Projet 4: Anticipez le retard de vol des avions

Réalisé par: Taher Haggui

Problématique :

Définir un modèle de prédiction des retards d'avion afin d'optimiser la logistique des compagnies aériennes .

Comment?

1.Recenser une base de données de milliers de vols.

2.Nettoyage et exploration des données .

3.Modélisation des données .

4.Développement d’un simple site pour mettre en oeuvre le modéle de prédiction.

Résultats attendues?

Un modèle de prédiction de vols

Un site simple met en ouvre le modéle de prédiction.

Importation des données

Contraintes: **Bases de données de large taille.**

```
col_inutiles=['YEAR','FL_DATE','UNIQUE_CARRIER','CARRIER','TAIL_NUM','FL_NUM','ORIGIN_AIRPORT_ID',\
              'ORIGIN_AIRPORT_SEQ_ID','ORIGIN_CITY_MARKET_ID','ORIGIN','ORIGIN_CITY_NAME','ORIGIN_STATE_ABR',\
             'ORIGIN_STATE_FIPS','ORIGIN_STATE_NM','DEST_AIRPORT_ID','DEST_AIRPORT_SEQ_ID','DEST_CITY_MARKET_ID',\
              'DEST','DEST_CITY_NAME','DEST_STATE_ABR','DEST_STATE_FIPS','DEST_STATE_NM','CANCELLATION_CODE',\
              'FLIGHTS','Unnamed: 64']
#list of columns that we judge useless to predict the lateness of flights.
```

```
dt=[] # list of the monthly data
for i in range(1,13):
    if i<10:
        fich='2016_0{}.csv'.format(i)
    else:
        fich='2016_{}.csv'.format(i)

    l=pd.read_csv(fich,sep=',',error_bad_lines=False)
    for c in col_inutiles: # delete preliminarly useless columns.
        del(l[c])
    dt.append(l)
```

Préparation des données : Valeurs manquantes

Missing values

```
tab_missing_values=pd.DataFrame({c:sum([dt[i][c].isna().sum() for i in range(12)]) for c in dt[0].columns},index=['m
#number of missing values for each column.
```

```
tab_missing_values # display the missing values table
```

ONTH	DAY_OF_WEEK	AIRLINE_ID	ORIGIN_WAC	DEST_WAC	CRS_DEP_TIME	DEP_TIME	DEP_DELAY	...	DISTANCE	DISTANCE_GROUP	CARRIER_DELAY
0	0	0	0	0	0	63538	63539	...	5	5	4667538

Variables de Causes de retard:

```
# for the columns CARRIER_DELAY,WEATHER_DELAY,NAS_DELAY,SECURITY_DELAY and LATE_AIRCRAFT_DELAY,
#the missing values mean the cause mentionned in the such column hadn't occur for the flight of the such row.
#so we will replace them by zero.
values={'CARRIER_DELAY':0,'WEATHER_DELAY':0,'NAS_DELAY':0,'SECURITY_DELAY':0,'LATE_AIRCRAFT_DELAY':0}
for i in range(12):
    dt[i]=dt[i].fillna(values)
```

Préparation des données : Valeurs manquantes

Variables de performances de départ/arrivé:

```python
def remp_dep(CRS_DEP_TIME,DEP_TIME,DEP_DELAY,DEP_DELAY_NEW,DEP_DEL15,DEP_DELAY_GROUP,DEP_TIME_BLK):
    """"This function allow to determinate the exaxtly value of an missing value when equivalent values
    are available."""
    if pd.isnull(DEP_TIME):
        if not pd.isnull(DEP_DELAY):
           DEP_TIME= CRS_DEP_TIME + DEP_DELAY
        elif not pd.isnull(DEP_DELAY_NEW):
            if DEP_DELAY_NEW > 0:
                DEP_TIME= CRS_DEP_TIME + DEP_DELAY_NEW
        elif not pd.isnull(DEP_TIME_BLK):
           DEP_TIME= DEP_TIME_BLK[5:]
    if pd.isnull(DEP_DELAY):
        if not pd.isnull(DEP_TIME):
           DEP_DELAY= int(DEP_TIME) - int(CRS_DEP_TIME)
        elif not pd.isnull(DEP_DELAY_NEW):
            if DEP_DELAY_NEW > 0:
                DEP_DELAY= DEP_DELAY_NEW
    if pd.isnull(DEP_DELAY_NEW):
        if not pd.isnull(DEP_DELAY):
            if DEP_DELAY > 0:
                DEP_DELAY_NEW= DEP_DELAY
            else:
                DEP_DELAY_NEW=0
    if pd.isnull(DEP_DEL15):
        if not pd.isnull(DEP_DELAY_NEW):
            DEP_DEL15= ceil(DEP_DELAY_NEW/15)
            if DEP_DEL15 > 1:
                DEP_DEL15=0
        elif not pd.isnull(DEP_DELAY_GROUP):
```

La relation linéaire entre les différentes variables de performance de départ.

