1. Introduction/Motivation

The aim of this project is to use the data set provided by the U.S. Department of Transportation (DOT) to analyze factors and predict whether the flight will be on time or delayed. The ultimate goal of the project is to help people travel more efficiently and know the relationship between flights, weather, and airports. We have all had flight delays and understanding the relationships between flight delays and factors like airport traffic flow can help people travel more efficiently through strategies such as flying at optimal times.

In this project we used all U.S. flights in 2018, totaling seven million flight records, and recorded 28 different variables. A basic exploration of the data shows that nearly 20% of flights are delayed. The reason may be due to the weather conditions at the place of departure or destination, air traffic at the arrival or destination, or diversions due to instructions from landing towers. For airport flow data, we can download it directly through the DOT website, but we do not have ready-made weather data, so we decided to use web scraping.

2. Data Processing

Web Scraping: In order to acquire the weather data for the flights, we decided to utilize web scraping. This methodology was determined by the fact that many airports have their own weather stations, but common weather APIs sometimes do not service these stations. We managed to find a website (www.wunderground.com), which had data on airport weather stations. This website also had a nice feature where we could edit the url string easily to move from city to city and day to day.

For the actual scraping process, we utilized a python web browser automation tool called Selenium combined with Chrome. The set up for the scraper involved acquiring the appropriate web driver for the Chrome version (107), analyzing the website to acquire the executable paths of the data tables, and writing the code to execute the scraping. The scraper was automated to run to a stopping condition of the number of cities to scrape for the specified date range. All of the data acquired was in the year 2018. We ended with weather data including the high and low temperatures, high and low dew points, maximum wind speed for 200 cities with 365 days for each city in 2018. To speed up the scraping process, we ran the scraper in headless mode where the GUI for the browser was not loaded. Scraping the data for a year for a particular city took about 20 minutes for a total of around 66 hours of time for the scraper to get all of the data. The scraping rate equates to around 18 days scraped per minute. This was done almost entirely without human intervention, except for reinitializing the scraper for a new set of 20 cities every 7 hours or so. Once the 20 city groupings were acquired, we aggregated them into one csv. Initially, we scraped 200 cities, but the website we scraped did not have any or full data for 18 of these stations. This resulted in our final dataset having 182 cities.

For the airport flow data, we utilized 2017 data instead of 2018 because we are forecasting flights in 2018, but the information about the airport passenger flow is not synchronized in real time. That is, in 2018, the airport passenger flow data would not be accessible for 2018, so forecasts must rely on the 2017 data.

Once all the data was acquired, we matched each flight with the weather data based on the dates and joined the two dataframes. Then, we matched up each flight with their corresponding passenger flow data from the DOT and merged the data frames again.

After we collected all the data, we performed some feature engineering. First, we used One-Hot Encoding via pandas.get_dummies instead of our commonly used label Encoding. This is because our categorical data does not have any hierarchical structure. For example, easy is 1, medium is 2, difficult is 3. In our case the day of the week or airline does not represent such an attribute. In fact, One-Hot Encoding is very easy to understand. Each corresponding value is 0 or 1. 0 means it does not have the property, and 1 means it has the property. For example, on 2015-01-01, the value corresponding to Thursday is 1, which means that 2015-01-01 is Thursday. At the end, we need to drop week_Friday and one airline column to avoid multicollinearity. We also performed a correlation analysis and found that almost all of the variables were not correlated with each other.

Finally, we split the data into 80/20 training and test sets where the split was done so that the chronological order was maintained. Essentially, we picked a date so that the training set was about 80% and the test set was around 20% of the total data that we had available. Each of the models utilized the same training and test sets.

3. Analytics models

Baseline Model: We assume that all of the flights are on time, which means that the delay should be 0. For the test set: we have 646460 on time flight and 148005 delayed flights, so the accuracy is 646460÷(646460+148005) = 0.8137. The accuracy is already very high, and this makes sense in real life, because most of the time, the flight won't be delayed. This naive baseline model is what we compare to and judge against our models.

Logistic Regression: We use the logistic regression model to predict the probability of a successful outcome of the dependent variable Y (delay). The formula we are using:

$$Pr(Y = 1|X) = \frac{1}{1+e^{-(\beta_0+ \beta_1 X_1+ \beta_2 X_2+...+\beta_p X_p)}}$$

The threshold we are using is 0.5, which means that if the probability is greater than 0.5, then we say that the flight will be delayed. The Accuracy we got is: 0.8595. This is higher than the baseline model, so it's a good sign, however, we still need to consider the confusion matrix:

Actuals\Predictions	Positive	Negative
Positive	36439	111566
Negative	518	645942

Now, we can calculate the True Positive Rate (TPR) = 36439 / (36439 + 111566) = 0.246
False Positive Rate (FPR) = 518 / (518 + 645942) = 0.0008.

