le fichier à soumettre à Kaggle avec le modèle de la médiane et la gestion des jours fériés
    """
    test = pd.read_csv("key_1.csv")
    train = pd.read_csv("train_1.csv")
    train = pd.melt(train[list(train.columns[-49:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    train['date'] = train['date'].astype('datetime64[ns]')
    train['ferie'] = ((train.date.dt.dayofweek) >=5).astype(float)
    train['origine']=train['Page'].apply(lambda x:re.split(".wikipedia.org", x)[0][-2:])

    train_us=['2015-07-04','2015-11-26','2015-12-25']+\
    ['2016-07-04','2016-11-24','2016-12-26']
    test_us=[]
    train_uk=['2015-12-25','2015-12-28'] +\
    ['2016-01-01','2016-03-28','2016-05-02','2016-05-30','2016-12-26','2016-12-27']
    test_uk=['2017-01-01']
    train_de=['2015-10-03', '2015-12-25', '2015-12-26']+\
    ['2016-01-01', '2016-03-25', '2016-03-26', '2016-03-27', '2016-01-01', '2016-05-05', '2016-05-15', '2016-05-16', '2016-10
-03', '2016-12-25', '2016-12-26']
    test_de=['2017-01-01']
    train_fr=['2015-07-14', '2015-08-15', '2015-11-01', '2015-11-11', '2015-12-25']+\
    ['2016-01-01','2016-03-28', '2016-05-01', '2016-05-05', '2016-05-08', '2016-05-16', '2016-07-14', '2016-08-15', '2016-11-
01','2016-11-11', '2016-12-25']
    test_fr=['2017-01-01']
    train_ru=['2015-11-04']+\
    ['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07', '2016-02-23', '2016-03
-08', '2016-05-01', '2016-05-09', '2016-06-12', '2016-11-04']
    test_ru=['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04', '2017-01-05', '2017-01-06', '2017-01-07', '2017-02-23']
    train_es=['2015-08-15', '2015-10-12', '2015-11-01', '2015-12-06', '2015-12-08', '2015-12-25']+\
    ['2016-01-01', '2016-01-06', '2016-03-25', '2016-05-01', '2016-08-15', '2016-10-12', '2016-11-01', '2016-12-06', '2016-12
-08', '2016-12-25']
    test_es=['2017-01-01', '2017-01-06']
    train_ja=['2015-07-20','2015-09-21', '2015-10-12', '2015-11-03', '2015-11-23', '2015-12-23']+\
    ['2016-01-01', '2016-01-11', '2016-02-11', '2016-03-20', '2016-04-29', '2016-05-03', '2016-05-04', '2016-05-05', '2016-07
-18', '2016-08-11', '2016-09-22', '2016-10-10', '2016-11-03', '2016-11-23', '2016-12-23']
    test_ja=['2017-01-01', '2017-01-09', '2017-02-11']
    train_zh=['2015-09-27', '2015-10-01', '2015-10-02','2015-10-03','2015-10-04','2015-10-05','2015-10-06','2015-10-07']+\
    ['2016-01-01', '2016-01-02', '2016-01-03', '2016-02-08', '2016-02-09', '2016-02-10', '2016-02-11', '2016-02-12', '2016-04
-04', '2016-05-01', '2016-05-02', '2016-06-09', '2016-06-10', '2016-09-15', '2016-09-16', '2016-10-03', '2016-10-04','2016-10
-05','2016-10-06','2016-10-07']
    test_zh=['2017-01-02', '2017-02-27', '2017-02-28', '2017-03-01']
    #in China some saturday and sundays are worked
    train_o_zh=['2015-10-10','2016-02-06', '2016-02-14', '2016-06-12', '2016-09-18', '2016-10-08', '2016-10-09']
    test_o_zh=['2017-01-22', '2017-02-04']