Préparation des données : Valeurs manquantes

Taux de remplissage:

```python
# for the rest of columns, we will drop all columns which have missing values more than threshold that we will fix
# to 80%. So we will define hereunder a function which can examinate the percentage of missing values of each
# column and drop all columns which have missing values more than 80%.

def remplissage(df,seuil):
    ''' This function, examinate each column of the introduced dataframe and eliminate each one who has percentage
        of missing values more than the introduced threshold.

        Args:
        dt(DataFrame): a dataframe that the function examinate all his columns.
        seuil(float): threshold upper it the column will be dropped.

        Returns:
        df(DataFrame): a dataframe with columns who contains missing values under the introduced threshold.
    '''
    l=len(df)

    for c in df.columns :
        cnt=1-((df[c].count())/l)
        if cnt > seuil:
            del(df[c])
    return df
```

```python
#drop columns which has more than 80% missing values.
for i in range(12):
    dt[i]=remplissage(dt[i],0.8)
```

Préparation des données : Restes valeurs manquantes

```
tab_missing_values=pd.DataFrame({c:sum([dt[i][c].isna().sum() for i in range(12)]) for c in dt[0].columns},index=['m
```

```
#Display the rest of columns which has missing values yet.
l=[]
for c in tab_missing_values.columns:
    if tab_missing_values.iloc[0][c]> 0:
        l.append(c)
tab_missing_values[l]
```

	TAXI_OUT	WHEELS_OFF	WHEELS_ON	TAXI_IN	CRS_ARR_TIME	ARR_TIME	ARR_DELAY	ARR_DELAY_NEW	ARR_DEL15	ARR_DELAY_GROUP
missing values	65523	65523	67983	67983	3	3	3	3	3	3

```
# As it shown in the table above , we notice that the percentage of rows who contains missing value, represent
# barely 2% of the whole data. So we decide to drop all rows which contains missing values.
for i in range(12):
    dt[i]=dt[i].dropna()
```

Préparation de données : opérations diverses

Rows duplicated

```
for i in range(12):
    dt[i]=dt[i].drop_duplicates()
```

Useless columns

```
# hereunder, we will define a list of columns that we find useless for our goal, which is to predict lateness.
useless_col=['DEP_TIME','DEP_DELAY','DEP_DELAY_NEW','DEP_DEL15','DEP_DELAY_GROUP','DEP_TIME_BLK','TAXI_OUT',\
             'WHEELS_OFF','WHEELS_ON','TAXI_IN','ARR_TIME','ARR_DELAY','ARR_DELAY_NEW','ARR_DEL15','ARR_DELAY_GROUP',
             'ARR_TIME_BLK','CANCELLED','ACTUAL_ELAPSED_TIME','AIR_TIME','DISTANCE_GROUP','DIVERTED']
for i in range(12):
    for c in useless_col:
        del (dt[i][c])
```

Data concatenate

```
data=pd.concat(dt) # concatenate the datas of each month datas in one unique dataframe.
```

Préparation de données : Catégorielle variables

Categorical variables

```
encoderAC=LabelEncoder()
encoderAO=LabelEncoder()
encoderAA=LabelEncoder()
data['AIRLINE_ID']=encoderAC.fit_transform(data['AIRLINE_ID'])
data['ORIGIN_WAC']=encoderAO.fit_transform(data['ORIGIN_WAC'])
data['DEST_WAC']=encoderAA.fit_transform(data['DEST_WAC'])
```

Préparation de données : Features engineering

Features engenering

We will create new variables, namely LATENESS and LATENESS_TIME. LATENESS take 1 if the flight make lateness and 0 if it is not. LATENESS_TIME give time delay of the flight.

```
# The value of LATENESS_TIME take the sum of values of differents causes of delay.
data["LATENESS_TIME"]=[sum([c[i] for i in range(5)]) for c in zip(data['CARRIER_DELAY'],data['WEATHER_DELAY'],\
                        data['NAS_DELAY'],data['SECURITY_DELAY'],data['LATE_AIRCRAFT_DELAY'])]
```

```
#The feature LATENESS take 1 if the value of the feature LATENESS_TIME is strictly positive and 0 if it is not.
data['LATENESS']=[1 if c>0 else 0 for c in data['LATENESS_TIME']]
```

```
#We can drop now columns of cause of delay.
del(data['CARRIER_DELAY'])
del(data['WEATHER_DELAY'])
del(data['NAS_DELAY'])
del(data['SECURITY_DELAY'])
del(data['LATE_AIRCRAFT_DELAY'])
```

Exploration de données: Analyse univariée

Analyse univariate

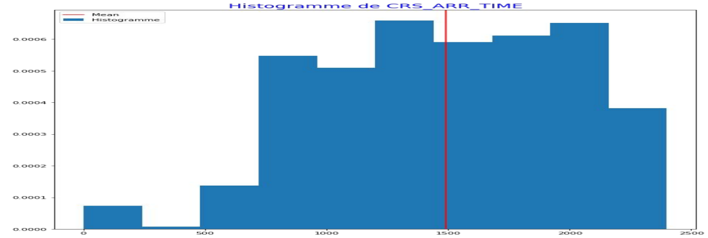
```
tab=pd.DataFrame({c:[data[c].median(),data[c].mean(),data[c].mode(),data[c].quantile(0.25),\
                     data[c].quantile(0.75)] for c in ['QUARTER','MONTH','DAY_OF_MONTH','DAY_OF_WEEK','LATENESS']},\
                index=['median','mean','mode','Q1','Q3'])
```

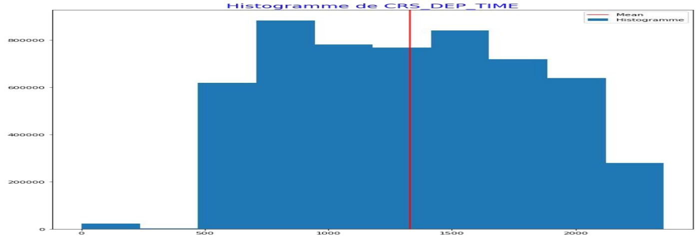
```
tab
```

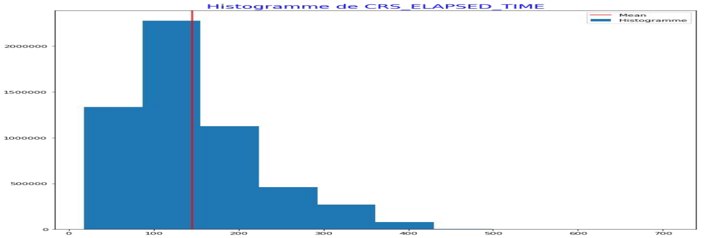
	QUARTER	MONTH	DAY_OF_MONTH	DAY_OF_WEEK	LATENESS
median	3	7	16	4	0
mean	2.49175	6.51562	15.8206	3.92816	0.174299
mode	0 1 dtype: int64	0 3 dtype: int64	0 26.0 dtype: float64	0 5 dtype: int64	0 0 dtype: int64
Q1	1	3	8	2	0
Q3	3	9	23	6	0

