Christophe Coudé, 2017

Parcours Data Scientist, projet n°4, Notebook d'exploration

Objectifs :
Afin d'optimiser la logistique et d'anticiper des retards de vol, on cherche à utiliser, pour une compagnie d'aviation, les données du site Bureau of Transportation Statistics : https://www.transtats.bts.gov/OT_Delay/OT_DelayCause1.asp

1- Choix des variables, recherche de nouvelles données

Nous avons vu que l'utilisation des variables de ce dataset posait problème. Le data leakage.

Si on étudie un peu plus précisément les variables qui sont proposées dans ce jeu de données, on s'aperçoit que les variables explicatives les plus importantes qui ressortent appartiennent à deux séries de données.

- les variables *_delay qui donnent pour chaque cause de retard, le nombre de minutes de retard
- les variables *_ct qui donnent pour chaque cause de retard, la proportion de vols en retard

Par exemple, pour un avion en retard de 30 minutes, si 15 mn sont imputables au temps et 15 mn à la compagnie, on incrémentera de 15 les variables weather_delay et carrier_delay, et de 0.5 les variables weather_ct et carrier_ct.

Seulement, ce sont des variables qui ne sont connues qu'une fois que l'avion a atteri. En outre, ce sont des variables qui composent directement la variable à prédire arr_delay (arr_delay = weather_delay + nas_delay + ...) ou arr_del15 (arr_del15 = weather_ct + security_ct + ...), mais n'expliquent pas réellement le délais puisque si on sait, après coup, quelles valeurs elles prennent, on ne peut pas prédire avant même le départ de l'avion, quelles valeurs elles prendront. On ne peut donc pas s'en servir pour prédire, avant le départ de l'avion, le retard éventuel de l'avion. Ces variables n'interviennent donc pas dans les données réelles qui créent le retard, elles ne sont qu'un retard calculé a posteriori.

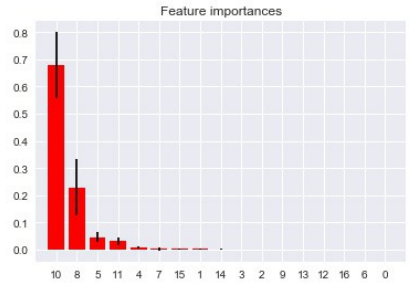
On a donc ici un cas de fuite de données, de data leakage qui, s'il permet d'obtenir un très bon score, comme on peut le voir ci-dessous, pose problème lors de prédiction de retards à venir.

Il a donc été nécessaire de trouver d'autres variables qui puissent échapper au risque de leakage, variables trouvées sur la page: https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time

C'est à partir de ces nouvelles variables qu'une exploration rapide a été faite et sur lesquelles a été basée la recherche de modèles de prédiction.

In [14]:
```
randomForestRegressor(dfOriginel, ['arr_del15', 'arr_delay'], "exclude", ['arr_delay'], 'arr_del15', 100, ['carrier', 'airport'], True)
```

```
('temps RandomForestRegressor = ', 171.32408822680026)
Le score r2 du random forest regressor est de : 0.992565601566
Le score mean_absolute_error du random forest regressor est de : 0.00867297930808
Feature ranking:
1. feature 10 nas_ct (0.678525)
2. feature 8 late_aircraft_delay (0.229151)
3. feature 5 carrier_delay (0.045097)
4. feature 11 nas_delay (0.030694)
5. feature 4 carrier_ct (0.006689)
6. feature 7 late_aircraft_ct (0.002653)
7. feature 15 weather_delay (0.002065)
8. feature 1 arr_cancelled (0.001986)
9. feature 14 weather_ct (0.000952)
10. feature 3 arr_flights (0.000657)
11. feature 2 arr_diverted (0.000409)
12. feature 9 month (0.000257)
13. feature 13 security_delay (0.000218)
14. feature 12 security_ct (0.000212)
15. feature 16 year (0.000189)
16. feature 6 carrier_num (0.000127)
17. feature 0 airport_num (0.000117)
```



Out[14]: 0.99256560156615015

Commentaires :
0.992 pour un score r2 dont le meilleur score possible est 1, c'était vraiment excellent !!! Mais inutilisable pour des prédictions.

2- Utilisation d'un autre jeu de données

Pour éviter ce problème de leakage, il a donc fallu trouver des variables dont on puisse remplir les valeurs avant même le départ du vol afin de prédire son éventuel retard.

La page https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time nous permet d'accéder à d'autres variables des vols.

Comme il y avait énormément de données, j'ai pris comme exemple un seul mois d'une année (décembre 2016) ce qui correspond déjà à près de 500 000 enregistrements.

Parmi le choix possible de variables, j'ai choisi comme variable à prédire : ArrDelayMinutes exportée sous le nom ARR_DELAY_NEW (Difference in minutes between scheduled and actual arrival time. Early arrivals set to 0.) C'est donc la seule variable connue après l'atterrissage d'un avion.

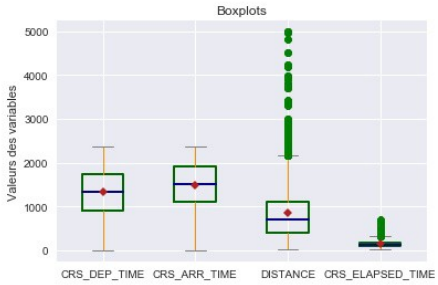
- Je n'ai donc pas retenu les variables qui auraient provoqué du leakage, comme les variables des blocs Departure Performance et Arrival Performance (sauf les variables CRSDepTime et CRSArrTime qui sont les horaires de départ et d'arrivée prévus par le système électronique)
- Je n'ai pas pris non plus les variables concernant l'annulation ou la déviation des vols car pour ces cas particuliers, la variable à prédire n'est pas renseignée.
- De même que je n'ai pas pris les variables décrivant les causes après coup de retard (leakage)

Data cleaning

- Il n'y a guère eu de data cleaning à faire, si ce n'est supprimer les enregistrements qui n'avaient pas de valeur à prédire. Cette suppression a eu pour effet de supprimer les quelques valeurs vides qui existaient sur certaines variables. Le dataset est donc finalement sans aucune valeur manquante. Si le nombre de valeurs extrêmes est important, il n'y a cependant aucune donnée aberrante.

In [15]:
```
dfBoxPlot(dfOriginelNew, ['CRS_DEP_TIME', 'CRS_ARR_TIME', 'DISTANCE', 'CRS_ELAPSED_TIME'], "Boxplots")
```

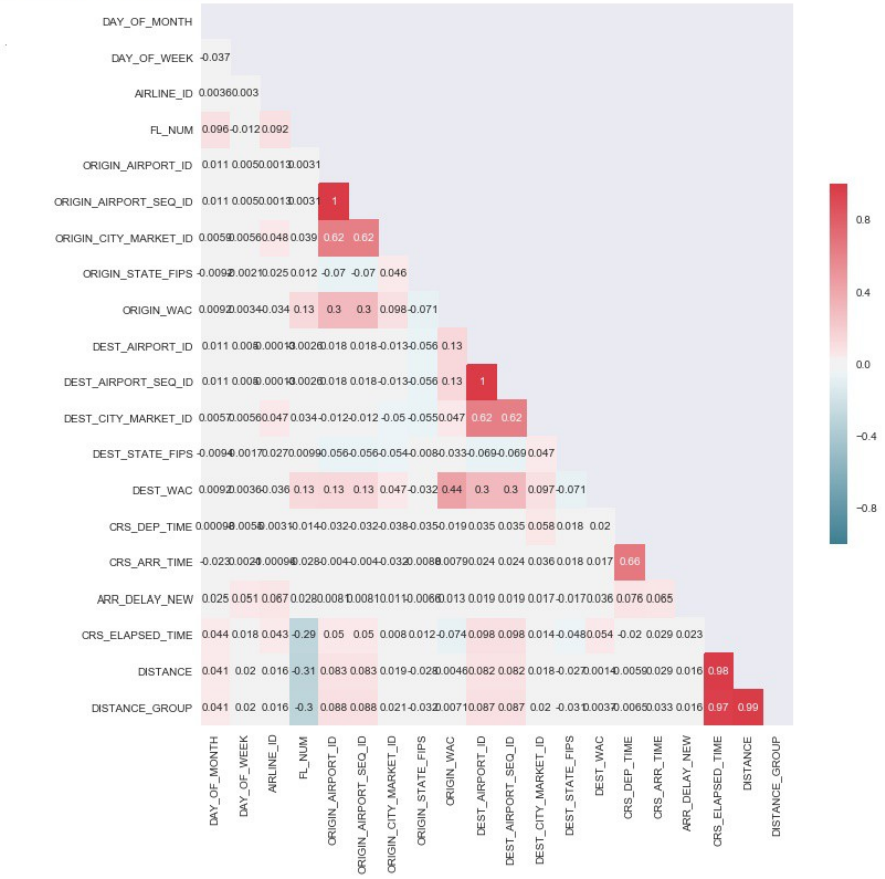
```
<matplotlib.figure.Figure at 0x486e5c0>
```



Feature engineering

- En ce qui concerne le feature engineering, la matrice de corrélation des variables a permis de voir qu'on pouvait supprimer deux variables, ORIGIN_AIRPORT_ID et DEST_AIRPORT_ID.