    #let's replace values in 'ferie' columns
    train.loc[(train.origine=='ts')&(train.date.isin(train_us+train_uk)), 'ferie']=1
    train.loc[(train.origine=='er')&(train.date.isin(train_us+train_uk)), 'ferie']=1
    train.loc[(train.origine=='en')&(train.date.isin(train_us+train_uk)), 'ferie']=1
    train.loc[(train.origine=='de')&(train.date.isin(train_de)), 'ferie']=1
    train.loc[(train.origine=='fr')&(train.date.isin(train_fr)), 'ferie']=1
    train.loc[(train.origine=='ru')&(train.date.isin(train_ru)), 'ferie']=1
    train.loc[(train.origine=='es')&(train.date.isin(train_es)), 'ferie']=1
    train.loc[(train.origine=='ja')&(train.date.isin(train_ja)), 'ferie']=1
    train.loc[(train.origine=='zh')&(train.date.isin(train_zh)), 'ferie']=1
    train.loc[(train.origine=='zh')&(train.date.isin(train_o_zh)), 'ferie']=0

    #same with test
    test['date'] = test.Page.apply(lambda a: a[-10:])
    test['Page'] = test.Page.apply(lambda a: a[:-11])
    test['date'] = test['date'].astype('datetime64[ns]')
    test['ferie'] = ((test.date.dt.dayofweek) >=5).astype(float)
    test['origine']=test['Page'].apply(lambda x:re.split(".wikipedia.org", x)[0][-2:])

    test.loc[(test.origine=='ts')&(test.date.isin(test_us+test_uk)), 'ferie']=1
    test.loc[(test.origine=='er')&(test.date.isin(test_us+test_uk)), 'ferie']=1
    test.loc[(test.origine=='en')&(test.date.isin(test_us+test_uk)), 'ferie']=1
    test.loc[(test.origine=='de')&(test.date.isin(test_de)), 'ferie']=1
    test.loc[(test.origine=='fr')&(test.date.isin(test_fr)), 'ferie']=1
    test.loc[(test.origine=='ru')&(test.date.isin(test_ru)), 'ferie']=1
    test.loc[(test.origine=='es')&(test.date.isin(test_es)), 'ferie']=1
    test.loc[(test.origine=='ja')&(test.date.isin(test_ja)), 'ferie']=1
    test.loc[(test.origine=='zh')&(test.date.isin(test_zh)), 'ferie']=1
    test.loc[(test.origine=='zh')&(test.date.isin(test_o_zh)), 'ferie']=0

    train_page_per_dow = train.groupby(['Page','ferie']).median().reset_index()
    test = test.merge(train_page_per_dow, how='left')

    test.loc[test.Visits.isnull(), 'Visits'] = 0
    test['Visits']=(test['Visits']*10).astype('int')/10
    test[['Id','Visits']].to_csv('sumitJFen.csv', index=False)
```

```
def setDatePredictionColumns(_df):
    """ Méthode qui ajoute les 60 colonnes pour la prévision de janvier/février
    Arguments:
    _df -- le dataframe auquel ajouter les 60 colonnes
    Retour:
    _df -- le dataframe avec les 60 colonnes de la prévision
    """
    _df['2017-01-01'] = _df['2017-01-02'] = _df['2017-01-03'] = _df['2017-01-04'] = _df['2017-01-05'] = 0
    _df['2017-01-06'] = _df['2017-01-07'] = _df['2017-01-08'] = _df['2017-01-09'] = _df['2017-01-10'] = 0
    _df['2017-01-11'] = _df['2017-01-12'] = _df['2017-01-13'] = _df['2017-01-14'] = _df['2017-01-15'] = 0
    _df['2017-01-16'] = _df['2017-01-17'] = _df['2017-01-18'] = _df['2017-01-19'] = _df['2017-01-20'] = 0
    _df['2017-01-21'] = _df['2017-01-22'] = _df['2017-01-23'] = _df['2017-01-24'] = _df['2017-01-25'] = 0
    _df['2017-01-26'] = _df['2017-01-27'] = _df['2017-01-28'] = _df['2017-01-29'] = _df['2017-01-30'] = 0
    _df['2017-01-31'] = _df['2017-02-01'] = _df['2017-02-02'] = _df['2017-02-03'] = _df['2017-02-04'] = 0
    _df['2017-02-05'] = _df['2017-02-06'] = _df['2017-02-07'] = _df['2017-02-08'] = _df['2017-02-09'] = 0
    _df['2017-02-10'] = _df['2017-02-11'] = _df['2017-02-12'] = _df['2017-02-13'] = _df['2017-02-14'] = 0
    _df['2017-02-15'] = _df['2017-02-16'] = _df['2017-02-17'] = _df['2017-02-18'] = _df['2017-02-19'] = 0
    _df['2017-02-20'] = _df['2017-02-21'] = _df['2017-02-22'] = _df['2017-02-23'] = _df['2017-02-24'] = 0
    _df['2017-02-25'] = _df['2017-02-26'] = _df['2017-02-27'] = _df['2017-02-28'] = _df['2017-03-01'] = 0
    return _df
```