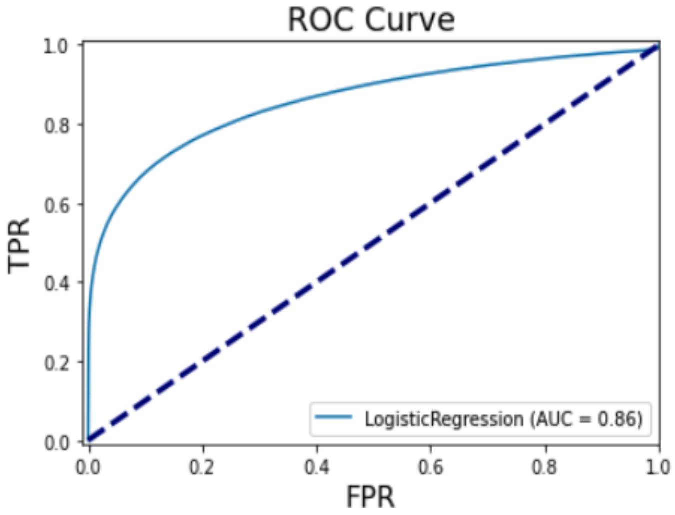

Figure 1: ROC Curve for Logistic Regression Model

Next, we draw the ROC Curve. ROC Curve plots the TPR and FPR for every break-even threshold p between 0 and 1, and we want ROC Curves that simultaneously achieve a high TPR and low FPR. This corresponds to a high area under the ROC Curve: Area Under the Curve (AUC). AUC is used to evaluate the overall quality of the logistic regression model. The AUC for our logistic regression model is 0.86, which is good.

Decision Tree: The second model that we built is the Decision Tree model.

We built a 10-fold cross validation decision tree model to select the complexity parameter. From Figure 2, we can see that the mean validation accuracy is the highest when the complexity parameter is .004, and the validation accuracy is 0.8164. The performance on the testing set is also pretty good. The accuracy on the test set is 0.78, which is slightly lower than the logistic regression model. However, the True Positive Rate and False Positive Rate are much lower. The true positive rate is 0.22 and the false positive rate is 0.0299 for the decision tree model.

The result of the decision tree model is easier to interpret, as it is shown on Figure 3. But it might not be very robust, if the training data change slightly, for example, if we do a random split instead of a chronological separation, it can come up with a totally different fitted tree model.

Random Forests: We have done logistic regression, which performed better than our baseline model. We have also built a CART model, which performed slightly worse than our baseline model. Now, we will utilize random forests to see if it can produce a better and steadier result.

To try to boost the performance of this model, we decided to do some further feature engineering by adding the variable delay_AR1. This field is a binary variable which is 1 if the previous flight was delayed and 0 if not. We used the GridSearchCV function to determine the model's optimal parameters. Since flights with late departures account for 20% of all flights, we set the corresponding weights of delayed and not delayed to one to five.

The best maximum number of features is seven in the random forest model, which is roughly the square root of the total feature number. And we find the following parameters as the best: min_samples_leaf as 500, n_estimators as 100, class_weight as {0: 1, 1: 5}.

We get the following

Accuracy	0.628
TPR	0.633
FPR	0.3746

as well as the feature importance graph (Figure 4), which shows that the most important features are departure time, arrival time, and delay_AR1 to predict whether the flight will be on time or delayed.

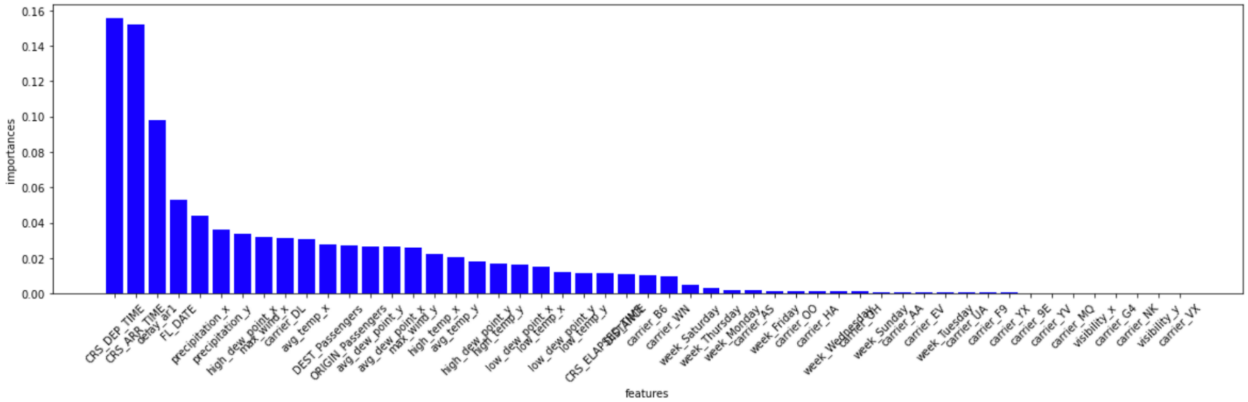

Figure 4: Feature Importance for Random Forest Model

Random forest models are less accurate than logistic regression models. This may be due to a strong linear relationship between the independent and dependent variables. When compared with the CART model, random forest prevents overfitting and has good generalization ability through data sampling and feature sampling. But it takes longer to train than CART and is not as good at visualization as a single decision tree.