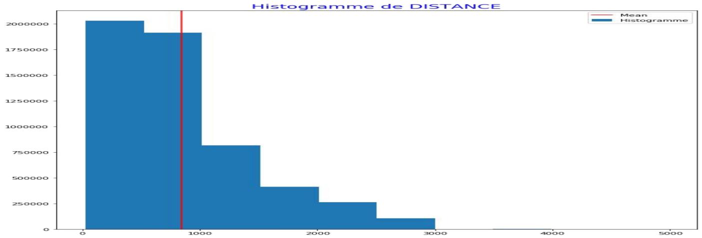
-The most flight of the year are in the first quarter especially in the third month.
-The most flight of the week are in saturday.
-More than 80% of flight are in time.

Exploration de données: Analyse univariée









Analyse multivariée :variables qualitatives

Répondre aux questions suivantes:

- Est ce que la fréquence de retards dépend de trimestre de l’année ? (Corrélation entre Quarter et LATENESS)
- Est ce que la fréquence de retards dépend du mois de l’année?(Corrélation entre Month et LATENESS)
- Est ce que la fréquence de retards dépend du jour de mois ? (Corrélation entre DAY_OF_MONTH et LATENESS)
- Est ce que la fréquence de retards dépend du jour de la semaine ?(Corrélation entre DAY_OF_WEEK et LATENESS)
- Est ce que la fréquence de retards dépend de la compagnie aérienne ?(Corrélation entre AIRLINE_ID et LATENESS)
- Est ce que la fréquence de retards dépend de l'aéroport d’origine ? (Corrélation entre ORIGIN_WAC et LATENESS)
- Est ce que la fréquence de retards dépend de l’aéroport d’arrivée ?(Corrélation entre DEST_WAC et LATENESS)

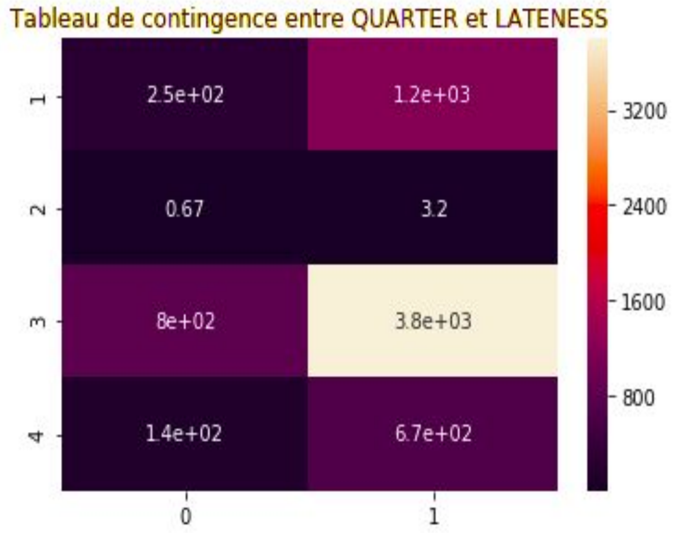
Analyse multivariée :variables qualitatives

```
def corr_var(var1,var2):
    '''This function, consist to mesure the correlation between the introduced variables and to give the contingency
       table between them.

       Args:

       var1(Str): the name of the first variable
       var2(str): the name of the second variable
    '''
    X=var1
    Y=var2
    cnt=data[[X,Y]].pivot_table(index=X,columns=Y,aggfunc=len)
    tx=data[X].value_counts()
    ty=data[Y].value_counts()
    tx=pd.DataFrame(tx)
    ty=pd.DataFrame(ty)
    tx.columns=["values"]
    ty.columns=['values']
    n=len(data)
    ind=tx.dot(ty.T)/n
    mesure=(ind-cnt)**2/ind
    xin=mesure.sum().sum()
    print("Mesure of correlation between the variable {} and the variable {} equal to : {}".format(var1,var2,xin))
    sns.heatmap(mesure,annot=True)
    plt.title('Tableau de contingence entre {} et {}'.format(var1,var2),color='b')
    plt.show()
```

Analyse multivariée :variables qualitatives

```
# Correlation between the variable QUARTER and the variable LATENESS.
corr_var('QUARTER','LATENESS')
```

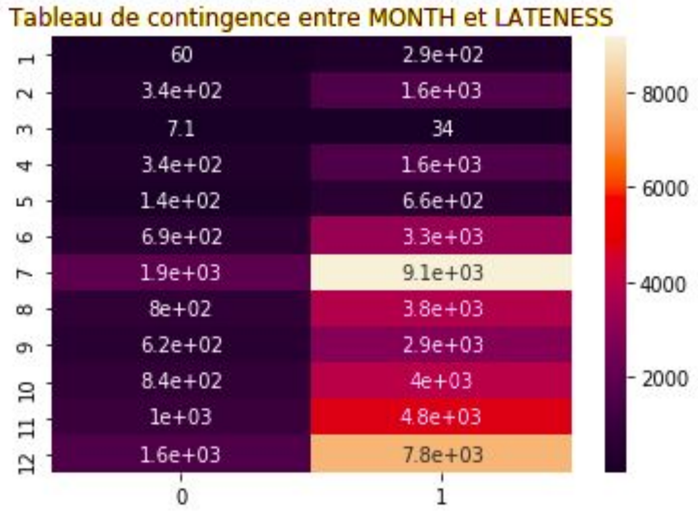
```
Mesure of correlation between the variable QUARTER and the variable LATENESS equal to : 6865.514592155778
```