```
def setFerieTrain(_train, _test):
    """ Méthode qui gère les jours fériés en ajoutant la colonne 'ferie'
    Arguments:
    _train -- les données d'entraînement
    _test -- les données de test
    Retour:
    _train -- les données d'entraînement où on a ajouté la gestion des jours fériés
    _test -- les données de test où on a ajouté la gestion des jours fériés
    """
    train_us=['2015-07-04','2015-11-26','2015-12-25']+\
    ['2016-07-04','2016-11-24','2016-12-26']
    train_uk=['2015-12-25','2015-12-28'] +\
    ['2016-01-01','2016-03-28','2016-05-02','2016-05-30','2016-12-26','2016-12-27']
    train_de=['2015-10-03', '2015-12-25', '2015-12-26']+\
    ['2016-01-01', '2016-03-25', '2016-03-26', '2016-03-27', '2016-01-01', '2016-05-05', '2016-05-15', '2016-05-16', '2016-10
-03', '2016-12-25', '2016-12-26']
    train_fr=['2015-07-14', '2015-08-15', '2015-11-01', '2015-11-11', '2015-12-25']+\
    ['2016-01-01','2016-03-28', '2016-05-01', '2016-05-05', '2016-05-08', '2016-05-16', '2016-07-14', '2016-08-15', '2016-11-
01','2016-11-11', '2016-12-25']
    train_ru=['2015-11-04']+\
    ['2016-01-01', '2016-01-02', '2016-01-03', '2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07', '2016-02-23', '2016-03
-08', '2016-05-01', '2016-05-09', '2016-06-12', '2016-11-04']
    train_es=['2015-08-15', '2015-10-12', '2015-11-01', '2015-12-06', '2015-12-08', '2015-12-25']+\
    ['2016-01-01', '2016-01-06', '2016-03-25', '2016-05-01', '2016-08-15', '2016-10-12', '2016-11-01', '2016-12-06', '2016-12
-08', '2016-12-25']
    train_ja=['2015-07-20','2015-09-21', '2015-10-12', '2015-11-03', '2015-11-23', '2015-12-23']+\
    ['2016-01-01', '2016-01-11', '2016-02-11', '2016-03-20', '2016-04-29', '2016-05-03', '2016-05-04', '2016-05-05', '2016-07
-18', '2016-08-11', '2016-09-22', '2016-10-10', '2016-11-03', '2016-11-23', '2016-12-23']
    train_zh=['2015-09-27', '2015-10-01', '2015-10-02','2015-10-03','2015-10-04','2015-10-05','2015-10-06','2015-10-07']+\
    ['2016-01-01', '2016-01-02', '2016-01-03', '2016-02-08', '2016-02-09', '2016-02-10', '2016-02-11', '2016-02-12', '2016-04
-04', '2016-05-01', '2016-05-02', '2016-06-09', '2016-06-10', '2016-09-15', '2016-09-16', '2016-10-03', '2016-10-04','2016-10
-05','2016-10-06','2016-10-07']
    #in China some saturday and sundays are worked
    train_o_zh=['2015-10-10','2016-02-06', '2016-02-14', '2016-06-12', '2016-09-18', '2016-10-08', '2016-10-09']