4. Model Comparison, Impacts and Conclusion

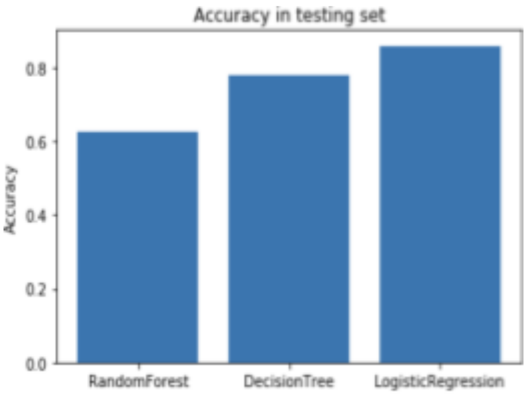

Figure 5: Accuracy in Testing Set

We treat the delayed takeoff of the aircraft as a positive sample and the absence of delay as a negative sample. When our prediction results are the same as the true value, it is regarded as a true positive (TP) or a true negative (TN) sample, otherwise, it is regarded as a false positive (FP) or false negative (FN) sample.

In general, the performances of the methods can be compared using different statistical criteria. Accuracy, True Positive Rate (TPR), and False Positive Rate (FPR) are widely used.

Accuracy means the proportion of the correctly predicted sample in the total sample, the higher the better. A good prediction has higher accuracy, higher TPR, and lower FPR.

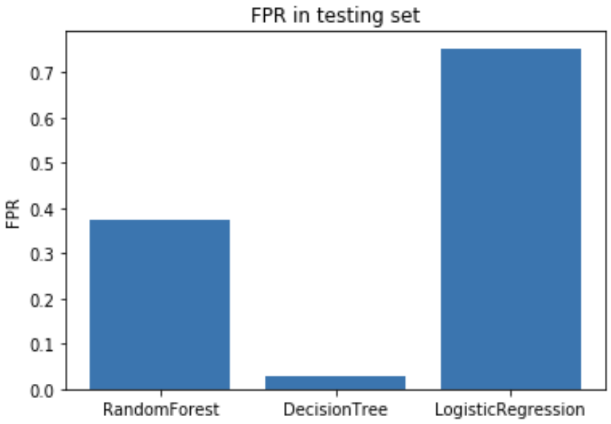

Figure 6: FPR in Testing Set

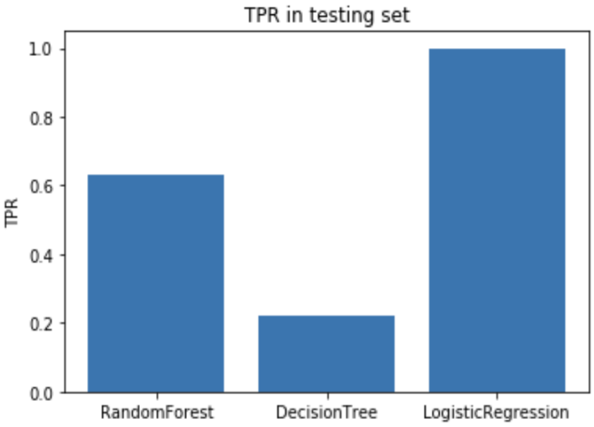

Figure 7: TPR in Testing Set

As shown above, the “Logistic Regression” predicted lots of data into the “Positive” label, causing a large FPR rate. This means the prediction of the “Positive” label can not be trusted. Therefore, if we use "logistic regression" to judge whether the aircraft is delayed, we will be utilizing a model which falsely predicts that many on-time flights are delayed.

The “Decision Tree” has a lower score than “Logistic Regression” in Accuracy but has the lowest score in FPR and TPR. It means that this algorithm makes the right “Negative” prediction more often than our other models. Therefore, if we use a "decision tree" to determine whether the aircraft is delayed, there is a higher chance that the delayed prediction is accurate.

“Random Forest” has the lowest score in Accuracy, the TPR and FPR of this algorithm are between the others. It means this algorithm has a balanced performance during these algorithms. Therefore, if we use "random forest" to determine whether the plane is delayed, the accuracy is greater than 60%.

The only model that performed better in accuracy compared to the baseline model was the Logistic Regression. However, the baseline model has a TPR and FPR of 0%, since it always predicts that the flight will be delayed. In other words, the amount of true positives will always be 0 and the amount of false positives will always be 0, since the baseline model never predicts a positive. Each of these models improves on the TPR, but have higher FPR rates as expected.

