Projet BigData 2019

Laurene CLADT

Préparation des données

Dans un premier temps, toutes les manipulations de données ont été effectuées sur un échantillon réduit à 1000 lignes. Pour constituer ce fichier, j'ai pris des lignes au hasard dans le fichier "2006" tout en vérifiant qu'elles englobent bien les différents cas demandés.

Les attributs manquants (ou "NA") ont étés remplacé par des 0, comme précisé dans le sujet. Les colonnes qui ont été utilisées pour produire les résultats aux questions posées sont les suivantes :

- Month
- DayOfMonth
- DayOfWeek
- Carrier
- ArrDelay
- DepDelay
- Origin
- Dest
- CarrierDelay
- WeatherDelay
- NASDelay
- SecurityDelay
- LateAircraftDelay
- Cancelled
- Diverted

Les données complémentaires concernant les aéroports et les compagnies ont également été utilisées afin de renvoyer le nom correct des aéroports et compagnies demandées.

Au niveau des choix effectués, j'ai tout d'abord remarqué que certaines données des colonnes "ArrDelay" et "DepDelay" étaient négatives, car elles correspondaient à une avance. J'ai choisis de les prendre en compte plutôt que de les supprimer, car une avance est bénéfique pour l'utilisateur et il est donc pertinent de la prendre en compte dans l'analyse des retards.

Concernant les valeurs "Cancelled" ou "Diverted", j'ai décidé de ne pas prendre en compte la ligne associée lors d'un calcul de retard si l'avion était effectivement annulé ou dérouté. Le cas échéant, son retard aurait été initialisé à 0 alors que l'avion ne serait pas parti, ce qui aurait faussé les calculs. En effet, pour un utilisateur, un avion dérouté ou annulé va souvent entraîner un retard : il serait alors absurde de considérer un avion ainsi annulé comme "à l'heure" pour notre analyse.

Les données ont été chargées une seule fois dans trois dataframes (une pour les données principales, et une pour chaque fichier complémentaire) au début du script. **Ce chargement a duré en moyenne 1min20.**

Travail demandé

Tous les traitements ont été effectués via Spark (Scala). Le Notebook Zeppelin (https://zeppelin.apache.org/) a été utilisé afin de produire les résultats graphiques et de formater les résultats.

Question 1

Quel est le meilleur moment (jour (par exemple le 15), jour dans la semaine (par exemple le mardi), mois (mois de mai) pour partir si on veut minimiser les retards ?

Script Spark

```
val df1 = df
    .withColumn("ArrDelay", col("ArrDelay").cast("double"))
    .filter($"Cancelled" =!= 1 && $"Diverted" =!= 1)
    .na.fill(0,Seq("ArrDelay"))

// On créé également des fonctions pour formatter le jour et le mois
def jourSemaineUDF = udf { (day:Int) =>
    day match {
        case 1 => "Lundi"
        case 2 => "Mardi"
        case 3 => "Mercredi"
        case 4 => "Jeudi"
        case 5 => "Vendredi"
        case 6 => "Samedi"
        case 7 => "Dimanche"
    }
}

def moisUDF = udf { (day:Int) =>
    day match {
        case 1 => "Janvier"
        case 2 => "Février"
        case 3 => "Mars"
        case 4 => "Avril"
        case 5 => "Mai"
        case 6 => "Juin"
        case 7 => "Juillet"
        case 8 => "Août"
        case 9 => "Septembre"
        case 10 => "Octobre"
        case 11 => "Novembre"
        case 12 => "Décembre"
    }
}

// Meilleur jour
val req1 = df1.groupBy("DayOfMonth").agg(avg("ArrDelay").as("AvgDelay"))
println("Meilleur jour du mois :")
req1.sort(asc("AvgDelay")).select("DayOfMonth").limit(1).show

// Meilleur jour de la semaine
val req2 = df1.groupBy("DayOfWeek").agg(avg("ArrDelay").as("AvgDelay")).withColumn("DayOfWeek",jourSemaineUDF(col("DayOfWeek")))
println("Meilleur jour de la semaine :")
req2.sort(asc("AvgDelay")).select("DayOfWeek").limit(1).show

// Meilleur mois
val req3 = df1.groupBy("Month").agg(avg("ArrDelay").as("AvgDelay")).withColumn("Month",moisUDF(col("Month")))
println("Meilleur mois :")
req3.sort(asc("AvgDelay")).select("Month").limit(1).show
```