    test_us=[]
    test_uk=['2017-01-01']
    test_de=['2017-01-01']
    test_fr=['2017-01-01']
    test_ru=['2017-01-01', '2017-01-02', '2017-01-03', '2017-01-04', '2017-01-05', '2017-01-06', '2017-01-07', '2017-02-23']
    test_es=['2017-01-01', '2017-01-06']
    test_ja=['2017-01-01', '2017-01-09', '2017-02-11']
    test_zh=['2017-01-02', '2017-02-27', '2017-02-28', '2017-03-01']
    test_o_zh=['2017-01-22', '2017-02-04']

    _train = pd.melt(_train[list(_train.columns[-55:])+['Page']], id_vars='Page', var_name='date', value_name='Visits')
    _train['date'] = _train['date'].astype('datetime64[ns]')
    _train['ferie'] = ((_train.date.dt.dayofweek) >=5).astype(float)
    _train['origine']=_train['Page'].apply(lambda x:re.split(".wikipedia.org", x)[0][-2:])
    _test['date'] = _test['date'].astype('datetime64[ns]')
    _test['ferie'] = ((_test.date.dt.dayofweek) >=5).astype(float)
    _test['origine']=_test['Page'].apply(lambda x:re.split(".wikipedia.org", x)[0][-2:])

    #let's replace values in 'ferie' columns
    _train.loc[(_train.origine=='en')&(_train.date.isin(train_us+train_uk)), 'ferie']=1
    _train.loc[(_train.origine=='de')&(_train.date.isin(train_de)), 'ferie']=1
    _train.loc[(_train.origine=='fr')&(_train.date.isin(train_fr)), 'ferie']=1
    _train.loc[(_train.origine=='ru')&(_train.date.isin(train_ru)), 'ferie']=1
    _train.loc[(_train.origine=='es')&(_train.date.isin(train_es)), 'ferie']=1
    _train.loc[(_train.origine=='ja')&(_train.date.isin(train_ja)), 'ferie']=1
    _train.loc[(_train.origine=='zh')&(_train.date.isin(train_zh)), 'ferie']=1
    _train.loc[(_train.origine=='zh')&(_train.date.isin(train_o_zh)), 'ferie']=0

    _test.loc[(_test.origine=='en')&(_test.date.isin(test_us+test_uk)), 'ferie']=1
    _test.loc[(_test.origine=='de')&(_test.date.isin(test_de)), 'ferie']=1
    _test.loc[(_test.origine=='fr')&(_test.date.isin(test_fr)), 'ferie']=1
    _test.loc[(_test.origine=='ru')&(_test.date.isin(test_ru)), 'ferie']=1
    _test.loc[(_test.origine=='es')&(_test.date.isin(test_es)), 'ferie']=1
    _test.loc[(_test.origine=='ja')&(_test.date.isin(test_ja)), 'ferie']=1
    _test.loc[(_test.origine=='zh')&(_test.date.isin(test_zh)), 'ferie']=1
    _test.loc[(_test.origine=='zh')&(_test.date.isin(test_o_zh)), 'ferie']=0

    return _train, _test

def run2Models(row):
    """ Méthode qui détermine le meilleur modèle entre la médiane et la moyenne
    Arguments:
    row -- les données de la TS
    Retour:
    row -- les données initiales auxquelles on a ajouté la meilleure prévision
    """
    # séparation train / test
    data_train = row.iloc[0:-115] # avant les 56 de novembre/décembre (les 60 de la prédiction en plus)
    data_test = row.iloc[-115:-60] # les 56 de novembre/décembre avant les 60 de la prédiction

    row_median = train14_page_ferie_median[train14_page_ferie_median.Page == row['Page']]
    row_median = row_median.reset_index(drop=True)

    med_ouv = row_median[row_median.ferie == 0.0].Visits.values[0]
    med_fer = row_median[row_median.ferie == 1.0].Visits.values[0]
    y_truth = data_test.T.fillna(med_ouv).T