In conclusion, the methods we propose can be used to predict whether the aircraft is delayed, and we can get different prediction results depending on different algorithms. If we want to predict more “Positive” labels or get higher TPR and Accuracy, we should choose “Logistic Regression”, but we should know that these “positive” labels may not be trusted. If we want to predict more “Negative” labels, we should choose the “Decision Tree”; these “negative” labels can be trusted.

Ultimately, a flier must decide which is most important to them. If they are more risk averse and hate getting delayed, then the logistic regression model would be more favorable. It is able to predict the true positives at a much higher rate than the others, so a risk-averse flier would use this model to predict a delay and reschedule their flight accordingly. On the other hand, if a flier was relatively indifferent to getting delayed, then the CART model might be a better choice. This is because, while it captures less true positives, it is more accurate when it determines that a flight is delayed.

Figure Appendix

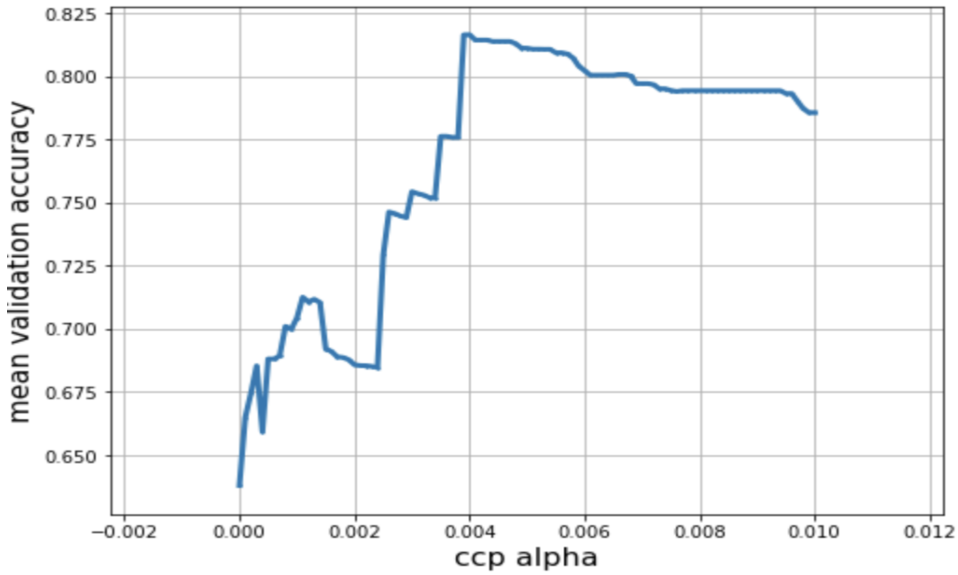

Figure 2: Accuracy vs. Complexity Parameter for Decision Tree Model

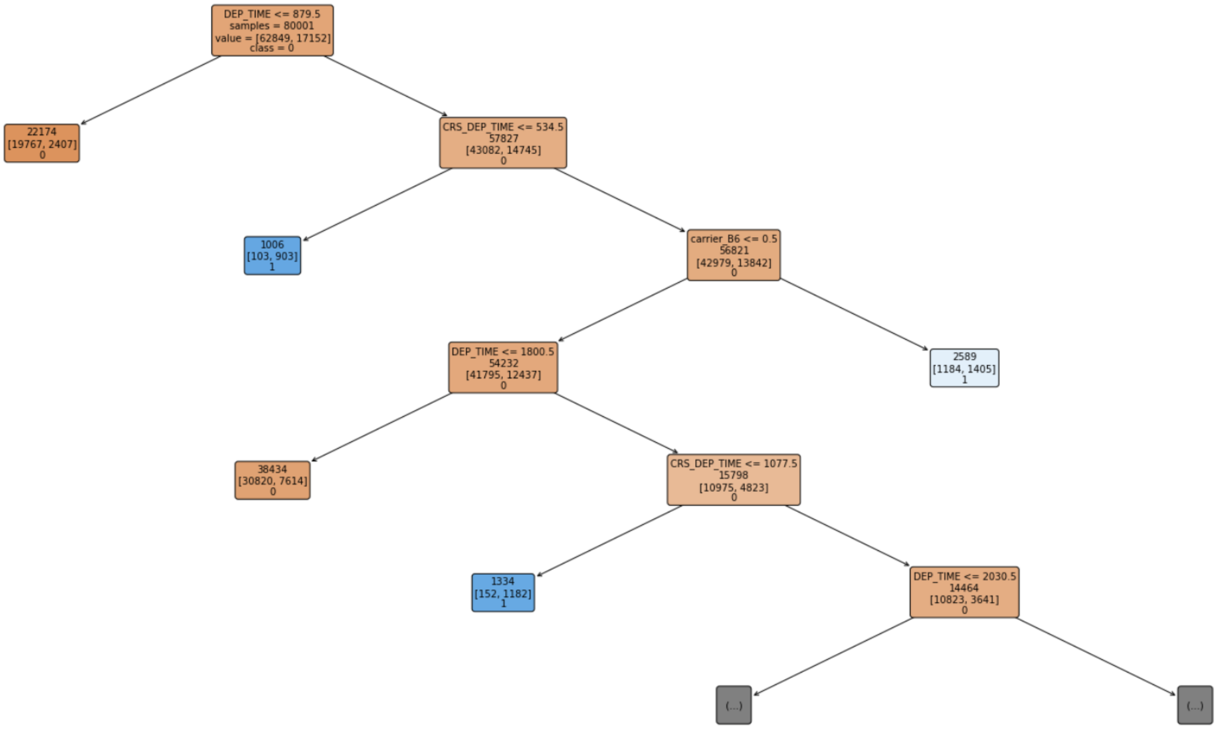

Figure 3: Final Decision Tree model Visualization

Code Appendix

December 11, 2022

1 RF

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
#from plotly import express as px
import numpy as np
import pandas as pd
#import plotly.io as pio
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
#from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
import re
import time
import datetime


# In[2]:


flight2018=pd.read_csv("2018.csv")
flight2018.head()


# In[3]:
```

```
airport2017 = pd.read_csv('airport.csv')
airport2017=airport2017.iloc[0:-1:,2:4]
airport2017.columns=list(['Code','Passengers'])
airport2017.head()


# In[4]:


weather2018 = pd.read_csv('weather2018.csv')
weather2018.head()


# In[5]:


flight2018=flight2018.iloc[:,0:-1]
flight2018.head()
flight2018.fillna(0,inplace=True)#because NA means there is no delay so we can␣
 ↪fill it with 0
flight2018["delay"]=(flight2018[["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY",