In [16]:
```
heatmap(dfOriginelNew, "pearson")
```



In [17]:
```
# suppression des variables ORIGIN_AIRPORT_ID et DEST_AIRPORT_ID
dfOriginelNew.drop('ORIGIN_AIRPORT_ID', axis=1, inplace=True)
dfOriginelNew.drop('DEST_AIRPORT_ID', axis=1, inplace=True)
```

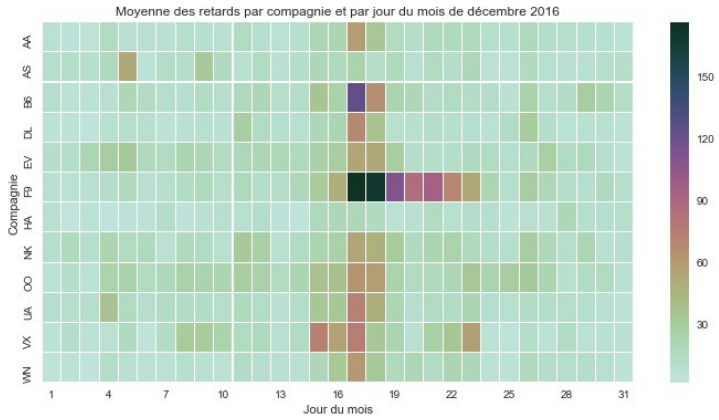
Voici la description des variables retenues pour la première esquisse du modèle: (elles sont de trois catégories)

- texte
- numérique avec sens "quantitatif" (peut être comparé avec les autres valeurs de la variable)
- numérique avec sens "qualitatif" (ne peut être comparé avec les autres valeurs de la variable)

Nom de la variable	Type de la variable	Description de la variable
DAY_OF_MONTH	NUM QUANT	Jour du mois
DAY_OF_WEEK	NUM QUANT	Jour de la semaine
UNIQUE_CARRIER	TEXT	Numéro unique de compagnie (si un code a été utilisé par plusieurs compagnie aériennes, un suffixe lui est appposé)
AIRLINE_ID	NUM QUAL	Numéro d'identification unique de la compagnie, indépendamment de son code, nom etc.
CARRIER	TEXT	code de la compagnie aérienne
TAIL_NUM	TEXT	numéro sur l'empennage de l'avion
FL_NUM	NUM QUAL	numéro du vol
ORIGIN_AIRPORT_SEQ_ID	NUM QUAL	Séquence unique identifiant l'aéroport de départ du vol.
ORIGIN_CITY_MARKET_ID	NUM QUAL	Identifiant du marché de la ville de l'aéroport de départ du vol.
ORIGIN	TEXT	Code de l'aéroport de départ d'un vol.
ORIGIN_CITY_NAME	TEXT	Nom de la ville de l'aéroport de départ d'un vol.
ORIGIN_STATE_ABR	TEXT	Abréviation de l'état de l'aéroport de départ d'un vol.
ORIGIN_STATE_FIPS	NUM QUAL	Code FIPS (Federal Information Processing Standards) de l'état de l'aéroport de départ d'un vol.
ORIGIN_STATE_NM	TEXT	Nom de l'état de l'aéroport de départ d'un vol.
ORIGIN_WAC	NUM QUAL	Code de la zone (World Area Code) de l'aéroport de départ d'un vol.
DEST_AIRPORT_SEQ_ID	NUM QUAL	Séquence unique identifiant l'aéroport d'arrivée du vol.
DEST_CITY_MARKET_ID	NUM QUAL	Identifiant du marché de la ville de l'aéroport d'arrivée du vol.
DEST	TEXT	Code de l'aéroport d'arrivée d'un vol.
DEST_CITY_NAME	TEXT	Nom de la ville de l'aéroport d'arrivée d'un vol.
DEST_STATE_ABR	TEXT	Abréviation de l'état de l'aéroport d'arrivée d'un vol.
DEST_STATE_FIPS	NUM QUAL	Code FIPS (Federal Information Processing Standards) de l'état de l'aéroport d'arrivée d'un vol.
DEST_STATE_NM	TEXT	Nom de l'état de l'aéroport d'arrivée d'un vol.
DEST_WAC	NUM QUAL	Code de la zone (World Area Code) de l'aéroport d'arrivée d'un vol.
CRS_DEP_TIME	NUM QUANT	Heure de départ du vol, prévue par le système numérique de réservation (Computerized Reservations Systems)
CRS_ARR_TIME	NUM QUANT	Heure d'arrivée du vol, prévue par le système numérique de réservation (Computerized Reservations Systems)
ARR_DELAY_NEW	NUM QUANT	Retard réel, différence en minutes entre l'arrivée prévue et l'arrivée réelle du vol. Les arrivées en avance, de valeur négative, sont mises à 0.
CRS_ELAPSED_TIME	NUM QUANT	= CRS_ARR_TIME – CRS_DEP_TIME, temps prévu de vol sur CRS
DISTANCE	NUM QUANT	Distance prévue du vol.
DISTANCE_GROUP	NUM QUANT	Nombre d'intervales de distance de 250 miles pour la distance prévue du vol.

Retard pour notre dataset du mois de décembre 2016

In [18]:
```
pivot(dfOriginelNew)
```



Le 17 décembre a été une journée difficile dans les aéroports américains.

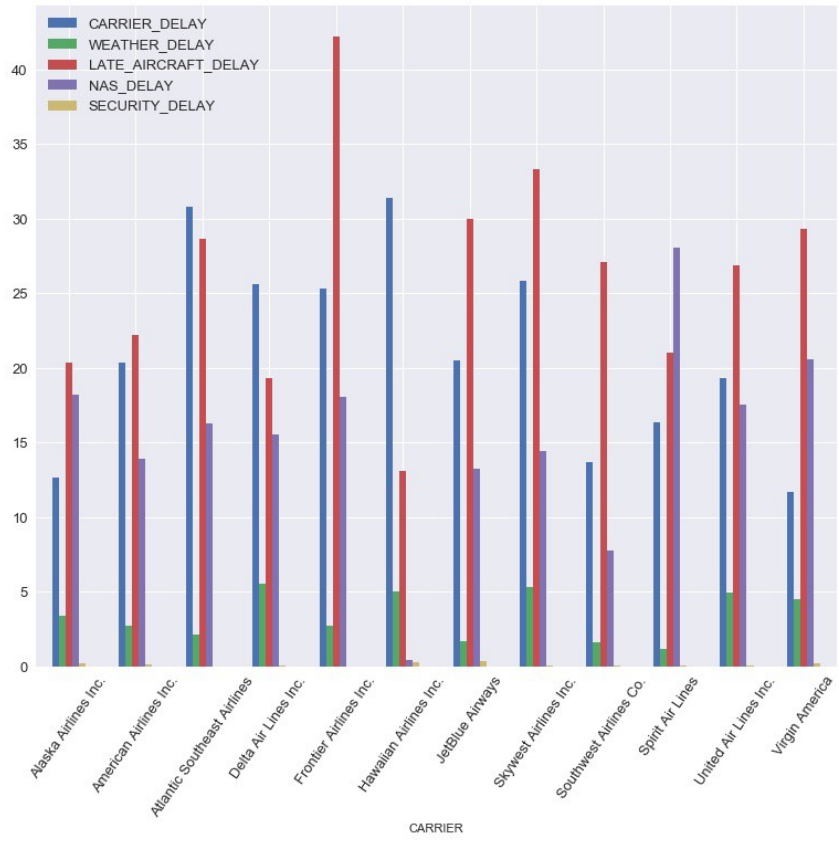
Causes des retards

Même si les variables relatant les causes des retards ne sont analysables qu'a posteriori et donc ne peuvent entrer dans la composition des variables explicatives des retards (leakage), on peut se faire une idée de ces causes et ainsi voir quelles informations seraient nécessaires pour bien prévoir les retards.