Analyse multivariée :variables qualitatives

```python
#Correlation between the variable MONTH and the variable LATENESS.
corr_var('MONTH','LATENESS')
```

```
Mesure of correlation between the variable MONTH and the variable LATENESS equal to : 48326.1644588004
```

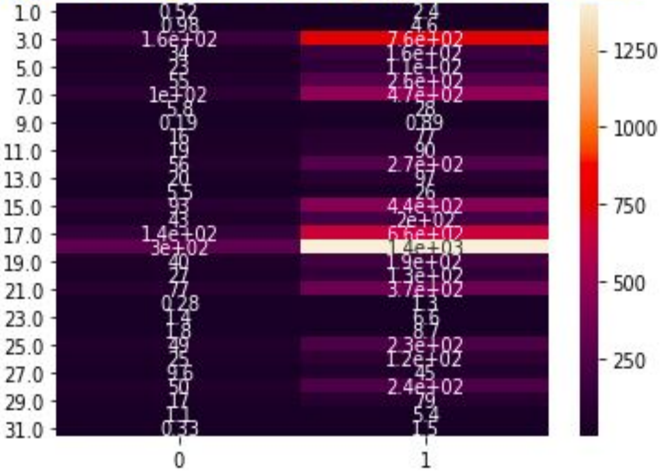


Analyse multivariée :variables qualitatives

```
#Correlation between the variable DAY_OF_MONTH and the variable LATENESS.
corr_var('DAY_OF_MONTH','LATENESS')
```

```
Mesure of correlation between the variable DAY_OF_MONTH and the variable LATENESS equal to : 7853.19455921787 5
```

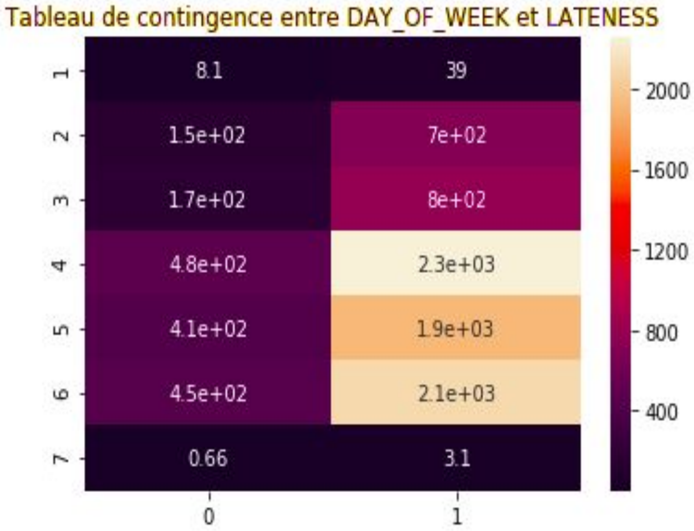
Tableau de contingence entre DAY_OF_MONTH et LATENESS



Analyse multivariée :variables qualitatives

```
#Correlation between the variable DAY_OF_WEEK and the variable LATENESS.
corr_var('DAY_OF_WEEK','LATENESS')
```

```
Mesure of correlation between the variable DAY_OF_WEEK and the variable LATENESS equal to : 9495.14832236862
```

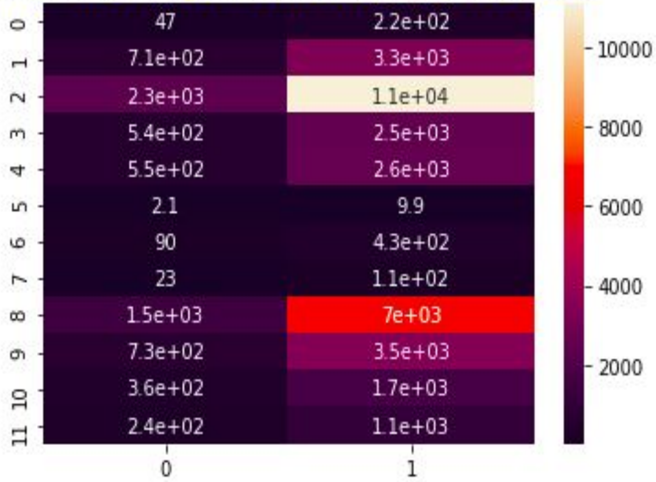


Analyse multivariée :variables qualitatives

```
# Correlation between the variable AIRLINE_ID and the variable LATENESS.
corr_var('AIRLINE_ID','LATENESS')
```

```
Mesure of correlation between the variable AIRLINE_ID and the variable LATENESS equal to : 40755.24168414295
```