    JF = train14[train14.Page == row['Page']].ferie # [1.0 0.0 0.0 0.0 1.0 1.0 0.0 ...]
    JJ_med = JF*med_fer + (1-JF)*med_ouv
```

```
    # calcul médiane
    y_truth = y_truth.reset_index(drop=True) # pour pouvoir incrémenter sur les index dans le smape
    JJ_med = JJ_med.reset_index(drop=True)

    smape_fast_median = smape_fast(y_truth, JJ_med)

    row_mean = train14_page_ferie_mean[train14_page_ferie_mean.Page == row['Page']]
    row_mean = row_mean.reset_index(drop=True)
    mea_ouv = row_mean[row_mean.ferie == 0.0].Visits.values[0]
    mea_fer = row_mean[row_mean.ferie == 1.0].Visits.values[0]

    y_truth = data_test.T.fillna(mea_ouv).T

    JJ_mea = JF*mea_fer + (1-JF)*mea_ouv

    # calcul moyenne
    y_truth = y_truth.reset_index(drop=True) # pour pouvoir incrémenter sur les index dans le smape
    JJ_mea = JJ_mea.reset_index(drop=True)

    testFerie = test14[test14.Page == row['Page']]

    smape_fast_mean = smape_fast(y_truth, JJ_mea)

    if smape_fast_median >= smape_fast_mean:
        JJ = testFerie.ferie * med_fer +(1-testFerie.ferie) * med_ouv
    else:
        JJ = testFerie.ferie * mea_fer +(1-testFerie.ferie) * mea_ouv

    row_transformed = np.append(row[:-60].values, JJ)

    return row_transformed
def submit2Models():
    """ Méthode pour créer le fichier à soumettre à Kaggle avec la compétition des deux modèles moyenne et médiane
    """
    dfTrain11 = train.copy()
    dfTest11 = test.copy()
    dfTest11['date'] = dfTest11.Page.apply(lambda a: a[-10:])
    dfTest11['Page'] = dfTest11.Page.apply(lambda a: a[:-11])
    train14, test14 = setFerieTrain(dfTrain11, dfTest11)
    dfTrain11 = setDatePredictionColumns(dfTrain11)
    train14_page_ferie_median = train14.groupby(['Page','ferie']).median().reset_index()
    train14_page_ferie_mean = train14.groupby(['Page','ferie']).mean().reset_index()

    #dfTrain111 = dfTrain11.iloc[16000:17000,:]
    #dfTest111 = dfTest11[dfTest11.Page.isin(dfTrain111.Page.values)]
    dfTrain111 = dfTrain11.copy()
    dfTest111 = dfTest11.copy()
    dfTrain111 = dfTrain111.apply(run2Models, axis=1)
    train_flattened = pd.melt(dfTrain111[list(dfTrain111.columns[-60:])+['Page']], id_vars='Page', var_name='date', value_nam
e='Visits')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    dfTest1111 = dfTest111[['Page', 'Id', 'date']].merge(train_flattened[['Page', 'date', 'Visits']], how='left')
    dfTest1111.loc[dfTest1111.Visits.isnull(), 'Visits'] = 0
    dfTest1111[['Id','Visits']].to_csv('2models_submit.csv', index=False)

def make_autopct(values):
    def my_autopct(pct):
        """Customisation de l'affichage des valeurs {p:.2f}% ({v:d})
        Arguments:
        values -- nombres de variables pour chaque section
        pct -- pourcentages de variables pour chaque section
        """
        total = sum(values)
        val = int(round(pct*total/100.0))
        if val == 0:
            return ''
        else:
            return '{p:.2f}% ({v:d})'.format(p=pct, v=val)
    return my_autopct