Comme ces variables ne pouvaient être introduites dans le dataset des variables explicatives, j'ai fait une autre extraction à partir du meme site :
https://www.transtats.bts.gov/DL_SelectFields.asp?Table_ID=236&DB_Short_Name=On-Time
Les causes décrites sont de cinq ordres:

- les retards dus à la compagnie (CARRIER_DELAY)
- les retards dus au temps (WEATHER_DELAY)
- les retards dus au fait que l'avion précédant l'avion en retard et utilisant la même piste était déjà en retard (LATE_AIRCRAFT_DELAY)
- les retards dus au National Air System (NAS_DELAY)
- les retards dus aux opérations de sécurité (SECURITY_DELAY)

In [19]:
```
grapheCausesRetards(dfDelays)
```



On observe ainsi que les problèmes météorologiques et sécuritaires sont vraiment minoritaires dans le retard des avions.
Une grande partie des retards est due à l'effet "boule de neige" des retards, aux files d'attente d'avions utilisant les mêmes pistes.

- Il conviendrait donc pour de meilleures prévisions du retard, nous en reparlerons en conclusion, d'intégrer dans les données des variables indiquant par exemple le mauvais temps, l'embouteillage des pistes, les zones de risque sécuritaires. La difficulté vient bien sûr que ce sont presque des données "en temps réel" et qu'il faudrait accéder à d'autres bases de données.

Si on prend par exemple le cas de thalès: https://www.thalesgroup.com/fr/worldwide/big-data/solutions/smarter-data-pour-le-transport
Ils utilisent comme données:

- données d'observation ou de prévisions météorologiques (force et orientation des vents, pluies, impacts de foudre, etc.),
- position des avions au-dessus du sol européen, recueillies grâce au système ADS-B (Automatic Dependent Surveillance-Broadcast),
- données de vol (route, type d'avion, altitude, etc.) fournies par Eurocontrol, l'organisme européen

Ils indiquent que les analyses permettent de mettre en évidence les données ayant un impact fort sur les retards - la météo, le taux d'occupation des pistes (notion d'heure de pointe), le retard moyen des avions sur l'heure précédente, notamment.

3- Proposition de différents modèles de régression

(entraînement et évaluation des performances de modèles de régression)

Choix des variables

Pour les algorithmes de régression, on peut prendre des variables numériques ou/et transformer des variables textes en variables numériques.

- pour les variables NUM QUANT, pas de problème, on peut les intégrer
- pour les variables TEXT, on peut les transformer en nombres quand cela a un sens. Par exemple, transformer le nom de l'état en variable numérique ne semble pas très utile vis-à-vis du retard d'un avion. Par contre, les codes d'un aéroport ou d'une compagnie doivent être intégrés dans le modèle puisque l'aéroport ou la compagnie peuvent influer sur le retard d'un avion. On utilise alors par exemple un label encoder qui permet de relier les avions d'un même aéroport ou d'une même compagnie même si bien sûr, cela ne transforme pas la variable en véritable variable numérique quantitative mais en variable numérique qualitative
- pour les variables NUM QUAL, ce sont des variables assimilables à des codes, donc à des variables textes, on peut choisir de les transformer ou non en des variables numériques avec un label encoder ou un one hot encoder. Par rapport à une régression, le one hot encoder a plus de sens que le label encoder, toutefois, s'il y a trop de valeurs possibles, les distinctions sont tellement minimes que cela n'apporte guère d'information discriminante.

Pour un premier essai, on peut de prendre que les variables numériques (QUANT et QUAL) et laisser l'algorithme décider des variables à retenir.

J'ai testé 3 méthodes, la régression linéaire, le random forest et le gradient boosting.

Régression linéaire

La régression linéaire (multivariée dans notre cas) est un algorithme d'apprentissage supervisé cherchant à prédire une variable donnée à l'aide de variables explicatives en minimisant la fonction de coût qui relie toutes ces variables.

In [20]:
```
regessionLineaire(dfOriginelNew)
```

```
Variables explicatives : Index([u'AIRLINE_ID', u'CRS_ARR_TIME', u'CRS_DEP_TIME', u'CRS_ELAPSED_TIME',
       u'DAY_OF_MONTH', u'DAY_OF_WEEK', u'DEST_AIRPORT_SEQ_ID',
       u'DEST_CITY_MARKET_ID', u'DEST_STATE_FIPS', u'DEST_WAC', u'DISTANCE',
       u'DISTANCE_GROUP', u'FL_NUM', u'ORIGIN_AIRPORT_SEQ_ID',
       u'ORIGIN_CITY_MARKET_ID', u'ORIGIN_STATE_FIPS', u'ORIGIN_WAC'],
      dtype='object')

Score r2 de la prédiction : 0.0163875606329
```

In [21]:
```
arrAlgo.append("Reg. lin.")
arrScore.append(0.0163875606329)
```

Ce score proche de 0.016 est loin de 1, donc pas très bon. Les données numériques du jeu de données ne permettent pas d'expliquer la variable ARR_DELAY_NEW. Nous allons voir si une méthode non linéaire comme l'algorithme du Random forest permet d'obtenir de meilleurs résultats.

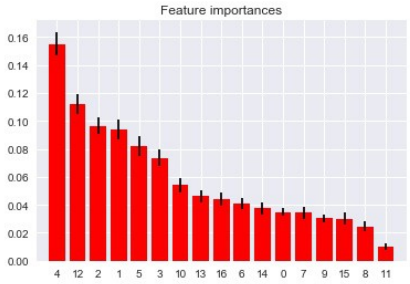
Random Forest Régression

Le Random forest est une méthode ensembliste basée sur les arbres de décisions et qui utilise un ensemble d'estimateurs ayant chacun une vision parcellaire du problème mais dont l'ensemblage fournit une vision globale.
Si la performance d'un arbre isolé dépend trop de l'échantillon de départ, l'utilisation d'une forêt d'arbres construits sur la base d'un tirage aléatoire permet un gain notable de performance.

Lors du premier essai, j'ai utilisé toutes les variables du dataset (sauf MONTH et YEAR puisque leurs valeurs sont constantes par choix). La variable à prédire est ARR_DELAY_NEW. Elle est donc exclue des variables explicatives.
C'est l'algorithme qui choisit les variables les plus importantes.

In [22]:
```
randomForestRegressor(dfOriginelNew, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, [], True)
```

```
('temps RandomForestRegressor = ', 351.78539882879943)
Le score r2 du random forest regressor est de : 0.0860852438342
Le score mean_absolute_error du random forest regressor est de : 0.411030823987
Feature ranking:
1. feature 4 DAY_OF_MONTH (0.155065)
2. feature 12 FL_NUM (0.112158)
3. feature 2 CRS_DEP_TIME (0.096333)
4. feature 1 CRS_ARR_TIME (0.094003)
5. feature 5 DAY_OF_WEEK (0.081799)
6. feature 3 CRS_ELAPSED_TIME (0.073553)
7. feature 10 DISTANCE (0.054090)
8. feature 13 ORIGIN_AIRPORT_SEQ_ID (0.046157)
9. feature 16 ORIGIN_WAC (0.044052)
10. feature 6 DEST_AIRPORT_SEQ_ID (0.040864)
11. feature 14 ORIGIN_CITY_MARKET_ID (0.037549)
12. feature 0 AIRLINE_ID (0.034854)
13. feature 7 DEST_CITY_MARKET_ID (0.034255)
14. feature 9 DEST_WAC (0.030569)
15. feature 15 ORIGIN_STATE_FIPS (0.030137)
16. feature 8 DEST_STATE_FIPS (0.024402)
17. feature 11 DISTANCE_GROUP (0.010160)
```



Out[22]: 0.08608524383417504

Le Random Forest Regressor a donc permis d'améliorer le score du modèle avec 0.086 contre 0.016 pour la régression linéaire.
Toutefois, on est encore loin de 1. On peut quand même garder cet algorithme et essayer de l'améliorer.

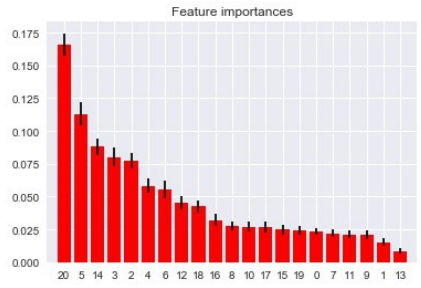
In [23]:
```
arrAlgo.append("Ran. for. All Var.")
arrScore.append(0.08608524383417504)
```

La hiérarchie des variables indique donc que le jour du mois "explique" mieux que les autres variables le retard d'avion. On voit d'ailleurs que les variables de positionnement temporel prédominent par rapport aux codes des états ou des aéroports, ce qui semble normal sans traitement de ces codes éventuellement par un label encoder.
Toutefois, que la deuxième variable la plus importante soit le numéro du vol... (il semble difficile d'établir un lien entre un numéro de vol et le retard de ce vol) semble indiquer que l'utilisation de variables NUM QUAL dans le modèle n'est pas forcément une bonne chose.