Tableau de contingence entre AIRLINE_ID et LATENESS

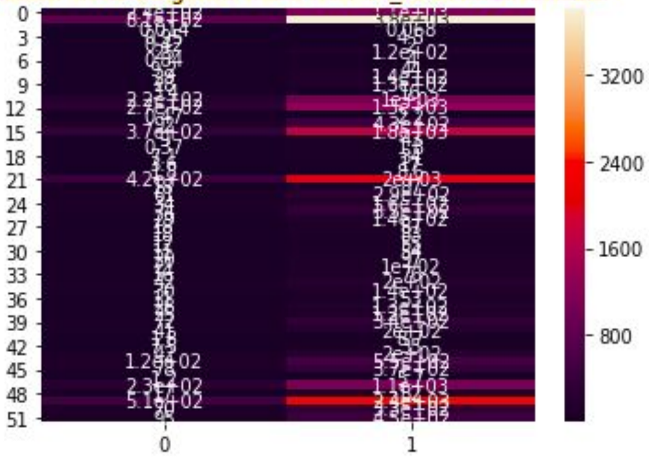


Analyse multivariée :variables qualitatives

```
#Correlation between the variable ORIGIN_WAC and the variable LATENESS.
corr_var('ORIGIN_WAC','LATENESS')
```

```
Mesure of correlation between the variable ORIGIN_WAC and the variable LATENESS equal to : 24638.035572611472
```

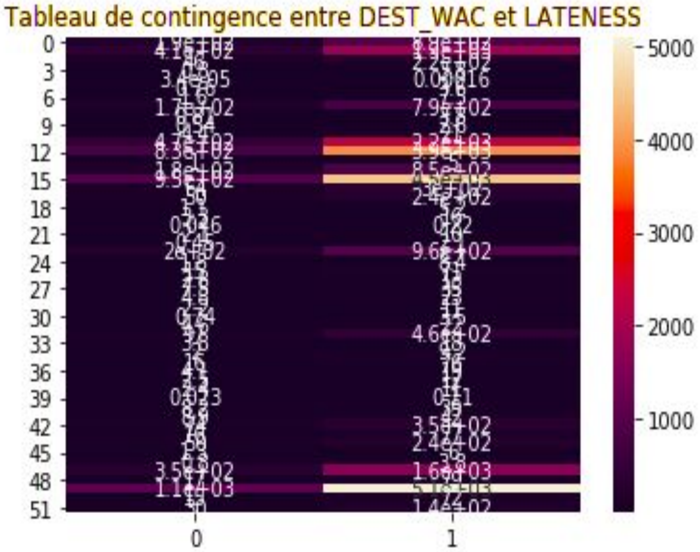
Tableau de contingence entre ORIGIN_WAC et LATENESS



Analyse multivariée :variables qualitatives

```
#Correlation between the variable DEST_WAC and the variable LATENESS.
corr_var('DEST_WAC','LATENESS')
```

```
Mesure of correlation between the variable DEST_WAC and the variable LATENESS equal to : 30841.83601403082
```



Analyse multivariée :variables quantitatives

Quatitatives variables

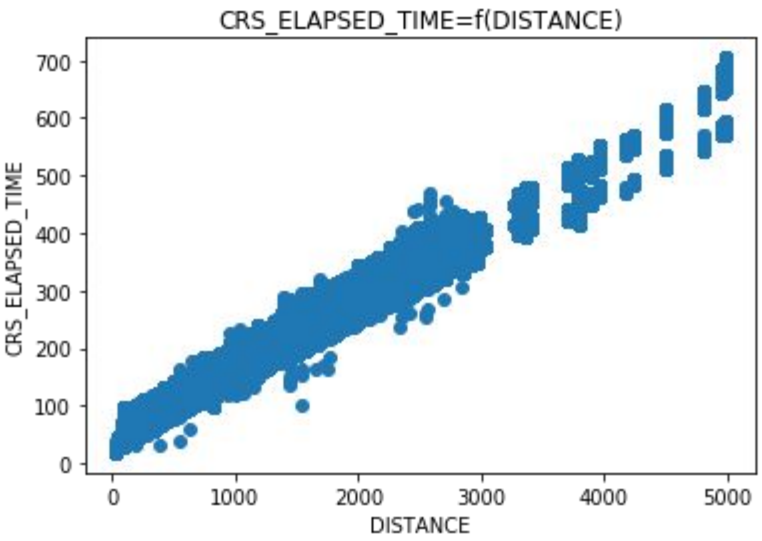
Hereunder, we will try to check the correlation between each pair of quantitative variables.

```
table_corr=data[['LATENESS_TIME','CRS_DEP_TIME','CRS_ARR_TIME','CRS_ELAPSED_TIME','DISTANCE']].corr()
table_corr
```

	LATENESS_TIME	CRS_DEP_TIME	CRS_ARR_TIME	CRS_ELAPSED_TIME	DISTANCE
LATENESS_TIME	1.000000	0.087546	0.075673	0.012693	0.008596
CRS_DEP_TIME	0.087546	1.000000	0.673809	-0.016601	-0.010763
CRS_ARR_TIME	0.075673	0.673809	1.000000	0.024086	0.017993
CRS_ELAPSED_TIME	0.012693	-0.016601	0.024086	1.000000	0.984570
DISTANCE	0.008596	-0.010763	0.017993	0.984570	1.000000

The above table of correlation, give us the following conclusions:
-There is barely no correlation between LATENESS_TIME and seperatly the rest of quantitatives variables.
-As it is waited, a perfect linear correlation between the variable DISTANCE and CRS_ELAPSED_TIME.
-As it is waited, a highly correlation between the variable CRS_DEP_TIME and the variable CRS_ARR_TIME.