Afin de répondre à la question, j'ai calculé la moyenne des retards d'arrivée pour chaque jour, jour de la semaine et mois. Uniquement les retards à l'arrivée ont été considérés, car ce sont ceux qui importent à l'utilisateur dans ce cas précis.

Ce traitement (avec affichage des résultats) a duré en moyenne 1min15.

Résultats

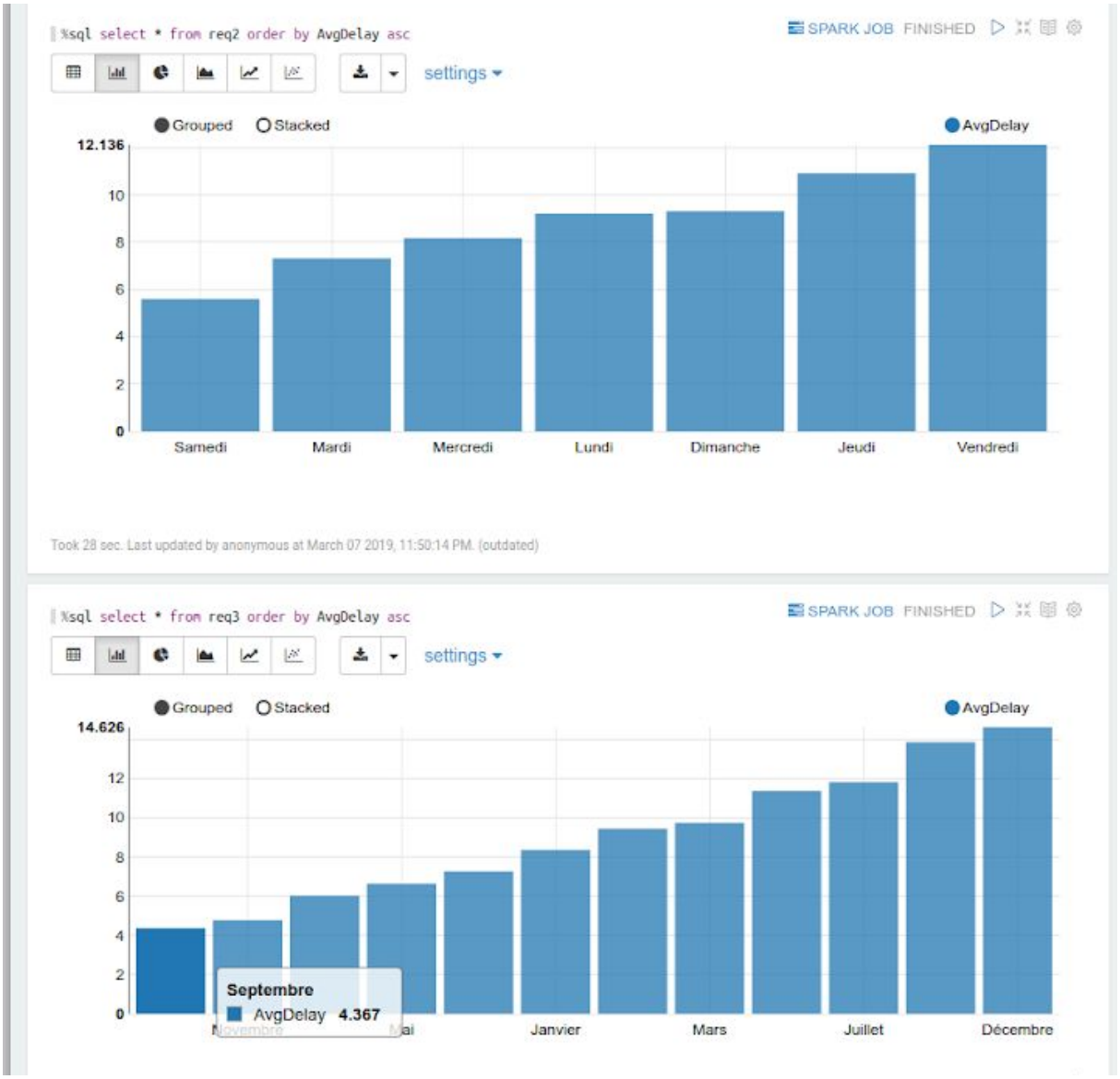
```
Meilleur jour du mois :
+----------+
|DayOfMonth|
+----------+
|         9|
+----------+
```

```
Meilleur jour de la semaine :
+---------+
|DayOfWeek|
+---------+
|   Samedi|
+---------+
```

```
Meilleur mois :
+---------+
|    Month|
+---------+
|Septembre|
+---------+
```