Afin d'améliorer le modèle, on peut envisager d'intégrer un label encoder du code compagnie (CARRIER) puisqu'une compagnie peut être une habituée des retard, mais aussi des codes aéroport (ORIGIN et DEST) où un aéroport situé par exemple en altitude peut subir des retards répétés à cause du mauvais temps, de même qu'un label encoder de TAIL_NUM, le numéro écrit sur l'empennage des avions (puisque le numéro de vol semble important, pourquoi pas le numéro sur l'empennage).

In [24]:
```
randomForestRegressor(dfOriginelNew, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, ['CARRIER', 'ORIG
IN', 'DEST', 'TAIL_NUM'], True)
```

```
('temps RandomForestRegressor = ', 455.1445207484528)
Le score r2 du random forest regressor est de : 0.11779982048
Le score mean_absolute_error du random forest regressor est de : 0.40941160246
Feature ranking:
1. feature 20 TAIL_NUM_num (0.166123)
2. feature 5 DAY_OF_MONTH (0.112990)
3. feature 14 FL_NUM (0.087992)
4. feature 3 CRS_DEP_TIME (0.080260)
5. feature 2 CRS_ARR_TIME (0.077269)
6. feature 4 CRS_ELAPSED_TIME (0.058274)
7. feature 6 DAY_OF_WEEK (0.055199)
8. feature 12 DISTANCE (0.045151)
9. feature 18 ORIGIN_WAC (0.042390)
10. feature 16 ORIGIN_CITY_MARKET_ID (0.032169)
11. feature 8 DEST_CITY_MARKET_ID (0.027906)
12. feature 10 DEST_WAC (0.027040)
13. feature 17 ORIGIN_STATE_FIPS (0.026675)
14. feature 15 ORIGIN_AIRPORT_SEQ_ID (0.024922)
15. feature 19 ORIGIN_num (0.024540)
16. feature 0 AIRLINE_ID (0.023442)
17. feature 7 DEST_AIRPORT_SEQ_ID (0.021952)
18. feature 11 DEST_num (0.021216)
19. feature 9 DEST_STATE_FIPS (0.020814)
20. feature 1 CARRIER_num (0.015148)
21. feature 13 DISTANCE_GROUP (0.008527)
```



Out[24]: 0.11779982048031445

Et il s'avère que le score a été un petit peu amélioré 0.117 au lieu de 0.086) avec l'ajout des label encoder de CARRIER, ORIGIN (aéroport de départ du vol) et DEST (aéroport d'arrivée du vol), mais surtout de TAIL_NUM qui prend la première place, les autres variables ajoutées n'apparaissant qu'à partir de la 15ème position.

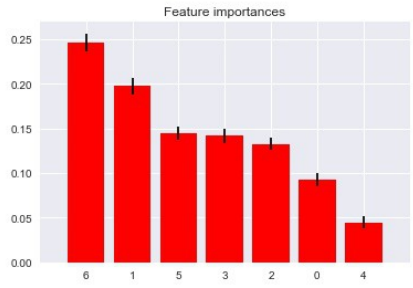
In [25]:
```
arrAlgo.append("Ran. for. Tail Num.")
arrScore.append(0.11779982048031445)
```

On peut essayer de ne retenir que les meilleures variables :

In [26]:
```
randomForestRegressor(dfOriginelNew, ['DAY_OF_MONTH', 'FL_NUM', 'CRS_DEP_TIME', 'CRS_ARR_TIME', 'DAY_OF_WEEK', 'CRS_ELAPSED_TIME', 'TAIL_NUM_num'], "include", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, ['TAIL_NUM'], True)
```

```
('temps RandomForestRegressor = ', 212.61646246576288)
Le score r2 du random forest regressor est de : 0.0516818953931
Le score mean_absolute_error du random forest regressor est de : 0.444963570967
Feature ranking:

1. feature 6 TAIL_NUM_num (0.246631)
2. feature 1 FL_NUM (0.197447)
3. feature 5 CRS_ELAPSED_TIME (0.144516)
4. feature 3 CRS_ARR_TIME (0.141624)
5. feature 2 CRS_DEP_TIME (0.132779)
6. feature 0 DAY_OF_MONTH (0.092161)
7. feature 4 DAY_OF_WEEK (0.044842)
```



Out[26]: 0.051681895393099309

Mais chaque variable représente une partie du problème. Et si cela améliore nettement le temps de traitement, cela diminue grandement le score. Ce n'est donc pas une bonne option.

Puisque l'ajout de variables améliore le modèle, mais de façon non flagrante (on est encore loin de 1), on peut essayer de jouer sur les paramètres du Random Forest Regressor.

Un des paramètres les plus importants du random forest est n_estimators, le nombre d'arbres de la forêt. Précédemment, il était à 100, ici on le passe à 200.

In [27]:
```
randomForestRegressor(dfOriginelNew, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 200, ['CARRIER', 'ORIGIN', 'DEST', 'TAIL_NUM'], False)
```

```
('temps RandomForestRegressor = ', 927.4913084410107)
Le score r2 du random forest regressor est de : 0.118816779522
Le score mean_absolute_error du random forest regressor est de : 0.408531846907
```

Out[27]: 0.11881677952150493

On constate qu'avoir augmenté le nombre d'arbres dans le Random Forest Regressor a amélioré le score, mais en contre-partie, le temps d'exécution a nettement augmenté. En outre, dans le compromis temps / score, la faible augmentation du score ne contre-balance pas la nette augmentation du temps d'exécution.

In [28]:
```
arrAlgo.append("Ran. for. Tail Num. 200 trees")
arrScore.append(0.11881677952150493)
```

On peut essayer un troisième algorithme de régression, le Gradient boosting (version régression).

Gradient boosting

Le gradient boosting est également une méthode ensembliste qui se base sur un ensemble d'arbres de décisions. Il combine les méthodes de boosting et de descente de gradient qui permet l'utilisation de nombreuses fonctions de coût (à la différence de la régression linéaire qui n'en utilise qu'une).

In [29]:
```
gradientBoostingRegressor(dfOriginelNew, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, ['CARRIER', '
ORIGIN', 'DEST', 'TAIL_NUM'])
```

```
('temps GradientBoostingRegressor = ', 123.96724936977762)
Le score r2 du gradient boosting regressor est de : 0.054065789246

Feature Importances
1. feature 5 DAY_OF_MONTH (0.605757)
2. feature 6 DAY_OF_WEEK (0.109854)
3. feature 0 AIRLINE_ID (0.102708)
4. feature 3 CRS_DEP_TIME (0.064407)
5. feature 2 CRS_ARR_TIME (0.044505)
6. feature 10 DEST_WAC (0.026304)
7. feature 18 ORIGIN_WAC (0.024139)
8. feature 14 FL_NUM (0.006978)
9. feature 4 CRS_ELAPSED_TIME (0.005687)
10. feature 1 CARRIER_num (0.003222)
11. feature 17 ORIGIN_STATE_FIPS (0.003105)
12. feature 20 TAIL_NUM_num (0.002349)
13. feature 12 DISTANCE (0.000764)
```

Si le gradient boosting a amélioré (0.054) le score de la régression linéaire (0.016), il est en deçà du random forest (0.118).

In [30]:
```
arrAlgo.append("Grad. boost.")
arrScore.append(0.054065789246)
```

4- Recherche des meilleurs paramètres

(adaptation des paramètres d'un modèle de régression afin d'amélioration)

Quand jouer sur les variables ne permet plus d'améliorer un modèle, on peut chercher à trouver les meilleurs paramètres d'un algorithme qui vont permettre d'optimiser les résultats. On peut le faire par exemple avec le gradient boosting.