def graphPourcentageNan():
    """ Graphe des pourcentages de valeurs manquantes
    """
    trainCount = train.iloc[:,1:].count().values/(train.shape[0]/10)
    for i in range(len(trainCount)):
        trainCount[i] = int(trainCount[i])
    trainCount = trainCount.astype(int)
    classement=[]
    for i in range(10):
        classement.append(0)
    for i in range (train.count().shape[0]-1):
        classement[trainCount[i]] += 1

    explode = (0, 0, 0, 0, 0, 0, 0, 0, 0, 0)

    the_grid = GridSpec(2,2)
    plt.clf()
    plt.figure(figsize=(17,10))
    plt.subplot(the_grid[:,0], aspect=1)
    _, _, autotexts = plt.pie(classement, explode=explode, autopct=make_autopct(classement), startangle=90, shadow=True, pctd
istance=0.5) #'%1.1f%%'
    plt.title(u"% de journées ayant\n entre . et . % de données renseignées (couleurs)", bbox={'facecolor':'0.8', 'pad':5})
    autotexts[0].set_color('white')
    autotexts[0].set_fontsize(16)
    autotexts[3].set_color('black')
```

```
    autotexts[3].set_fontsize(16)
    autotexts[3].set_position((0.60, 0.10))
    autotexts[4].set_color('black')
    autotexts[4].set_fontsize(16)
    autotexts[4].set_position((0.70, 0.35))
    autotexts[5].set_color('black')
    autotexts[5].set_fontsize(16)
    autotexts[5].set_position((0.30, 0.90))
    plt.axis('equal')
    labels = ['0-10%', '10-20%', '20-30%', '30-40%', '40-50%', '50-60%', '60-70%', '70-80%', '80-90%', '90-100%']
    plt.legend(labels, loc='lower right')
    plt.show()

def getLanguage(page):
    """ Méthode qui renvoie la langue d'une page
    Arguments:
    page -- la page
    Retour:
    la langue de la page
    """
    res = re.search('[a-z][a-z].wikipedia.org',page)
    if res:
        return res[0][0:2]
    return 'na'

def influenceLangage():
    """
    Arguments:
     --
    Retour:
     --
    """
    dfTrainLang = train.copy()
    dfTrainLang['lang'] = dfTrainLang.Page.map(getLanguage)

    lang_sets = {}
    lang_sets['en'] = dfTrainLang[dfTrainLang.lang=='en'].iloc[:,0:-1]
    lang_sets['ja'] = dfTrainLang[dfTrainLang.lang=='ja'].iloc[:,0:-1]
    lang_sets['de'] = dfTrainLang[dfTrainLang.lang=='de'].iloc[:,0:-1]
    lang_sets['na'] = dfTrainLang[dfTrainLang.lang=='na'].iloc[:,0:-1]
    lang_sets['fr'] = dfTrainLang[dfTrainLang.lang=='fr'].iloc[:,0:-1]
    lang_sets['zh'] = dfTrainLang[dfTrainLang.lang=='zh'].iloc[:,0:-1]
    lang_sets['ru'] = dfTrainLang[dfTrainLang.lang=='ru'].iloc[:,0:-1]
    lang_sets['es'] = dfTrainLang[dfTrainLang.lang=='es'].iloc[:,0:-1]

    sums = {}
    for key in lang_sets:
        sums[key] = lang_sets[key].iloc[:,1:].sum(axis=0) / lang_sets[key].shape[0]

    days = [r for r in range(sums['en'].shape[0])]

    fig = plt.figure(1,figsize=[10,10])
    plt.ylabel('Visites')
    plt.xlabel('Jour')
    plt.title('Nombre moyen de visites par page pour les différents langages')
    labels={'en':'English','ja':'Japanese','de':'German', 'na':'Media','fr':'French','zh':'Chinese', 'ru':'Russian','es':'Spa
nish'}