Les meilleurs moments pour minimiser les retards sont donc le 9 du mois, le samedi et en septembre.

Pour ces résultats, le graphique sous forme d'histogramme semble approprié : il nous permet facilement de visualiser quels moments sont les plus avantageux en termes de retard.



L'affichage des graphiques via requête SQL a pris en moyenne 25sec.

Question 2

Quelles sont les causes principales des retards (âge des avions, météo, compagnie...) ?

Script Spark

Pour cette question, deux interprétations m'ont semblées envisageables. La première d'entre elles était de calculer le nombre de fois où la raison du retard appropriée était positive. Ainsi, nous obtenons les causes les plus fréquentes de retard, indépendamment du délais causé.

```
val col2 = Array("CarrierDelay","WeatherDelay","NASDelay","SecurityDelay","LateAircraftDelay")

val df2 = col2.map { name:String => (name, df.filter(name+">0").count)}.toSeq.toDF("Name","CountDelay")
df2.sort(desc("CountDelay")).show
```

```
+-----------------+----------+
|             Name|CountDelay|
+-----------------+----------+
|         NASDelay|   2914154|
|LateAircraftDelay|   2269341|
|     CarrierDelay|   2203893|
|     WeatherDelay|    341577|
|    SecurityDelay|     26144|
+-----------------+----------+
```

Ce script a duré en moyenne 1min30.

La seconde interprétation était de calculer la somme des délais causés pour chaque cause retard. De ce fait, plus une cause particulière a engendré des délais importants, plus elle est valorisée.

```
val df2_1 = col2.map {name:String => (name,df.select(sum(df(name))).first.getDouble(0)) }.toSeq.toDF("Name","CountDelay")
df2_1.sort(desc("CountDelay")).show
```

```
+-----------------+-----------+
|             Name| CountDelay|
+-----------------+-----------+
|LateAircraftDelay|  1.01921E8|
|         NASDelay|7.9982962E7|
|     CarrierDelay|7.7111945E7|
|     WeatherDelay|1.5227209E7|
|    SecurityDelay|   512593.0|
+-----------------+-----------+
```