Grid search du gradient boosting

Comme l'utilisation d'un GridSearchCV est très longue puisqu'il teste l'algorithme avec les différents paramètres, j'ai restreint le nombre d'enregistrements afin que cela donne une idée du résultat. Et pour comparaison des scores, je lance un gradientBoostingRegressor avec seulement 5000 enregistrements:

In [31]:
```
dfOriginelNew2 = dfOriginelNew.iloc[0:5000,:]
gradientBoostingRegressor(dfOriginelNew2, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, ['CARRIER',
'ORIGIN', 'DEST', 'TAIL_NUM'])
```

```
('temps GradientBoostingRegressor = ', 0.31619686565818483)
Le score r2 du gradient boosting regressor est de : -0.0337508768711

Feature Importances
1. feature 5 DAY_OF_MONTH (0.325632)
2. feature 17 ORIGIN_STATE_FIPS (0.237518)
3. feature 20 TAIL_NUM_num (0.169094)
4. feature 4 CRS_ELAPSED_TIME (0.067765)
5. feature 3 CRS_DEP_TIME (0.041696)
6. feature 19 ORIGIN_num (0.036283)
7. feature 16 ORIGIN_CITY_MARKET_ID (0.029860)
8. feature 18 ORIGIN_WAC (0.025533)
9. feature 2 CRS_ARR_TIME (0.023269)
10. feature 6 DAY_OF_WEEK (0.011811)
11. feature 15 ORIGIN_AIRPORT_SEQ_ID (0.009654)
12. feature 9 DEST_STATE_FIPS (0.006525)
13. feature 0 AIRLINE_ID (0.006311)
14. feature 1 CARRIER_num (0.004923)
15. feature 12 DISTANCE (0.003314)
16. feature 10 DEST_WAC (0.000591)
17. feature 14 FL_NUM (0.000219)
18. feature 7 DEST_AIRPORT_SEQ_ID (0.000000)
19. feature 8 DEST_CITY_MARKET_ID (0.000000)
20. feature 11 DEST_num (0.000000)
21. feature 13 DISTANCE_GROUP (0.000000)
```

Avec seulement 5000 enregistrements, on obtient un score de -0.0337508768711

In [32]:
```
dfOriginelNew2 = dfOriginelNew.iloc[0:5000,:]
gridSearchCVGradientBoosting(dfOriginelNew2)
```

```
Index([u'AIRLINE_ID', u'CARRIER_num', u'CRS_ARR_TIME', u'CRS_DEP_TIME',
       u'CRS_ELAPSED_TIME', u'DAY_OF_MONTH', u'DAY_OF_WEEK',
       u'DEST_AIRPORT_SEQ_ID', u'DEST_CITY_MARKET_ID', u'DEST_STATE_FIPS',
       u'DEST_WAC', u'DEST_num', u'DISTANCE', u'DISTANCE_GROUP', u'FL_NUM',
       u'ORIGIN_AIRPORT_SEQ_ID', u'ORIGIN_CITY_MARKET_ID',
       u'ORIGIN_STATE_FIPS', u'ORIGIN_WAC', u'ORIGIN_num', u'TAIL_NUM_num'],
      dtype='object')
('temps RandomForestRegressor = ', 189.07917356214284)
('best_params_', {'max_features': 1.0, 'n_estimators': 300, 'max_depth': 6, 'min_samples_leaf': 5})
('best_score_ = ', 0.08629643600809556)
```

On voit que cette recherche (limitée) des meilleurs paramètres a déjà permis d'améliorer le score de -0.03 à 0.08.

Grid search du random forest

In [33]:
```
dfOriginelNew2 = dfOriginelNew.iloc[0:10000,:]
randomForestRegressor(dfOriginelNew2, ['ARR_DELAY_NEW'], "exclude", ['ARR_DELAY_NEW'], 'ARR_DELAY_NEW', 100, ['CARRIER', 'ORI
GIN', 'DEST', 'TAIL_NUM'], False)
```

```
('temps RandomForestRegressor = ', 5.609019104797881)
Le score r2 du random forest regressor est de : -0.0356435821982
Le score mean_absolute_error du random forest regressor est de : 0.454667978424
```

Out[33]:
```
-0.035643582198216617
```

In [34]:
```
# test avec seulement 10000 enregistrements
dfOriginelNew2 = dfOriginelNew.iloc[0:10000,:]
gridSearchCVGradientBoosting(dfOriginelNew2)
```

```
Index([u'AIRLINE_ID', u'CARRIER_num', u'CRS_ARR_TIME', u'CRS_DEP_TIME',
       u'CRS_ELAPSED_TIME', u'DAY_OF_MONTH', u'DAY_OF_WEEK',
       u'DEST_AIRPORT_SEQ_ID', u'DEST_CITY_MARKET_ID', u'DEST_STATE_FIPS',
       u'DEST_WAC', u'DEST_num', u'DISTANCE', u'DISTANCE_GROUP', u'FL_NUM',
       u'ORIGIN_AIRPORT_SEQ_ID', u'ORIGIN_CITY_MARKET_ID',
       u'ORIGIN_STATE_FIPS', u'ORIGIN_WAC', u'ORIGIN_num', u'TAIL_NUM_num'],
      dtype='object')
('temps RandomForestRegressor = ', 296.51990104489914)
('best_params_', {'max_features': 1.0, 'n_estimators': 10, 'max_depth': 6, 'min_samples_leaf': 9})
('best_score_ = ', 0.046213829327586727)
```

On voit également une amélioration grâce entre autres au passage de 100 à 200 arbres.

5- Blending de différents algorithmes

(combinaison de plusieurs modèles de régression afin d'en améliorer les performances)

De la même façon que différents estimateurs peuvent avoir une vue parcellaire des données, on peut tester si une combinaison linéaire de différents algorithmes ne permettrait pas d'améliorer le modèle "en ne prenant que le meilleur de chacun".
On va donc utiliser la méthode de blending qui va associer à chaque algorithme particulier un poids et estimer quelle est la meilleure des pondérations.

In [35]:
```
X, y, X_train_scaler, y_train_scaler, reg_rfr, reg_gbr, reg_lr, GLOBAL_MEAN, res = blending(dfOriginelNew)
```

```
Index([u'AIRLINE_ID', u'CARRIER_num', u'CRS_ARR_TIME', u'CRS_DEP_TIME',
       u'CRS_ELAPSED_TIME', u'DAY_OF_MONTH', u'DAY_OF_WEEK',
       u'DEST_AIRPORT_SEQ_ID', u'DEST_CITY_MARKET_ID', u'DEST_STATE_FIPS',
       u'DEST_WAC', u'DEST_num', u'DISTANCE', u'DISTANCE_GROUP', u'FL_NUM',
       u'ORIGIN_AIRPORT_SEQ_ID', u'ORIGIN_CITY_MARKET_ID',
       u'ORIGIN_STATE_FIPS', u'ORIGIN_WAC', u'ORIGIN_num', u'TAIL_NUM_num'],
      dtype='object')
LinearRegression score = 0.017287575515
GradientBoostingRegressor score = 0.0849594288079
RandomForestRegressor score = 0.122363521009
Les poids des différents algorithmes sont : [-0.10491159  0.45924141  0.5936957  -0.02842374]
Le score r2 pour cette pondération est : 0.158671017124
```

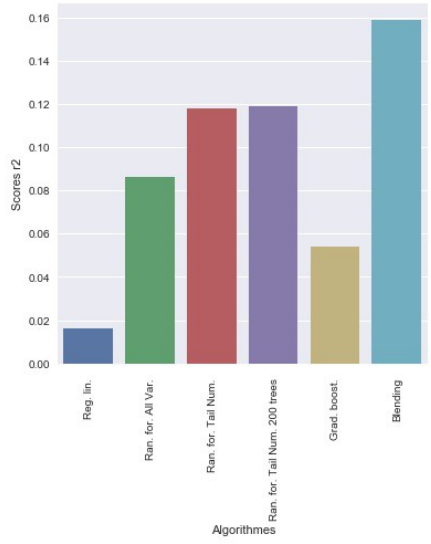
On observe bien qu'on a une amélioration du meilleur des scores (max(0.017, 0.084, 0.122) --> 0.158) grâce au blending de ces 3 algorithmes.

In [36]:
```
arrAlgo.append("Blending")
arrScore.append(0.158671017124)
```

6- Choix du modèle et exemples de prédiction

(Sélection du modèle de régression approprié)

In [37]:
```
graphScores(arrAlgo, arrScore)
```



Résultats:

- de tous les modèles essayés, c'est donc la combinaison linéaire des trois modèles (régression linéaire, random forest, gradient boosting) qui est la plus performante
- même si cela nécessite de lancer les 3 algorithmes et donc de cumuler les temps d'exécution, le gain de performance est appréciable, même si on est encore loin du score r2 de 1
- la combinaison des trois modèles est donc retenue

Prenons par exemple le premier avion qui ait du retard : la valeur réelle est de 3 minutes.