Analyse multivariée :variables quantitatives




```
#Linear model links the variable DISTANCE to the variable CRS_ELAPSED_TIME
X=pd.DataFrame({'cte':list(np.ones(len(data))),'d':list(data['DISTANCE'].values)})
Y=data['CRS_ELAPSED_TIME']
Xtr,Xts,Ytr,Yts=train_test_split(X,Y,train_size=0.7)
regressor=LinearRegression(fit_intercept=False)
regressor.fit(Xtr,Ytr)
```

```
/home/taher/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2026: Fut
on 0.21, test_size will always complement train_size unless both are specified.
  FutureWarning)
```

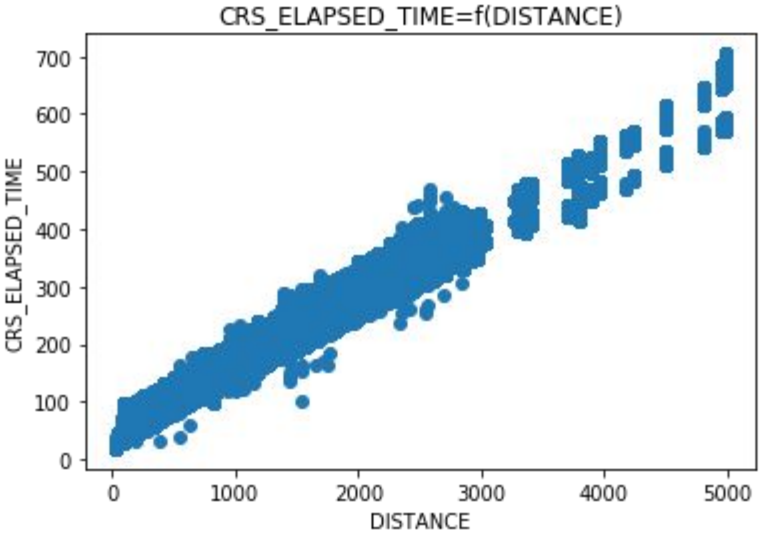
```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
```

```
#Compute the robustness of found model
ypred=regressor.predict(Xts)
print('The robustness of the found model is :{}'.format(sqrt(mean_squared_error(Yts,ypred))))
```

```
The robustness of the found model is :13.359753549811428
```

CRS_ELAPSED_TIME=42.13+0.12*DISTANCE

Analyse multivariée :variables quantitatives




```
#Linear model links the variable DISTANCE to the variable CRS_ELAPSED_TIME
X=pd.DataFrame({'cte':list(np.ones(len(data))),'d':list(data['DISTANCE'].values)})
Y=data['CRS_ELAPSED_TIME']
Xtr,Xts,Ytr,Yts=train_test_split(X,Y,train_size=0.7)
regressor=LinearRegression(fit_intercept=False)
regressor.fit(Xtr,Ytr)
```

```
/home/taher/anaconda3/lib/python3.6/site-packages/sklearn/model_selection/_split.py:2026: Fut
on 0.21, test_size will always complement train_size unless both are specified.
  FutureWarning)
```

```
LinearRegression(copy_X=True, fit_intercept=False, n_jobs=1, normalize=False)
```

```
#Compute the robustness of found model
ypred=regressor.predict(Xts)
print('The robustness of the found model is :{}'.format(sqrt(mean_squared_error(Yts,ypred))))
```

```
The robustness of the found model is :13.359753549811428
```

CRS_ELAPSED_TIME=42.13+0.12*DISTANCE

Prédiction de retard d'avion: tester différents modèles

KNeighbors Model

```
lr=KNeighborsClassifier()
params={'n_neighbors':[3,5,7,9,11,13,15]}
gs_kn=GridSearchCV(lr,params,cv=5)
gs_kn.fit(Xtr,Ytr)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': [3, 5, 7, 9, 11, 13, 15]},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
#The best hyperparametrs of the Knn model
print(gs_kn.best_params_)
```

```
{'n_neighbors': 15}
```

Prédiction de retard d'avion: tester différents modèles

Logistic Model

```
lc=LogisticRegression()
params={'C':np.logspace(-3,3,7),'penalty':['l1','l2']}
gs_lc=GridSearchCV(lc,params,cv=5)
gs_lc.fit(Xtr,Ytr)
```

```
GridSearchCV(cv=5, error_score='raise',
       estimator=LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03]), 'penalty': ['l1', 'l2']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
#The best hyperparametrs of the Logistic model.
print(gs_lc.best_params_)
```

```
{'C': 0.01, 'penalty': 'l1'}
```

Prédiction de retard d'avion: tester différents modèles

SVM Linear Model

```
lsvm=LinearSVC()
params={'C':np.logspace(-3,3,10)}
gs_svc=GridSearchCV(lsvm,params,cv=5)
gs_svc.fit(Xtr,Ytr)
```