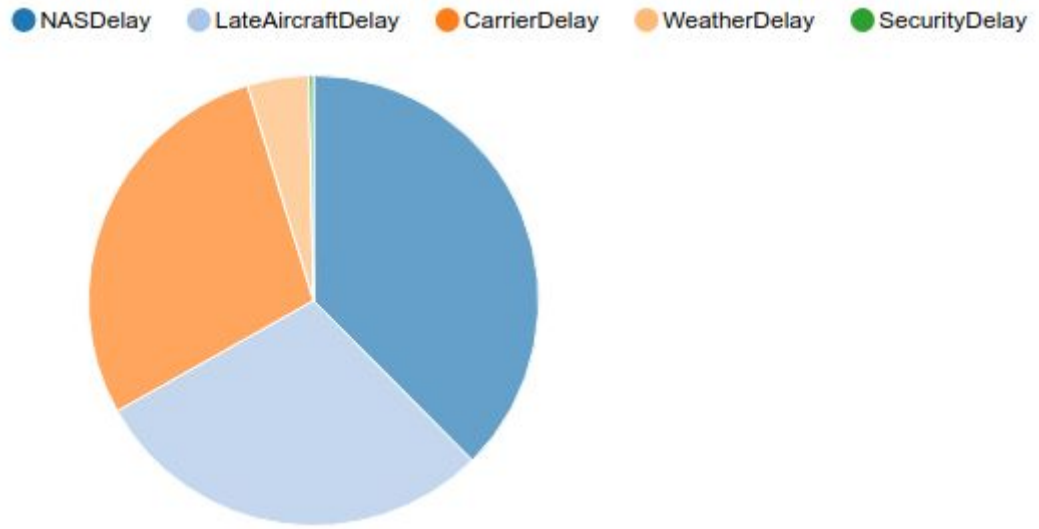
Ce script a duré en moyenne 2min.

On remarque que ces deux interprétations nous donnent des résultats différents quant à la cause principale de délais. **Dans tous les cas, les deux causes principales de retard sont les priorités données aux autres vols et le contrôle NAS.**

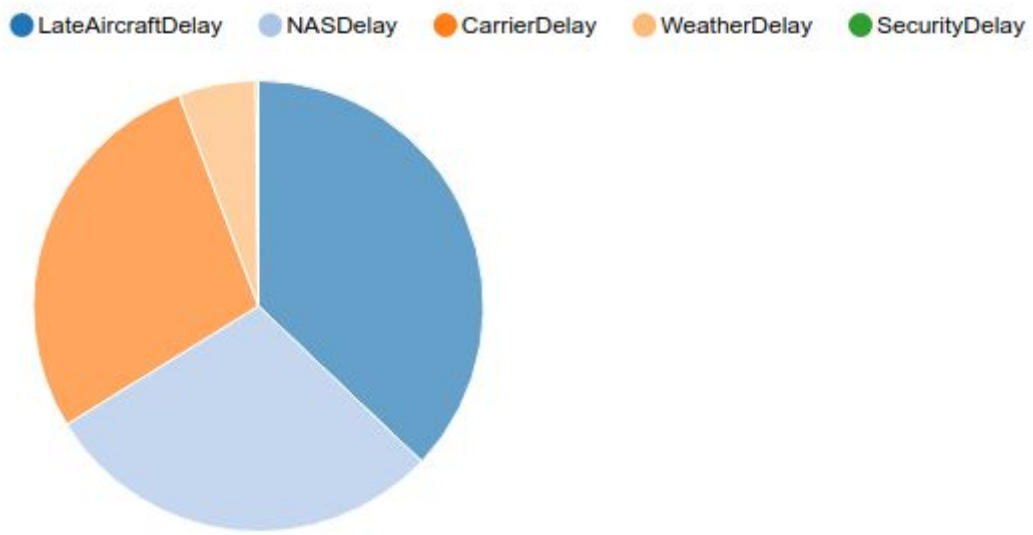
Résultats

On peut également visualiser ces différents résultats : ici, le diagramme en secteurs est le plus approprié.

Nombre d'occurrences



Somme des délais



Note: les couleurs de "LateAircraftDelay" et "NASDelay" sont inversées par rapport au schéma précédent.

Encore une fois, bien que les résultats diffèrent, on constate que les proportions sont similaires. Cependant je préfère la première interprétation, qui correspond plus à ma vision personnelle de la question posée.

Question 3

Quelles sont les compagnies les plus/moins sujettes aux retards (on classifiera les compagnies en 5 groupes) ?