In [48]:
```
prediction(True, 1, X, y, X_train_scaler, y_train_scaler, reg_rfr, reg_gbr, reg_lr, GLOBAL_MEAN, res)
```

```
AIRLINE_ID                 19790.0
CARRIER_num                    3.0
CRS_ARR_TIME                1259.0
CRS_DEP_TIME                1104.0
CRS_ELAPSED_TIME             115.0
DAY_OF_MONTH                  30.0
DAY_OF_WEEK                    5.0
DEST_AIRPORT_SEQ_ID      1198603.0
DEST_CITY_MARKET_ID        31986.0
DEST_STATE_FIPS               26.0
DEST_WAC                      43.0
DEST_num                     118.0
DISTANCE                     640.0
DISTANCE_GROUP                 3.0
FL_NUM                      2038.0
ORIGIN_AIRPORT_SEQ_ID    1039705.0
ORIGIN_CITY_MARKET_ID      30397.0
ORIGIN_STATE_FIPS             13.0
ORIGIN_WAC                    34.0
ORIGIN_num                    17.0
TAIL_NUM_num                3976.0
Name: 1, dtype: float64
ARR_DELAY_NEW    3.0
Name: 1, dtype: float64

Valeur réelle = ARR_DELAY_NEW    3.0
Name: 1, dtype: float64

Valeur prédite = 2.78228298342
```

Pour cet avion, alors que la valeur réelle était de 3.0, on a une prédiction à 2.78228298342

Si on prend cet avion complètement inventé :

In [49]:
```
X_enreg = [20302, 9, 1905, 1656, 125, 12, 1, 1345502, 31650, 28, 63, 199, 596, 4, 2500, 1591902, 31834, 5, 71, 294, 1639]
```

In [50]:
```
prediction(False, 0, X_enreg, 0, X_train_scaler, y_train_scaler, reg_rfr, reg_gbr, reg_lr, GLOBAL_MEAN, res)
```

```
Valeur prédite = 56.8596761525
```

On prévoit cette fois-ci un retard de 56 minutes.

7- API sur AWS et tests

L'API consiste en un service REST / Flask hébergé sur AWS (Amazon Web Service) appelé par HTTP et qui rend le résultat sous format Json.

Comme le temps d'exécution du blending est assez long, qu'une requête HTTP peut tomber rapidement en timeout, et que l'utilisateur n'a peut-être pas envie d'attendre 20 minutes la réponse de sa requête, j'ai utilisé la librairie joblib qui permet de faire un dump de différentes données, comme les modèles. L'utilisateur n'est donc pas obligé d'attendre l'entraînement du modèle. Celui-ci est fait une première fois, mis en persistance, et l'utilsateur ne fait plus que lancer la partie prédiction.

L'API transforme les données texte en données numériques car l'utilisateur ne sait pas quel est le code utilisé par le label encoder. Il peut donc laisser les données textes en entrée pour les variables CARRIER, DEST, ORIGIN et TAIL_NUM.

Il faut absolument entrer les données dans le bon ordre. Cet ordre peut être obtenu à partir de l'API via l'url : http://127.0.0.1:5000/projet4/columns


```
{
    "1": "AIRLINE_ID",
    "2": "CARRIER",
    "3": "CRS_ARR_TIME",
    "4": "CRS_DEP_TIME",
    "5": "CRS_ELAPSED_TIME",
    "6": "DAY_OF_MONTH",
    "7": "DAY_OF_WEEK",
    "8": "DEST_AIRPORT_SEQ_ID",
    "9": "DEST_CITY_MARKET_ID",
    "10": "DEST_STATE_FIPS",
    "11": "DEST_WAC",
    "12": "DEST",
    "13": "DISTANCE",
    "14": "DISTANCE_GROUP",
    "15": "FL_NUM",
    "16": "ORIGIN_AIRPORT_SEQ_ID",
    "17": "ORIGIN_CITY_MARKET_ID",
    "18": "ORIGIN_STATE_FIPS",
    "19": "ORIGIN_WAC",
    "20": "ORIGIN",
    "21": "TAIL_NUM"
}
```

Exemple : pour cet avion, le retard réel est de 17 minutes. La prévision donne un peu moins de 12 minutes.
http://127.0.0.1:5000/projet4/delay/20366:EV:756:620:99:5:1:1226603:31453:48:74:IAH:427:2:4223:1342202:30562:1:51:MOB:N14203


```
{
    "Plane delay prediction": 11.741745360280255,
    "Plane initial data": [
        {
            "ORIGIN": "MOB",
            "CRS_ELAPSED_TIME": "99",
            "DEST_STATE_FIPS": "48",
            "DISTANCE": "427",
            "DEST": "IAH",
            "FL_NUM": "4223",
            "ORIGIN_AIRPORT_SEQ_ID": "1342202",
            "DAY_OF_MONTH": "5",
            "CRS_ARR_TIME": "756",
            "ORIGIN_STATE_FIPS": "1",
            "CRS_DEP_TIME": "620",
            "ORIGIN_CITY_MARKET_ID": "30562",
            "DEST_WAC": "74",
            "CARRIER": "EV",
            "DAY_OF_WEEK": "1",
            "AIRLINE_ID": "20366",
            "ORIGIN_WAC": "51",
            "DISTANCE_GROUP": "2",
            "DEST_CITY_MARKET_ID": "31453",
            "DEST_AIRPORT_SEQ_ID": "1226603",
            "TAIL_NUM": "N14203"
        }
    ]
}
```


8- Conclusions

- Ce projet a mis en exergue un problème délicat dans le choix des variables permettant d'expliquer une variable à prédire, le problème de data leakage (fuite des données). Utiliser pour son modèle des variables construites a posteriori (équivalent, en machine learning, d'une pétition de principe, en rhétorique) et par exemple à partir d'informations à prédire (comme ce fut le cas dans le premier jeu de données avec les variables *_delay et* _ct) a deux conséquences. Cela peut certes donner un très bon score (dans notre cas, score r2 de 0.992) et donner l'impression d'un très bon niveau de prédiction (ce qui était normal puisque les données étaient établies une fois l'information à prédire connue, le délais réel), toutefois ce modèle était inutilisable pour les prédictions puisque ces données n'étaient pas connues pour tout avion à venir.
- Il a donc été nécessaire de récupérer d'autres variables qu'on puisse utiliser pour les prédictions et qu'on puisse remplir avant que l'avion ne décolle afin d'en prédire, avant, l'éventuel retard.
- J'ai essayé trois différents modèles de régression, la régression linéaire, le random forest et le gradient boosting. Le meilleur s'est révélé être le random forest. Toutefois, avec des scores proches de 0.1, il fut nécessaire d'essayer d'améliorer le modèle. Le feature engineering (label encoder sur TAIL_NUM) a permis de l'améliorer un peu, mais plutôt de façon marginale. Jouer avec les paramètres du modèle a également donné quelques résultats. Toutefois, c'est le blending (mélange, combinaison linéaire) de ces trois algorithmes qui a donné le meilleur résultat, et c'est donc ce modèle que j'ai retenu pour prédire le retard des avions. Néanmoins, avec un score de 0.158, très loin de 1, le modèle n'est pas très performant.
- Cette performance pourrait être améliorée en incorporant dans le dataset, des informations qui semblent plus à même d'expliquer le retard des avions. Par exemple en intégrant des informations météorologiques puisque nous avons vu que le mauvais temps était une cause réelle, même si non principale, de retard. Les avions atterissant "à la chaîne" sur les mêmes pistes, la tendance aux embouteillages peut également être une information importante. On doit ainsi penser aux causes probables, intuitives (même s'il y a bien sûr souvent des causes cachées non immédiates et des événements exceptionnels) des retards et essayer de récupérer toutes les données s'y rapportant afin de se rapprocher de bonnes prédictions.

Code

In [47]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings("ignore", category=DeprecationWarning)
import seaborn as sns
%matplotlib inline
import math
import time
from matplotlib.gridspec import GridSpec
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import cross_val_score
from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn import linear_model
from sklearn.ensemble.gradient_boosting import GradientBoostingRegressor
from sklearn.cross_validation import ShuffleSplit
from sklearn.grid_search import GridSearchCV
from scipy.optimize import minimize
```