 ↪!= 0).astype(int).sum(axis=1)
flight2018["delay"][flight2018.delay>1]=1
flight2018=flight2018.
 ↪drop(["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY","LATE_AIRCRAFT_DELAY"],a
flight2018['FL_DATE'] = pd.to_datetime(flight2018['FL_DATE'], errors='coerce')
flight2018[ "week"] = flight2018[ "FL_DATE"].dt.day_name()
encoder = OneHotEncoder(handle_unknown='ignore')
#perform one-hot encoding on 'week' and 'OP_CARRIER' column
encoder_week=pd.get_dummies(flight2018.week,prefix="week")
flight2018=flight2018.join(encoder_week)
encoder_carrier=pd.get_dummies(flight2018.OP_CARRIER,prefix="carrier")
flight2018=flight2018.join(encoder_carrier)
#flight2018.drop(labels=['week'],axis=1, inplace=True)
cols=["CRS_DEP_TIME","CRS_ARR_TIME"]
flight2018["DISTANCE"] = flight2018["DISTANCE"]  / flight2018["DISTANCE"].abs().
 ↪max()
for col in cols:
    hour=flight2018[col]//100  #        00 00
    mins=flight2018[col]%100
    flight2018[col]=hour*60+mins


# In[6]:


for i in range(len(airport2017)):
    airport2017["Passengers"][i]=int(airport2017["Passengers"][i].replace(',',''))
#min max normalization
airport2017["Passengers"] = airport2017["Passengers"]  / airport2017["Passengers"].abs().max()
merge1=pd.merge(flight2018, airport2017, left_on='ORIGIN', right_on='Code')
merge1.rename(columns = {'Passengers':'ORIGIN_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)
merge1=pd.merge(merge1, airport2017, left_on='DEST', right_on='Code')
merge1.rename(columns = {'Passengers':'DEST_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)


# In[7]:


pd.set_option('display.max_columns', None) #


# In[8]:


merge1["FL_DATE"]=merge1["FL_DATE"].astype('datetime64[ns]')
weather2018["date"]=weather2018["date"].astype('datetime64[ns]')
merge1=merge1.merge(weather2018, how="inner",left_on=['FL_DATE','ORIGIN'], right_on=['date','cities'])
merge1.drop(labels=['cities','date'],axis=1, inplace=True)
final=pd.merge(merge1, weather2018, how="inner", left_on=['FL_DATE','DEST'], right_on=['date','cities'])
final.drop(labels=['cities','date'],axis=1, inplace=True)


# In[9]:


final


# In[10]:


final1 = final
#final1['FL_DATE'] = final1['FL_DATE'].apply(lambda x: x.strftime("%m%d"))