Pour cette question, j'ai choisi d'utiliser l'algorithme des K-Means avec 5 clusters et comme données les moyennes des retards de départ et d'arrivée. Cela permet d'obtenir 5 groupes de données similaires.

Script Spark

```
val df3 = df
    .select(col("UniqueCarrier"),col("ArrDelay").cast("double"),col("DepDelay").cast("double"))
    .filter($"Cancelled" =!= 1 && $"Diverted" =!= 1)
    .na.fill(0,Seq("ArrDelay"))
    .na.fill(0,Seq("DepDelay"))

val df3g = df3.groupBy("UniqueCarrier").agg(avg("ArrDelay").as("ArrDelay"),avg("DepDelay").as("DepDelay"))

// K-means
import org.apache.spark.ml.feature.VectorAssembler
import org.apache.spark.ml.clustering.{KMeans,KMeansModel}
import org.apache.spark.ml.Pipeline

val assembler = new VectorAssembler().setInputCols(Array("ArrDelay","DepDelay")).setOutputCol("features")
val kmeans = new KMeans().setK(5).setFeaturesCol("features").setPredictionCol("prediction")
val pipeline = new Pipeline().setStages(Array(assembler, kmeans))
val model = pipeline.fit(df3g)
val predictions = model.transform(df3g)

// Afficher les coordonées des clusters pour caractériser les différents groupes
model.stages.last.asInstanceOf[KMeansModel].clusterCenters.foreach(println)
```

```
[6.106280676890199,7.027108939759698]
[-0.6122804308610512,-0.2377525587749199]
[11.334688358265883,12.19582286155677]
[7.654144663034837,8.729630494036641]
[14.879534459720709,14.5144340403966]
```

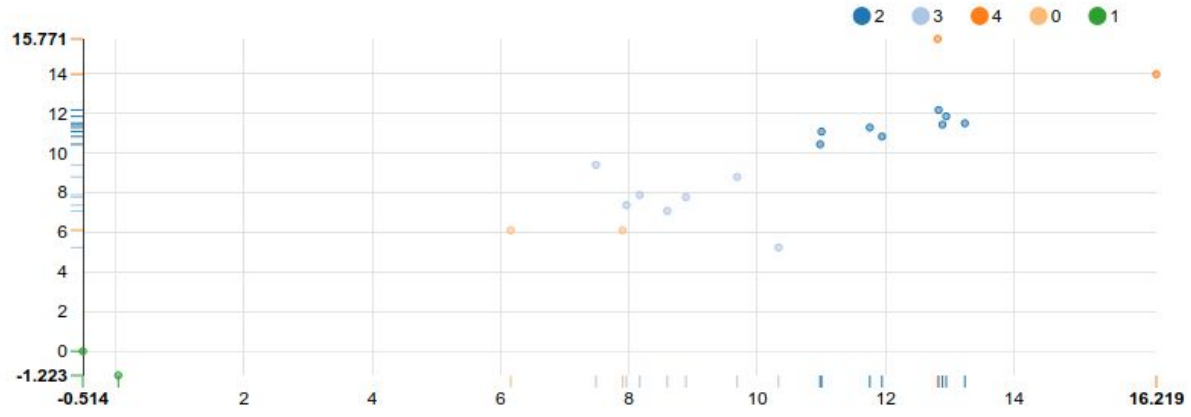
Ce script nous permet de visualiser les centroids des différents clusters ainsi produits. Ainsi, nous pouvons caractériser chaque groupe en fonction des données moyennes de son centre. Grâce à ce résultat, on peut en déduire que le deuxième groupe contient les compagnies les moins sujettes aux retards, tandis que le dernier groupe comprend celles les plus sujettes aux retards. **Ce script a mis en moyenne 1min20 pour s'exécuter.**