    for key in sums:
        plt.plot(days,sums[key],label = labels[key] )

    plt.legend()
    plt.show()

def crossWeekdaysMonth():
    """ Méthode qui croise les données pour mettre en exergue les plus ou moins importantes moyennes de trafic en fonction de
s mois et des jours de la semaine
    """
    train_flattened = pd.melt(train[list(train.columns[-184:])+['Page']], id_vars='Page', var_name='date', value_name='Visits
')
    train_flattened['date'] = train_flattened['date'].astype('datetime64[ns]')
    train_flattened['weekend'] = ((train_flattened.date.dt.dayofweek) // 5 == 1).astype(float)
    df_median = pd.DataFrame(train_flattened.groupby(['Page'])['Visits'].median())
    df_median.columns = ['median']
    df_mean = pd.DataFrame(train_flattened.groupby(['Page'])['Visits'].mean())
    df_mean.columns = ['mean']
    train_flattened = train_flattened.set_index('Page').join(df_mean).join(df_median)
    train_flattened.reset_index(drop=False,inplace=True)
    train_flattened.loc[(train_flattened['Visits']>1.5*train_flattened['median']), 'Visits'] = np.nan
    train_flattened['Visits'] = train_flattened['Visits'].interpolate()
    train_flattened['weekday'] = train_flattened['date'].apply(lambda x: x.weekday())
    train_flattened['Date_str'] = train_flattened['date'].apply(lambda x: str(x))
    train_flattened['year'] = train_flattened['Date_str'].str[0:4]
    train_flattened['month'] = train_flattened['Date_str'].str[5:7]
    train_flattened['day'] = train_flattened['Date_str'].str[8:10]
    train_flattened.drop('Date_str',axis = 1, inplace =True)
    # For the next graphics
    train_flattened['month_num'] = train_flattened['month']
    train_flattened['month'].replace('07','07 - July',inplace=True)
    train_flattened['month'].replace('08','08 - August',inplace=True)
    train_flattened['month'].replace('09','09 - September',inplace=True)
    train_flattened['month'].replace('10','10 - Oktober',inplace=True)
    train_flattened['month'].replace('11','11 - November',inplace=True)
    train_flattened['month'].replace('12','12 - December',inplace=True)
```

```
    train_flattened['weekday_num'] = train_flattened['weekday']
    train_flattened['weekday'].replace(0,'01 - Monday',inplace=True)
    train_flattened['weekday'].replace(1,'02 - Tuesday',inplace=True)
    train_flattened['weekday'].replace(2,'03 - Wednesday',inplace=True)
    train_flattened['weekday'].replace(3,'04 - Thursday',inplace=True)
    train_flattened['weekday'].replace(4,'05 - Friday',inplace=True)
    train_flattened['weekday'].replace(5,'06 - Saturday',inplace=True)
    train_flattened['weekday'].replace(6,'07 - Sunday',inplace=True)
    train_group = train_flattened.groupby(["month", "weekday"])['Visits'].mean().reset_index()
    train_group = train_group.pivot('weekday','month','Visits')
    train_group.sort_index(inplace=True)
    sns.set(font_scale=3.5)

    # Draw a heatmap with the numeric values in each cell
    f, ax = plt.subplots(figsize=(50, 30))
    sns.heatmap(train_group, annot=False, ax=ax, fmt="d", linewidths=2)
    plt.title('Web Traffic Months cross Weekdays')
    plt.show()

def headTSTrain(_id):
    """ Méthode qui réoriente les données afin d'être utilisée par le test de Dickey-Fuller
    Arguments:
    _id -- l'index de la TS
    Retour:
    data_train -- les données d'entraînement de la TS
    """
    data=train.iloc[_id,:]
    data_train, data_test = data.iloc[:-60], data.iloc[-60:]
    test_median = data_test.median()
    test_cleaned = data_test.T.fillna(test_median).T
    train_median = data_train.iloc[1:].median()
    train_cleaned = data_train.T.iloc[1:].fillna(train_median).T
    data_train=train_cleaned.iloc[:].to_frame()
    data_train.columns = ['Visits']
    data_train['date'] = data_train.index
    data_train.reset_index(drop=True, inplace=True)
    return data_train