# In[11]:
```

```
#create delay lag 1 (AR(1))
final1.sort_values(["OP_CARRIER", "OP_CARRIER_FL_NUM",␣
 ↪"ORIGIN","DEST","FL_DATE"], inplace=True, ascending=True)

final1['delay_ar1'] = final1['delay']
final1['delay_ar1'] = final1['delay_ar1'].shift() #create

final1['OP_CARRIER_ar1'] = final1['OP_CARRIER']
final1['OP_CARRIER'] = final1['OP_CARRIER'].shift()
final1['OP_CARRIER_FL_NUM_ar1'] = final1['OP_CARRIER_FL_NUM']
final1['OP_CARRIER_FL_NUM_ar1'] = final1['OP_CARRIER_FL_NUM_ar1'].shift()
final1['ORIGIN_ar1'] = final1['ORIGIN']
final1['ORIGIN_ar1'] = final1['ORIGIN_ar1'].shift()
final1['DEST_ar1'] = final1['DEST']
final1['DEST_ar1'] = final1['DEST_ar1'].shift()

final1 = final1.drop(final1[(final1['OP_CARRIER'] != final1['OP_CARRIER_ar1'])␣
 ↪|                             (final1['OP_CARRIER_FL_NUM'] !=␣
 ↪final1['OP_CARRIER_FL_NUM_ar1']) |                          ␣
 ↪(final1['ORIGIN'] != final1['ORIGIN_ar1']) |                          ␣
 ↪(final1['DEST'] != final1['DEST_ar1']) ].index) #.index   | means or

final1 = final1.
 ↪drop(['OP_CARRIER_ar1','OP_CARRIER_FL_NUM_ar1','ORIGIN_ar1','DEST_ar1'],axis=1)
final1


# In[12]:


final1.head(50)


# In[13]:


final1.sort_values(["FL_DATE"], inplace=True, ascending=True)


# In[14]:


#df = final1.iloc[0:1000000,:]
df = final1.iloc[:,:]
```

```
df.
 ↪drop(labels=['ORIGIN','DEST','OP_CARRIER','OP_CARRIER_FL_NUM','DEP_TIME','DEP_DELAY','TAXI_
 ↪inplace=True)
df['FL_DATE'] = df['FL_DATE'].apply(lambda x: x.strftime("%m%d")) # convert␣
 ↪datetime to object


# In[15]:


final1['delay'].sum()


# In[16]:


#df.info()
df_train = df.iloc[0:int(0.8*len(df)),:]
df_test = df.iloc[int(0.8*len(df)):,:]

y_train = df_train['delay']
x_train = df_train.drop(['delay'], axis=1)

y_test = df_test['delay']
x_test = df_test.drop(['delay'], axis=1)
x_test


# In[17]:


from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
import time
np.random.seed(10)

grid_values = {'max_features': np.linspace(7,7,1, dtype='int32'),
              'min_samples_leaf': [500],
              'n_estimators': [100],
              'random_state': [10]}

rf2 = RandomForestClassifier(class_weight = {0: 1, 1: 5})
rf_cv2 = GridSearchCV(rf2, param_grid=grid_values, scoring='accuracy', cv=5)
rf_cv2.fit(x_train, y_train)


# In[18]:
```

```
rf_cv2.best_params_


# In[19]:


rf_cv2.best_score_


# In[20]:


y_pred = rf_cv2.predict(x_test)
y_pred.sum()


# In[21]:


# Accuracy
def Accuracy(y_pred, y_test):
    cm = confusion_matrix(y_test, y_pred)
    return (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())

#True Positive Rate (TPR)
#TPR = TP / (FN + TP)
def TPR(y_pred, y_test):
    cm = confusion_matrix(y_test, y_pred)
    return cm.ravel()[3]/(cm.ravel()[2]+cm.ravel()[3])

#False Positive Rate (FPR)
#FPR = FP / (TN + FP)
def FPR(y_pred, y_test):
    cm = confusion_matrix(y_test, y_pred)
    return cm.ravel()[1]/(cm.ravel()[0]+cm.ravel()[1])


# In[22]:


from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score

#Recall is also called sensitiviy or TPR := tp / (tp + fn)
```

```
#Precision := tp / (tp + fp)

#y_pred = rf_cv.predict(x_test)
cm = confusion_matrix(y_test, y_pred)

print ("Confusion Matrix first tree: \n", cm)