J'ai également pensé à prendre en compte le nombre de vols annulés ou déroutés lors du calcul des différents groupes. Cependant, je n'ai pas trouvé cela très pertinent par rapport à la question des retards et ai choisi de garder uniquement les moyennes des retards comme variables. Les groupes ainsi obtenus dépendent uniquement des retards moyens plutôt que du nombre de vols annulés ou déroutés.

```
val df3g = df3.groupBy("UniqueCarrier").agg(
    avg("ArrDelay").as("ArrDelay"),
    avg("DepDelay").as("DepDelay"),
    sum("Cancelled").as("Cancelled"),
    sum("Diverted").as("Diverted"))
```

Résultats

Via Zeppelin, j'ai produit un nuage de points représentant les différentes compagnies, avec leurs délais moyens de départ et d'arrivées comme coordonnées. L'étiquette colorée correspond au groupe (ou cluster) associé à chaque compagnie. Malheureusement, Zeppelin ne permet pas d'ajouter un label aux points. Sinon, on aurait pu ajouter le code de la compagnie en label sur chaque point afin de pouvoir facilement visualiser ses valeurs moyennes de retard ainsi que son groupe.



Ce graphique, ainsi que le résultat précédent pour les centroids des clusters, nous permettent alors d'afficher les groupes des compagnies pertinentes. Par exemple, voici les compagnies les moins (respectivement les plus) sujettes aux retards :

```
// On récupère le nom complet des compagnies via le fichier complémentaire
println("Compagnies les moins sujettes aux retards")
predictions
    .join(df_carriers,predictions("UniqueCarrier")===df_carriers("Code"))
    .select("UniqueCarrier","Description","Prediction")
    .filter($"prediction" === 1) // celon le résultat précédent
    .show(false) // afficher le nom complet (sans limite de caractères)

println("Compagnies les plus sujettes aux retards")
predictions
    .join(df_carriers,predictions("UniqueCarrier")===df_carriers("Code"))
    .select("UniqueCarrier","Description","Prediction")
    .filter($"prediction" === 4) // celon le résultat précédent
    .show(false) // afficher le nom complet (sans limite de caractères)
```

```
Compagnies les moins sujettes aux retards
+-------------+---------------------+----------+
|UniqueCarrier|Description          |Prediction|
+-------------+---------------------+----------+
|AQ           |Aloha Airlines Inc.  |1         |
|HA           |Hawaiian Airlines Inc.|1         |
+-------------+---------------------+----------+

Compagnies les plus sujettes aux retards
+-------------+--------------------------+----------+
|UniqueCarrier|Description               |Prediction|
+-------------+--------------------------+----------+
|EV           |Atlantic Southeast Airlines|4         |
|TZ           |ATA Airlines d/b/a ATA    |4         |
+-------------+--------------------------+----------+
```

Ce traitement a duré en moyenne 1min.

Question 4

Quels sont les 3 aéroports les plus/moins sujets aux retards (départ/arrivée) ?

Script Spark

```
val df4 = df.select(col("Origin"),col("Dest"),col("ArrDelay").cast("double"),col("DepDelay").cast("double"))
    .filter($"Cancelled" =!= 1 && $"Diverted" =!= 1)
    .na.fill(0,Seq("ArrDelay"))
    .na.fill(0,Seq("DepDelay"))

// On récupère les retards au départ et à l'arrivée
val df4o = df4.groupBy("Origin").agg(avg("DepDelay").as("DepDelay"))
val df4d = df4.groupBy("Dest").agg(avg("ArrDelay").as("ArrDelay"))

// Délais total
/*val df4g = df4o
    .join(df4d,df4o("Origin")===df4d("Dest"))
    .groupBy("Origin")
    .agg((avg("DepDelay")+avg("ArrDelay")).as("TotalDelay"))*/

// On utilise le fichier des aéroports pour obtenir le nom des aéroports concernés
println("Aéroports les plus sujets aux retards de départ")
df4o
    .join(df_airports,df4o("Origin")===df_airports("iata"))
    .select("Origin","airport","DepDelay")
    .sort(desc("DepDelay"))
    .limit(3)
    .show(false)

println("Aéroports les moins sujets aux retards de départ")
df4o
    .join(df_airports,df4o("Origin")===df_airports("iata"))
    .select("Origin","airport","DepDelay")
    .sort(asc("DepDelay"))
    .limit(3)
    .show(false)

println("Aéroports les plus sujets aux retards d'arrivée")
df4d
    .join(df_airports,df4d("Dest")===df_airports("iata"))
    .select("Dest","airport","ArrDelay")
    .sort(desc("ArrDelay"))
    .limit(3)
    .show(false)

println("Aéroports les moins sujets aux retards d'arrivée'")
df4d
    .join(df_airports,df4d("Dest")===df_airports("iata"))
    .select("Dest","airport","ArrDelay")
    .sort(asc("ArrDelay"))
    .limit(3)
    .show(false)
```