def dickeyFullerAugmentedTest(_data):
    """ Méthode qui donne les résultats du test
    Arguments:
    _data -- les données de la TS sur lequel porte le test
    Retour:
    print -- impression des résultats du test
    """
    dftest = sm.tsa.adfuller(_data, autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)

def arimarima(_all_data, _id):
    """ Méthode qui lance ARIMA
    Arguments:
    _all_data -- les données
    _id -- l'index de la TS sur laquelle porte ARIMA
    Retour:
    print -- imprime le SMAPE
    """
    n_cols = len(_all_data.columns) - 1
    n_cols_train = round(n_cols / 10*8)
    cols_train = _all_data.columns[1:n_cols_train]
    cols_predict = _all_data.columns[n_cols_train:-1]
    data = np.array(_all_data.loc[_id, _all_data.columns[1:-1]],'f')
    data_train = np.array(_all_data.loc[_id, cols_train],'f')
    data_predict = np.array(_all_data.loc[_id, cols_predict],'f')
    arima = ARIMA(data_train, [2,1,2])
    result = arima.fit(disp=False)
    pred = result.forecast(steps=data_predict.shape[0])[0]
    x = [i for i in range(len(data))]
    x_train = [i for i in range(n_cols_train-1)]
    x_pred = [i for i in range(n_cols_train, n_cols_train+data_predict.shape[0])]
    print("smape = ", smape_fast(data_predict, pred))
    fig = plt.figure(1,figsize=(10,5))
    plt.plot(x, data, label='Data')
    plt.plot(x_train, data_train, label='Train Data')
    plt.plot(x_pred, pred,label='ARIMA Model')
    plt.xlabel('Days')
    plt.ylabel('Views')
    plt.legend()
    plt.show()
```

```python
def getDfTS(_id):
    """ MEt correctement en forme les données de la TS
    Arguments:
    _id -- l'index de la TS
    Retour:
    trainId -- les données d'entraînement de la TS
    testId -- les données de test de la TS
    """
    serie = train.iloc[_id, 1:]
    dfTS = pd.DataFrame(serie)
    dfTS['date'] = dfTS.index
    dfTS.reset_index(drop=True, inplace=True)
    dfTS.columns=['Visits', 'date']
    trainId, testId = dfTS.iloc[:-60,:], dfTS.iloc[-60:,:]
    return trainId, testId
def create_dataset(dataset, look_back):
    """
    Arguments:
    dataset -- les données de la TS
    loop_back -- nombre de jours de rétrovision
    Retour:
     --
    """
    dataX = []
    dataY = []
    for i in range(len(dataset)-look_back-1):
        a = dataset.iloc[i:(i+look_back), 0].values[0]
        b = dataset.iloc[i+look_back, 0]
        dataX.append(a)
        dataY.append(b)
    return np.array(dataX), np.array(dataY)

def runKeras(_id, _look_back):
    """ Méthode qui lance Keras
    Arguments:
    _id -- index de la TS
    _look_back -- nombre de jours de rétrovision
    Retour:
    trainPredict -- prévision sur l'entraînement
    testPredict -- prévision sur le test
    testId -- les données réelles du test
    """
    trainId, testId = getDfTS(_id)
    trainX, trainY = create_dataset(trainId, _look_back)
    testX, testY = create_dataset(testId, _look_back)

    model = Sequential()
    model.add(Dense(8, input_dim=_look_back, activation='relu'))
    model.add(Dense(1))
    model.compile(loss='mean_absolute_error', optimizer='adam')
    model.fit(trainX, trainY, epochs=150, batch_size=2, verbose=0)

    trainPredict = model.predict(trainX)
    testPredict = model.predict(testX)

    return trainPredict, testPredict, testId
```