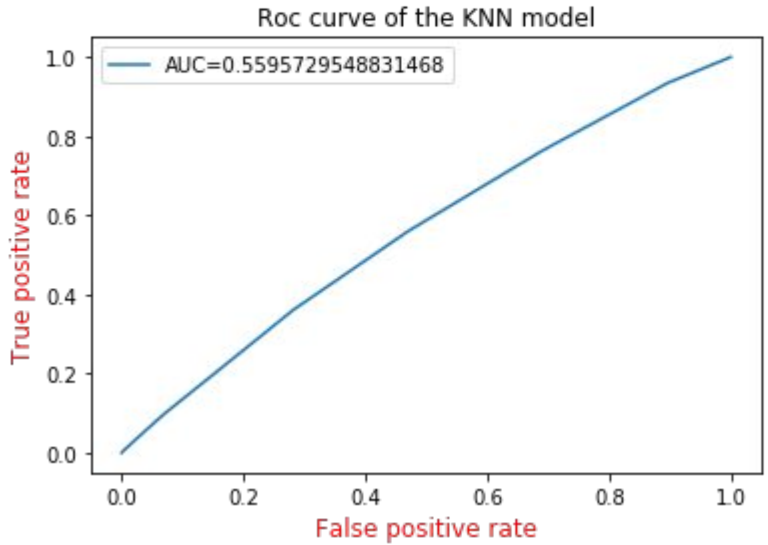
```
GridSearchCV(cv=5, error_score='raise',
       estimator=LinearSVC(C=1.0, class_weight=None, dual=True, fit_intercept=True,
     intercept_scaling=1, loss='squared_hinge', max_iter=1000,
     multi_class='ovr', penalty='l2', random_state=None, tol=0.0001,
     verbose=0),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'C': array([1.00000e-03, 4.64159e-03, 2.15443e-02, 1.00000e-01, 4.64159e-01,
       2.15443e+00, 1.00000e+01, 4.64159e+01, 2.15443e+02, 1.00000e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

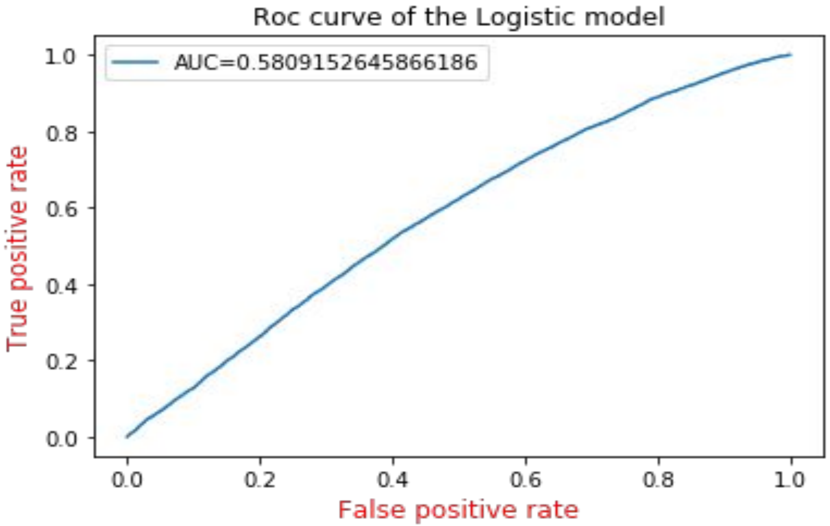
```
#The best hyperparametr of the SVM linear model.
print(gs_svc.best_params_)
```

```
{'C': 0.021544346900318832}
```

Prédiction du temps de retards: évaluation des performances

Aire de la courbe de Roc:





Prédiction du temps de retards: évaluation des performances

Coefficient de précision:

```
#estimate the performance of the knn model.
ypro=gs_kn.predict_proba(Xts)[:,1]
ypre=gs_kn.predict(Xts)
false_pos,tr_pos,thresh=roc_curve(Yts,ypro)
roc_auc_knn=auc(false_pos,tr_pos)
print('roc_auc={}'.format(roc_auc_knn)),
print('Accuracy={}'.format(accuracy_score(Yts,ypre)))
```

```
roc_auc=0.5595729548831468
Accuracy=0.81722
```

```
#estimate the performance of the logistic model.
ypro=gs_lc.predict_proba(Xts)[:,1]
ypr=gs_lc.predict(Xts)
false_pos,tr_pos,thresh=roc_curve(Yts,ypro)
roc_auc_logistic=auc(false_pos,tr_pos)
print('roc_auc={}'.format(roc_auc_logistic)),
print('Accuracy={}'.format(accuracy_score(Yts,ypr)))
```

```
roc_auc=0.5809152645866186
Accuracy=0.8217
```

```
#Estimate the performance of the SVM linear model.
ypre=gs_svc.predict(Xts)

print('Accuracy={}'.format(accuracy_score(Yts,ypre)))
```

```
Accuracy=0.82168
```

Prédiction de retard d'avion: Choix du modèle

	KNN	Logistic	SVM
roc_auc	0.61	0.61	-
Accuracy	0.82	0.82	0.82

Le modèle logistique , vu son avantage d'être explicite

	features	origin_coef
11	intercpt	-3.197662
0	QUARTER	-0.151833
1	MONTH	0.064476
2	DAY_OF_MONTH	0.001588
3	DAY_OF_WEEK	0.001825
4	AIRLINE_ID	0.036058
5	ORIGIN_WAC	0.004055
6	DEST_WAC	-0.003477
7	CRS_DEP_TIME	0.000648
8	CRS_ARR_TIME	0.000215
9	CRS_ELAPSED_TIME	0.002444
10	DISTANCE	-0.000223

$$P(LATENESS = 1) = \frac{1}{1+\exp(eq\_lin)}$$

Prédiction du temps de retards: tester différents modèles

Linear regression model

```
lr=LinearRegression()
lr.fit(Xtr,Ytr)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=False)
```

```
yprd=lr.predict(Xts)
```

```
lr_squared_error=sqrt(mean_squared_error(Yts,yprd)) # compute mean squared error.
```

Prédiction du temps de retards: tester différents modèles

Ridge Model

```
lR=Ridge()
params={'alpha':np.logspace(-3,3,7)}
gs_lR=GridSearchCV(lR,params,cv=10)
gs_lR.fit(Xtr,Ytr)
```

```
GridSearchCV(cv=10, error_score='raise',
       estimator=Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
#The best hyperparametr of the Ridge Model.
print(gs_lR.best_params_)
```

```
{'alpha': 1000.0}
```

```
yprd=gs_lR.predict(Xts)
```

```
lR_squared_error=sqrt(mean_squared_error(Yts,ypr))#compute mean squared error.
```

Prédiction du temps de retards: tester différents modèles

Lasso Model

```
ll=Lasso()
params={'alpha':np.logspace(-3,3,7)}
gs_ll=GridSearchCV(ll,params,cv=10)
gs_ll.fit(Xtr,Ytr)
```