Pour répondre à cette question, j'ai simplement regroupé les retards de départ avec l'aéroport d'origine et les retards d'arrivée avec l'aéroport de destination. Cela nous permet d'obtenir facilement un classement des aéroports les plus ou moins sujets aux retards. Afin d'obtenir le classement des aéroports tout retard confondu, il aurait suffi de regrouper les valeurs par aéroport et de faire la somme des deux retards moyens. **Ce script a duré en moyenne 1min30.**

Résultats

Aéroports les plus sujets aux retards de départ

Origin	airport	DepDelay
ACK	Nantucket Memorial	37.283018867924525
PIR	Pierre Regional	32.888888888888886
PUB	Pueblo Memorial	27.0

Aéroports les plus sujets aux retards d'arrivée

Dest	airport	ArrDelay
MQT	Marquette County Airport	27.525698516729907
OTH	North Bend Muni	26.79233870967742
ACK	Nantucket Memorial	26.086244541484717

Aéroports les moins sujets aux retards de départ

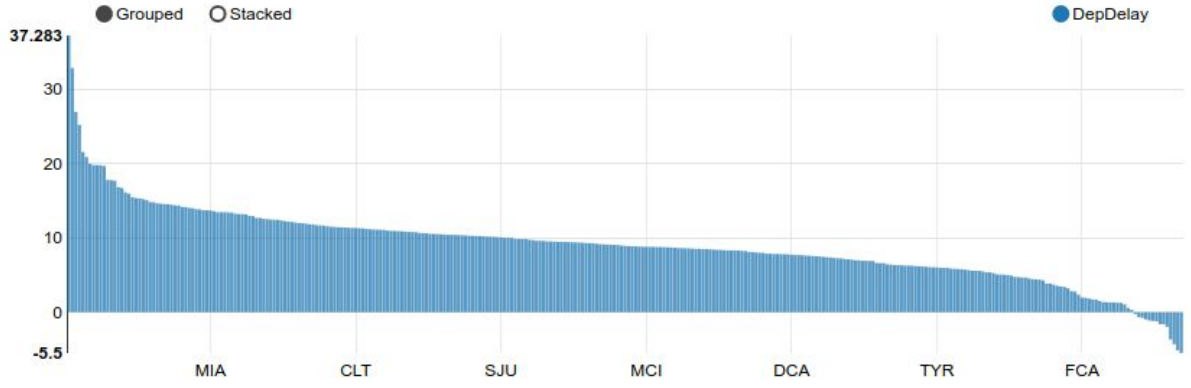
Origin	airport	DepDelay
GLH	Mid Delta Regional	-5.5
WYS	Yellowstone	-5.137229987293519
PIH	Pocatello Regional	-4.306110793832096