```
def getNoDuplication(_df, _col):
    return _df[np.logical_not(np.isnan(_df[_col]))]
def resetIndex(_df):
    _df.reset_index(drop=True, inplace=True)
    return _df
def getNum(_df):
    return _df.select_dtypes(include = ['float64', 'int64']).iloc[:, :]
def getX(_df, _colList, _inexclude):
    if _inexclude == "include":
        return _df[_colList]
    elif _inexclude == "exclude":
        return  _df[_df.columns.difference(_colList)]
def getY(_df, _colPredict):
    return _df[_colPredict]
def separateTrainTest(X, y):
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    return X_train, X_test, y_train, y_test
def normaliseXy(_X, _y):
    X_scaler = StandardScaler()
    X = X_scaler.fit_transform(_X)
    y_scaler = StandardScaler()
    y = y_scaler.fit_transform(_y)
    return X, y
def playRandomForestRegressor(_n_estimators, _X_train, _y_train):
    fitted_rfr = RandomForestRegressor(n_estimators=_n_estimators).fit(_X_train, _y_train)
    return fitted_rfr
def predictRandomForestRegressor(_fitted_rfr, _X_test):
    predict_rfr = _fitted_rfr.predict(_X_test)
    return predict_rfr
def r2Score(_y_test, _predict_rfr):
    r2Score = r2_score(_y_test, _predict_rfr)
    print("Le score r2 du random forest regressor est de : {0}".format(r2Score))
    return r2Score
def mae(_y_test, _predict_rfr):
    mae = mean_absolute_error(_y_test, _predict_rfr)
    print("Le score mean_absolute_error du random forest regressor est de : {0}".format(mae))
def importanceFeature(_fitted_rfr):
    importances = _fitted_rfr.feature_importances_
    std = np.std([tree.feature_importances_ for tree in _fitted_rfr.estimators_], axis=0)
    indices = np.argsort(importances)[::-1]
    return importances, indices, std
def grapheImportanceFeature(_X_train, _importances, _indices, _std, _col):
    # Print the feature ranking
    print("Feature ranking:")
    Xshape = _X_train.shape[1]
    for f in range(Xshape):
        print("%d. feature %d %s (%f)" % (f + 1, _indices[f], _col[_indices[f]], _importances[_indices[f]]))

    # Plot the feature importances of the forest
    plt.figure()
    plt.title("Feature importances")
    plt.bar(range(Xshape), _importances[_indices], color="r", yerr=_std[_indices], align="center")
    plt.xticks(range(Xshape), _indices)
    plt.xlim([-1, Xshape])
    plt.show()

def addLabelEncoder(_dfNum, _dfOriginelSansVide, _colLabelEncoder):
    dfNumColNum = _dfNum
    for col in _colLabelEncoder:
        le = preprocessing.LabelEncoder()
        le.fit(_dfOriginelSansVide[col])
        col_num = le.transform(_dfOriginelSansVide[col])
        dfColNum = pd.DataFrame({col+'_num' : col_num})
        dfNumColNum = pd.concat([dfNumColNum, dfColNum], axis = 1)
    return dfNumColNum
def randomForestRegressor(_df, _Xcol, _inexclude, _ycol, _colNoDuplic, _nbtrees, _colLabelEncoder, _printGraphe):
    dfOriginelSansVide = getNoDuplication(_df, _colNoDuplic)
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    if len(_colLabelEncoder) > 0:
        dfNum = addLabelEncoder(dfNum, dfOriginelSansVide, _colLabelEncoder)
    X = getX(dfNum, _Xcol, _inexclude)
    col = X.columns
    y = getY(dfNum, _ycol)
    X, y = normaliseXy(X, y)
    X_train, X_test, y_train, y_test = separateTrainTest(X, y)
    t1 = time.clock()
    fitted_rfr = playRandomForestRegressor(_nbtrees, X_train, y_train)
    predict_rfr = predictRandomForestRegressor(fitted_rfr, X_test)
    t2 = time.clock()
    print("temps RandomForestRegressor = ", t2-t1)
    scoreR2 = r2Score(y_test, predict_rfr)
    mae(y_test, predict_rfr)
    if _printGraphe == True:
        importances, indices, std = importanceFeature(fitted_rfr)
        grapheImportanceFeature(X_train, importances, indices, std, col)
    return scoreR2
```

```
def gradientBoostingRegressor(_df, _Xcol, _inexclude, _ycol, _colNoDuplic, _n_estimators, _colLabelEncoder):
    dfOriginelSansVide = getNoDuplication(_df, _colNoDuplic)
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    if len(_colLabelEncoder) > 0:
        dfNum = addLabelEncoder(dfNum, dfOriginelSansVide, _colLabelEncoder)
    X = getX(dfNum, _Xcol, _inexclude)
    col = X.columns
    y = getY(dfNum, _ycol)
    X, y = normaliseXy(X, y)
    X_train, X_test, y_train, y_test = separateTrainTest(X, y)
    params = {'n_estimators': _n_estimators, 'max_depth': 4, 'min_samples_split': 2, 'learning_rate': 0.01, 'loss': 'ls'}
    reg_gb = GradientBoostingRegressor(**params)
    t1 = time.clock()
    reg_gb.fit(X_train, y_train)
    predict_y_gbr = reg_gb.predict(X_test)
    t2 = time.clock()
    print("temps GradientBoostingRegressor = ", t2-t1)
    print("Le score r2 du gradient boosting regressor est de : {0}".format(r2_score(y_test, predict_y_gbr)))
    print "\nFeature Importances"

    importances = reg_gb.feature_importances_

    indices = np.argsort(importances)[::-1]

    for f in range(X.shape[1]):
        print("%d. feature %d %s (%f)" % (f + 1, indices[f], col[indices[f]], importances[indices[f]]))
def regessionLineaire(_df):
    dfOriginelSansVide = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    # variables explicatives
    X = getX(dfNum, ['ARR_DELAY_NEW'], "exclude")
    print("Variables explicatives : {0}".format(X.columns))
    # variable à expliquer
    y = getY(dfNum, ['ARR_DELAY_NEW'])
    # normalisation des données
    X, y = normaliseXy(X, y)
    # séparation des enregistrements d'entraînement et de test
    X_train, X_test, y_train, y_test = separateTrainTest(X, y)
    # instantiation du modèle de régression linéaire
    reg_lr = linear_model.LinearRegression()
    # entraînement du modèle avec les données d'entraînement
    fittedData = reg_lr.fit(X_train, y_train)
    # prédiction à partir des données de test
    predict_y = reg_lr.predict(X_test)
    # score r2 de la prédiction
    print("\nScore r2 de la prédiction : {0}".format(r2_score(y_test, predict_y)))
def grapheCausesRetards(_df):
    dfDelaysSup0 = _df[_df['ARR_DELAY_NEW']>0]
    dfDelaysSup0 = resetIndex(dfDelaysSup0)
    delayReasons = pd.DataFrame(dfDelaysSup0.groupby(['CARRIER'])['CARRIER_DELAY', 'WEATHER_DELAY', 'LATE_AIRCRAFT_DELAY', '
NAS_DELAY', 'SECURITY_DELAY'].mean())
    delayReasons.plot.bar(legend = True,figsize = (13,11),rot=55)
    plt.legend(loc=2,prop={'size':13})
    plt.tick_params(labelsize = 13)
def gridSearchCVGradientBoosting(_df):
    dfOriginelSansVide = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    dfNum = addLabelEncoder(dfNum, dfOriginelSansVide, ['CARRIER', 'ORIGIN', 'DEST', 'TAIL_NUM'])
    X = getX(dfNum, ['ARR_DELAY_NEW'], "exclude")
    col = X.columns
    print(col)
    y = getY(dfNum, ['ARR_DELAY_NEW'])
    X, y = normaliseXy(X, y)
    X_train, X_test, y_train, y_test = separateTrainTest(X, y)
    t1 = time.clock()
    estimator = GradientBoostingRegressor()
    cv = ShuffleSplit(X_train.shape[0], n_iter=10, test_size=0.2)
    param_grid={'n_estimators':[10, 50, 100, 200, 300],
                'learning_rate': [0.1, 0.05, 0.02],# 0.05, 0.02, 0.01],
                'max_depth':[4, 6, 8],#4,6],
                'min_samples_leaf':[3, 5],#,5,9,17],
                'max_features':[1.0, 0.3],#,0.3]#,0.1]
               }
    ref = GridSearchCV(estimator=estimator, cv=cv, param_grid=param_grid, n_jobs=4)
    ref.fit(X_train, y_train)
    t2 = time.clock()
    print("temps GradientBoostingRegressor = ", t2-t1)
    print("best_params_", ref.best_params_)
    print("best_score_ = ",ref.best_score_)
```