```
GridSearchCV(cv=10, error_score='raise',
       estimator=Lasso(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': array([1.e-03, 1.e-02, 1.e-01, 1.e+00, 1.e+01, 1.e+02, 1.e+03])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

```
#The best hyperparametr of the lasso model.
print(gs_ll.best_params_)
```

```
{'alpha': 0.001}
```

```
ypr=gs_ll.predict(Xts)
```

```
ll_squared_error=sqrt(mean_squared_error(Yts,ypr)) # compute mean squared error.
```

Prédiction du temps de retards: choix du modèle

	baseline regression model	Ridge model	Lasso model
squared_error	38.513333	40.193911	38.513308
C	0.000000	100.000000	0.001000

```
#display the coefficient to respectives features for the baseline model
coef_base_model=pd.DataFrame({'Features':data.columns[:11],'coef':coeficients})
coef_base_model=pd.concat([coef_base_model,pd.DataFrame({'Features':'intercept','coef':constant},index=[11])])
coef_base_model
```

	Features	coef
0	QUARTER	-0.598694
1	MONTH	0.325369
2	DAY_OF_MONTH	0.016985
3	DAY_OF_WEEK	0.155295
4	AIRLINE_ID	0.557054
5	ORIGIN_WAC	0.078322
6	DEST_WAC	-0.092506
7	CRS_DEP_TIME	0.006094
8	CRS_ARR_TIME	0.001365
9	CRS_ELAPSED_TIME	0.108674
10	DISTANCE	-0.012766
11	intercept	-7.624608

Site simple :Code flask

```
from flask import Flask,request,render_template
from math import exp
import numpy as np
app= Flask(__name__)
@app.route('/')
def index():
    quarter=request.args.get('quarter')
    month=request.args.get('month')
    day_of_month=request.args.get('day_month')
    day_of_week=request.args.get('day_week')
    airlineid=request.args.get('airline_id')
    aeroport_or=request.args.get('code_aereport_origine')
    aeroport_ar=request.args.get('code_aereport_arrive')
    heure_depart=request.args.get('heure_depart')
    heure_arrive=request.args.get('heure_arrive')
    elapsed_time=request.args.get('elapsed_time')
    distance=request.args.get('distance')
    prediction='RAS'
    temps_retard="0 s"
    if quarter is not None and month is not None and day_of_month is not None and airlineid is not None and aeroport_or is not None and\
            aeroport_ar is not None and heure_depart is not None and heure_arrive is not None and elapsed_time is not None and\
            distance is not None:

           val=[int(quarter),int(month),int(day_of_month),int(day_of_week),int(airlineid),int(aeroport_or),int(aeroport_ar),\
                   int(heure_depart),int(heure_arrive),int(elapsed_time),int(distance)]
           val=np.array(val)
           val=val.reshape(1,-1)

           prediction_retard=-3.197-0.151*val[0,0]+0.064*val[0,1]+0.0015*val[0,2]+0.0018*val[0,3]+0.036*val[0,4]+0.004*val[0,5]\
                   -0.003*val[0,6]+0.0006*val[0,7]+0.0002*val[0,8]+0.002*val[0,9]-0.0002*val[0,10]
           prediction_retard=1/(1+exp(prediction_retard))


           temps_retard=-7.62-0.6*val[0,0]+0.32*val[0,1]+0.017*val[0,2]+0.155*val[0,3]+0.55*val[0,4]+0.07*val[0,5]-0.09*val[0,6]+\
                   0.006*val[0,7]+0.0013*val[0,8]+0.108*val[0,9]-0.012*val[0,10]
           if prediction_retard > 0.9:
              prediction='OUI'
                                                                                                              20,91         Haut
```

Site simple : Code html

```
<!DOCTYPE html>
<html>
   <head>
      <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1,shrink-to-fit=no">
      <title>Anticipez le retard de vol des avions</title>
   </head>
   <body style="background-color:teal">
      <h1 style="color:fuchsia"> Anticipez le retard de vol des avions </h1>
      <p size=8> Ce site permet de prédir si un vol va étre en retard.Il permet aussi d'estimer le temps de retard dans le cas ou celui-ci  n'
arriverra pas à temps. Pour plus de détail consulter : <a href="https://openclassrooms.com/fr/projects/109/assignment">ce site</a></p>
      <table>
              <tr>
                      <td>
                              <img src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRXpO7a0HFU4MjEDwz3dBAA4JxJFowkAuwwgEgXicwPUohxGB
rY"width=400 height=500 >
                      </td>
                      <td>
                           <table>
                            <tr>
                             <th>  <h1 style="color:white"> Donnés de vol: </h1></th>
                             <th> <h1 style="color:white"> Anticipations:</h1>
                            </tr>
                            <tr>
                             <td>
                              <table cellspacing="0" style="color:silver">
                                    <tr >
                                            <th align=left> Quarter:</th>
                                            <td align=center size=4><b> {{quarter}}</b> </td>
                                    </tr>
                                    <tr>
                                            <th align=left> Month: </th>
                                            <td align=center size=4><b> {{month}} </b> </td>
                                    </tr>
                                    <tr>
                                            <th align=left> Day of month: </th>
                                            <td align=center size=4><b>{{day_month}} </b></td>
                                                                                1,15          Haut
```

site : lien

https://prediction-retard-avions.herokuapp.com/

Conclusion:

- A l’aide des variables temporelles d’une année entière de plusieurs compagnies aériennes, nous avons pu définir des modèles de prédiction de retard de vol.

- A l’aide de heroku, le framework flask . Nous avons pu développer un site permettant de met en oeuvre le modèle choisie.