print('Precision:',precision_score(y_test, y_pred))
print('Recall first tree:',recall_score(y_test, y_pred))
print ("\nAccuracy:", Accuracy(y_pred, y_test))
print('TPR:',TPR(y_pred, y_test))
print('FPR:',FPR(y_pred, y_test))
```

2 DT

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:



#import sys
#!conda install --yes --prefix {sys.prefix} plotly
get_ipython().system('conda install --yes wrapt')
import wrapt

import pandas as pd
from plotly import express as px
import numpy as np
import pandas as pd
import plotly.io as pio
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
import re
import time
import datetime


# In[2]:
```

```
flight2018=pd.read_csv("2018.csv")
flight2018.head()


# In[3]:


airport2017 = pd.read_csv('airport.csv')
airport2017=airport2017.iloc[0:-1:,2:4]
airport2017.columns=list(['Code','Passengers'])
airport2017.head()


# In[4]:


weather2018 = pd.read_csv('weather2018.csv')
weather2018.head()


# In[5]:


flight2018=flight2018.iloc[:,0:-1]
flight2018.head()
flight2018.fillna(0,inplace=True)#because NA means there is no delay so we can␣
 ↪fill it with 0
flight2018["delay"]=(flight2018[["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY",
 ↪!= 0).astype(int).sum(axis=1)
flight2018["delay"][flight2018.delay>1]=1
flight2018=flight2018.
 ↪drop(["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY","LATE_AIRCRAFT_DELAY"],a
flight2018['FL_DATE'] = pd.to_datetime(flight2018['FL_DATE'], errors='coerce')
flight2018[ "week"] = flight2018[ "FL_DATE"].dt.day_name()
encoder = OneHotEncoder(handle_unknown='ignore')

#perform one-hot encoding on 'week' and 'OP_CARRIER' column
encoder_week=pd.get_dummies(flight2018.week,prefix="week")
flight2018=flight2018.join(encoder_week)

encoder_carrier=pd.get_dummies(flight2018.OP_CARRIER,prefix="carrier")
flight2018=flight2018.join(encoder_carrier)
#flight2018.drop(labels=['week'],axis=1, inplace=True)

cols=["CRS_DEP_TIME","CRS_ARR_TIME"]
```

```
flight2018["DISTANCE"] = flight2018["DISTANCE"]  / flight2018["DISTANCE"].abs().max()
for col in cols:
    hour=flight2018[col]//100  #        00 00
    mins=flight2018[col]%100
    flight2018[col]=hour*60+mins


# In[6]:


for i in range(len(airport2017)):
    airport2017["Passengers"][i]=int(airport2017["Passengers"][i].replace(',',''))
#min max normalization
airport2017["Passengers"] = airport2017["Passengers"]  / airport2017["Passengers"].abs().max()
merge1=pd.merge(flight2018, airport2017, left_on='ORIGIN', right_on='Code')
merge1.rename(columns = {'Passengers':'ORIGIN_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)
merge1=pd.merge(merge1, airport2017, left_on='DEST', right_on='Code')
merge1.rename(columns = {'Passengers':'DEST_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)


# In[7]:


pd.set_option('display.max_columns', None) #


# In[8]:


merge1["FL_DATE"]=merge1["FL_DATE"].astype('datetime64[ns]')
weather2018["date"]=weather2018["date"].astype('datetime64[ns]')
merge1=merge1.merge(weather2018, how="inner",left_on=['FL_DATE','ORIGIN'], right_on=['date','cities'])
merge1.drop(labels=['cities','date'],axis=1, inplace=True)
final=pd.merge(merge1, weather2018, how="inner", left_on=['FL_DATE','DEST'], right_on=['date','cities'])
final.drop(labels=['cities','date'],axis=1, inplace=True)


# In[9]:
```

```
final


# ### Decision Tree

# In[10]:


from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier


# In[11]:


print(final.corr())


# In[12]:


# Drop columns that we don't need for the model
df_dt = final.drop(['FL_DATE', 'OP_CARRIER', 'OP_CARRIER_FL_NUM', 'ORIGIN',␣
 ↪'DEST','week',
                        'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON', 'TAXI_IN',
                        'CANCELLED', 'CANCELLATION_CODE', 'DIVERTED', 'ARR_TIME',
                        'ARR_DELAY', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME', 'week_Friday',
                        'carrier_9E', # dummies-drop first 'DEP_DELAY', 'DISTANCE'], axis = 1)

df_dt.head(5)


# In[13]:


df_dt.shape


# In[14]:


# Split Traning set(70%) and Test set(30%)

#flight_train, flight_test = train_test_split(df_dt, test_size=0.3,␣
 ↪random_state=42)

#flight_train = df_dt.loc[:80000, :]
```

```
#flight_test = df_dt.loc[80001:100000, :]

flight_train = df_dt.loc[:2873004, :]
flight_test = df_dt.loc[2873005:, :]

flight_train.shape, flight_test.shape


# In[15]:


# X_train, y_train, X_test, y_test

X_train = flight_train.drop(['delay'], axis = 1)
y_train = flight_train['delay']

X_test = flight_test.drop(['delay'], axis = 1)
y_test = flight_test['delay']


# In[16]:


grid_values = {'ccp_alpha': np.linspace(0, 0.01, 101)}

dtc = DecisionTreeClassifier(min_samples_leaf=5,
                             ccp_alpha= 0.001,
                             criterion = 'gini',
                             random_state = 88)
dtc_cv = GridSearchCV(dtc, param_grid=grid_values, scoring = 'accuracy', cv=10,␣
 ↪verbose=1)
dtc_cv.fit(X_train, y_train)


# In[17]:


acc = dtc_cv.cv_results_['mean_test_score']
# what sklearn calls mean_test_score is the holdout set, i.e. the validation␣
 ↪set.
ccp = dtc_cv.cv_results_['param_ccp_alpha'].data

pd.DataFrame({'ccp alpha' : ccp, 'Validation Accuracy': acc}).head(20)


# In[18]:
```

```
plt.figure(figsize=(8, 6))
plt.xlabel('ccp alpha', fontsize=16)
plt.ylabel('mean validation accuracy', fontsize=16)
plt.scatter(ccp, acc, s=2)
plt.plot(ccp, acc, linewidth=3)
plt.grid(True, which='both')
plt.show()


# In[19]:


print('The best complexity parameter from the cross validation:')
dtc_cv.best_params_


# In[20]:


print('Grid best score (Accuracy): ')
dtc_cv.best_score_


# In[21]:


print('Node count =')
dtc_cv.best_estimator_.tree_.node_count


# In[22]:


import matplotlib.pyplot as plt
from sklearn.tree import plot_tree
plt.figure(figsize=(30,15))
plot_tree(dtc_cv.best_estimator_,
          feature_names=X_train.columns,
          class_names=['0','1'],
          filled=True,
          impurity=False,
          rounded=True,
          fontsize=10,
          max_depth = 5,
          label='root')
plt.show()
```

```python


# Confusion Matrix:

# In[23]:


from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score

y_pred = dtc_cv.predict(X_test)
cm = confusion_matrix(y_test, y_pred)


# In[24]:


# Accuracy
acc_dt = (cm.ravel()[0]+cm.ravel()[3])/sum(cm.ravel())
print('Decision Tree Accuracy is: ')
print(acc_dt)


# In[25]:


TPR_dt = cm.ravel()[3] / (cm.ravel()[3] + cm.ravel()[2])
print('Decision Tree True Positive Rate (TPR) is: ')
print(TPR_dt)

# In[26]:
# False Positive rate
FPR_dt = cm.ravel()[1] / (cm.ravel()[1] + cm.ravel()[0])
print('Decision Tree False Positive Rate (FPR) is: ')
print(FPR_dt)
```