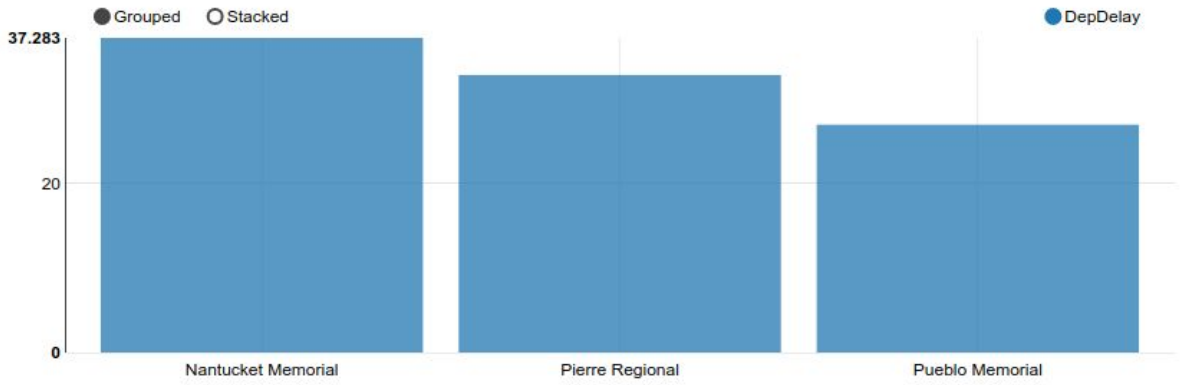
Aéroports les moins sujets aux retards d'arrivée'

Dest	airport	ArrDelay
HVN	Tweed-New Haven	-2.3333333333333335
ITH	Tompkins Cty	-1.7560975609756098
LIH	Lihue	-0.38210751474978255

Grâce au script précédent, on obtient facilement le classement des aéroports. Il est intéressant de noter que les aéroports avec le moins de retards sont les aéroports avec le plus d'avance selon l'interprétation choisie.

Une fois de plus, l'histogramme est le graphique le plus approprié pour visualiser ces données. Il est cependant plus pertinent de limiter les données affichées lors de la visualisation afin d'obtenir un graphique plus facilement navigable au vu du nombre de données représentées.





Histogrammes pour les aéroports sujets aux délais de départ.

Conclusion

J'ai beaucoup apprécié ce projet, qui nous a permis de s'initier à Spark et d'apprendre les concepts basiques tels que la manipulation des données, la classification (ici via l'algorithme K-Means) et le "Top K". On remarque également que des statistiques intéressantes peuvent être produites de la sorte, notamment lorsqu'on utilise des données pratiques tels que des informations sur les aéroports. Pour résumer, ce projet a été une très bonne introduction au monde de la Big Data.

Au niveau de la durée de traitement, chaque partie de script utilisée pour produire des résultats prenait plus ou moins 1 minute à s'exécuter sous Zeppelin. Le temps d'exécution pouvait varier jusqu'à 15 secondes de différence en fonction de l'utilisation de l'ordinateur. J'ai trouvé ce temps correct pour des données de plus de 2Go.

Concernant les questions, j'ai remarqué qu'elles pouvaient être interprétées de différentes manières. J'ai trouvé nécessaire de revenir plusieurs fois sur ce projet afin de cogiter sur mes résultats et la manière dont j'avais choisi de répondre aux questions. Il faut en effet réfléchir à quelles colonnes utiliser, quels calculs effectuer et de quelle manière traiter les données. Par exemple, toutes les questions auraient pu être traitées différemment si j'avais choisi de donner une plus grande importance aux valeurs des colonnes "Cancelled" et "Diverted". Les résultats auraient également été très différents si je n'avais pas normalisé les données en effectuant une moyenne sur les retards et à la place choisi d'effectuer une somme : les aéroports avec le plus de retard auraient alors correspondu aux aéroports avec le plus de départs et arrivées. J'ai tenu à insister sur le choix de l'interprétation de chaque question dans ce rapport, car c'est un point très important lorsqu'on est chargé de produire des rapports sur des données.

Le script Spark complet, le jeu de données et le notebook Zeppelin (au format json) se trouvent également sur mon github à l'adresse https://github.com/claurene/spark-airports.