```
def gridSearchCVGradientBoosting(_df):
    dfOriginelSansVide = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    dfNum = addLabelEncoder(dfNum, dfOriginelSansVide, ['CARRIER', 'ORIGIN', 'DEST', 'TAIL_NUM'])
    X = getX(dfNum, ['ARR_DELAY_NEW'], "exclude")
    col = X.columns
    print(col)
    y = getY(dfNum, ['ARR_DELAY_NEW'])
    X, y = normaliseXy(X, y)
    X_train, X_test, y_train, y_test = separateTrainTest(X, y)
    t1 = time.clock()
    estimator = RandomForestRegressor()
    cv = ShuffleSplit(X_train.shape[0], n_iter=10, test_size=0.2)
    param_grid={'n_estimators':[10, 50, 100, 200, 300],
                'max_depth':[2, 4, 6],#4,6],
                'min_samples_leaf':[3, 5, 9],#,5,9,17],
                'max_features':[1.0, 0.3],#,0.3]#,0.1]
               }
    reg = GridSearchCV(estimator=estimator, cv=cv, param_grid=param_grid, n_jobs=4)
    reg.fit(X_train, y_train)
    t2 = time.clock()
    print("temps RandomForestRegressor = ", t2-t1)
    print("best_params_", reg.best_params_)
    print("best_score_ = ",reg.best_score_)

def blending(_df):
    dfOriginelSansVide = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    dfNum = addLabelEncoder(dfNum, dfOriginelSansVide, ['CARRIER', 'ORIGIN', 'DEST', 'TAIL_NUM'])
    X = getX(dfNum, ['ARR_DELAY_NEW'], "exclude")
    col = X.columns
    print(col)
    y = getY(dfNum, ['ARR_DELAY_NEW'])
    X_scaler = StandardScaler()
    X_scaled = X_scaler.fit_transform(X)
    y_scaler = StandardScaler()
    y_scaled = y_scaler.fit_transform(y)
    X_train_scaled, X_test_scaled, y_train_scaled, y_test_scaled = separateTrainTest(X_scaled, y_scaled)

    GLOBAL_MEAN = y_train_scaled.mean()

    reg_lr = linear_model.LinearRegression()
    reg_lr.fit(X_train_scaled, y_train_scaled)
    predict_y_lr_scaled = reg_lr.predict(X_test_scaled)
    temp = []
    for i in range(len(predict_y_lr_scaled)):
        temp.append(predict_y_lr_scaled[i][0])
    predict_y_lr_scaled = np.array(temp)
    print('LinearRegression score = {score}'.format(score=r2_score(y_test_scaled, predict_y_lr_scaled)))

    reg_gbr = GradientBoostingRegressor(n_estimators=100)
    reg_gbr.fit(X_train_scaled, y_train_scaled)
    predict_y_gbr_scaled = reg_gbr.predict(X_test_scaled)
    print('GradientBoostingRegressor score = {score}'.format(score=r2_score(y_test_scaled, predict_y_gbr_scaled)))

    reg_rfr = RandomForestRegressor(n_estimators=200)
    reg_rfr.fit(X_train_scaled, y_train_scaled)
    predict_y_rfr_scaled = reg_rfr.predict(X_test_scaled)
    print('RandomForestRegressor score = {score}'.format(score=r2_score(y_test_scaled, predict_y_rfr_scaled)))
    def minimize_train_r2(W):
        final_prediction_scaled  = GLOBAL_MEAN + predict_y_lr_scaled * W[0] + predict_y_gbr_scaled * W[1] + predict_y_rfr_sc
aled * W[2] + W[3]
        # on veut maximiser le score r2, donc minimiser le score -r2
        score = -r2_score(y_test_scaled, final_prediction_scaled)
        return(score)

    res = minimize(minimize_train_r2, [ 0.0,  0.0,  0.0,  0.0], method='Nelder-Mead', tol=1e-5, options={'maxiter': 5000})

    print("Les poids des différents algorithmes sont : {0}".format(res.x))
    print("Le score r2 pour cette pondération est : {0}".format(-res.fun))

    return X, y, X_scaler, y_scaler, reg_rfr, reg_gbr, reg_lr, GLOBAL_MEAN, res

def prediction(_exist, _n, _X, _y, _X_scaler, _y_scaler, _reg_rfr, _reg_gbr, _reg_lr, _GLOBAL_MEAN, _res):
    if _exist:
        X1 = _X.iloc[_n]
        print(X1)
        y1 = _y.iloc[_n]
        print(y1)
    else:
        X1 = _X
    X1_scaled = _X_scaler.transform(X1)
    predict_y_rfr_X1_scaled = _reg_rfr.predict(X1_scaled)
    predict_y_gbr_X1_scaled = _reg_gbr.predict(X1_scaled)
    predict_y_lr_X1_scaled = _reg_lr.predict(X1_scaled)
    pred_y1_scaled = _GLOBAL_MEAN + predict_y_lr_X1_scaled[0][0] * _res.x[0] + predict_y_gbr_X1_scaled[0] * _res.x[1] + pred
ict_y_rfr_X1_scaled[0] * _res.x[2] + _res.x[3]
    pred_y1_scaled_arr = []
    pred_y1_scaled_arr.append(pred_y1_scaled)
    pred_y1 = _y_scaler.inverse_transform(pred_y1_scaled_arr)
    if _exist:
        print("\nValeur réelle = {0}".format(y1))
    print("\nValeur prédite = {0}".format(pred_y1[0]))
```

```
def graphScores(arrAlgo, arrScore):
    dfScores = pd.DataFrame({'Algorithmes': arrAlgo, 'Scores r2': arrScore})
    sns.set_style("darkgrid")
    plt.figure(figsize = (6, 6))
    sns.barplot(x = dfScores['Algorithmes'], y = dfScores['Scores r2'])
    plt.xticks(rotation=90)
    plt.xlabel("Algorithmes")
    plt.ylabel("Scores r2")

def heatmap(_df, _methode):
    """Affichage de la heatmap de la matrice de corrélation des variables continues
    Arguments:
    _df -- dataframe contenant les variables
    _methode -- méthode statistique pour la corrélation, Pearson, Spearman ou Kendall
    """
    dfOriginelSansVide = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansVide = resetIndex(dfOriginelSansVide)
    dfNum = getNum(dfOriginelSansVide)
    # suppression des figures
    plt.clf()
    # on calcule la matrice de corrélation
    corr = dfNum.corr(method=_methode)
    # masque d'affichage de la heatmap
    mask = np.zeros_like(corr, dtype=np.bool)
    mask[np.triu_indices_from(mask)] = True
    f, ax = plt.subplots(figsize=(12,12))
    cmap = sns.diverging_palette(220, 10, as_cmap=True)

    # traçage de la heatmap
    sns.heatmap(corr, cmap=cmap, cbar_kws={"shrink": .5}, mask=mask, annot=True)
    plt.show()

def dfBoxPlot(_df, _listeCol, _title):
    """Dessine le boxplot
    Arguments:
    _df -- dataframe contenant les données initiales
    _title -- titre du boxplot
    """
    dfOriginelSansPredictNull = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansPredictNull = resetIndex(dfOriginelSansPredictNull)
    dfOriginelSansPredictNullCol = dfOriginelSansPredictNull[_listeCol]
    plt.figure(figsize=(12,6))
    colors=dict(boxes='DarkGreen', whiskers='DarkOrange', medians='DarkBlue', caps='Gray')
    boxprops = dict(linewidth=2)
    medianprops = dict(linewidth=2)
    meanpointprops = dict(marker='D', markeredgecolor='black', markerfacecolor='firebrick')
    flierprops = dict(marker='o', linestyle='none', markerfacecolor='green', markersize=6)
    ax = dfOriginelSansPredictNullCol.plot.box(showmeans=True, showfliers=True, flierprops=flierprops, boxprops=boxprops, me
dianprops=medianprops, color=colors, meanprops=meanpointprops)
    ax.set_ylabel('Valeurs des variables')
    ax.set_title(_title)

def pivot(_df):
    dfOriginelSansPredictNull = getNoDuplication(_df, 'ARR_DELAY_NEW')
    dfOriginelSansPredictNull = resetIndex(dfOriginelSansPredictNull)
    dfg = dfOriginelSansPredictNull[['DAY_OF_MONTH', 'CARRIER', 'ARR_DELAY_NEW']]
    dfggrouped = dfg.groupby(['CARRIER', 'DAY_OF_MONTH'], as_index=False).mean()
    pt = dfggrouped.pivot_table(index = 'CARRIER', columns = 'DAY_OF_MONTH', values = 'ARR_DELAY_NEW')
    f, ax = plt.subplots(figsize = (12, 6))
    cmap = sns.cubehelix_palette(start = 1.5, rot = 1.5, as_cmap = True)
    sns.heatmap(pt, xticklabels = 3, cmap = cmap, linewidths = 0.05, ax = ax)
    ax.set_title(u"Moyenne des retards par compagnie et par jour du mois de décembre 2016")
    ax.set_xlabel(u"Jour du mois")
    ax.set_ylabel(u"Compagnie")

# premier dataset (avec leakage)
dfOriginel = pd.read_csv('.spyder/notebooks/projet4/airline_delay_causes_old.csv', sep=',')
# deuxième dataset (sans leakage)
dfOriginelNew = pd.read_csv('.spyder/notebooks/projet4/airline_delay_causes.csv', sep=',')
# dataset des causes de retards
dfDelays = pd.read_csv('.spyder/notebooks/projet4/delays.csv', sep=',')

arrAlgo = []
arrScore = []
```