3 LR

```python
#!/usr/bin/env python
# coding: utf-8

# In[1]:


import pandas as pd
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
import re
import time
import datetime


# In[2]:


flight2018=pd.read_csv("2018.csv")
flight2018.head()


# In[3]:

airport2017 = pd.read_csv('airport.csv')
airport2017=airport2017.iloc[0:-1:,2:4]
airport2017.columns=list(['Code','Passengers'])
airport2017.head()

# In[4]:

weather2018 = pd.read_csv('weather2018.csv')
weather2018.head()


# In[5]:


flight2018=flight2018.iloc[:,0:-1]
flight2018.head()
flight2018.fillna(0,inplace=True)#because NA means there is no delay so we can␣
 ↪fill it with 0
flight2018["delay"]=(flight2018[["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY",
 ↪!= 0).astype(int).sum(axis=1)
flight2018["delay"][flight2018.delay>1]=1
flight2018=flight2018.
 ↪drop(["CARRIER_DELAY","WEATHER_DELAY","NAS_DELAY","SECURITY_DELAY","LATE_AIRCRAFT_DELAY"],a
flight2018['FL_DATE'] = pd.to_datetime(flight2018['FL_DATE'], errors='coerce')
flight2018[ "week"] = flight2018[ "FL_DATE"].dt.day_name()
encoder = OneHotEncoder(handle_unknown='ignore')
```

```
#perform one-hot encoding on 'week' and 'OP_CARRIER' column
encoder_week=pd.get_dummies(flight2018.week,prefix="week")
flight2018=flight2018.join(encoder_week)
encoder_carrier=pd.get_dummies(flight2018.OP_CARRIER,prefix="carrier")
flight2018=flight2018.join(encoder_carrier)
#flight2018.drop(labels=['week'],axis=1, inplace=True)
cols=["CRS_DEP_TIME","CRS_ARR_TIME"]
flight2018["DISTANCE"] = flight2018["DISTANCE"]  / flight2018["DISTANCE"].abs().
 ↪max()
for col in cols:
    hour=flight2018[col]//100  #        00 00
    mins=flight2018[col]%100
    flight2018[col]=hour*60+mins


# In[6]:


for i in range(len(airport2017)):
    airport2017["Passengers"][i]=int(airport2017["Passengers"][i].
 ↪replace(',',''))
#min max normalization
airport2017["Passengers"] = airport2017["Passengers"]  /␣
 ↪airport2017["Passengers"].abs().max()
merge1=pd.merge(flight2018, airport2017, left_on='ORIGIN', right_on='Code')
merge1.rename(columns = {'Passengers':'ORIGIN_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)
merge1=pd.merge(merge1, airport2017, left_on='DEST', right_on='Code')
merge1.rename(columns = {'Passengers':'DEST_Passengers'},inplace=True)
merge1.drop(labels=['Code'],axis=1, inplace=True)


# In[7]:


pd.set_option('display.max_columns', None) #


# In[8]:


merge1["FL_DATE"]=merge1["FL_DATE"].astype('datetime64[ns]')
weather2018["date"]=weather2018["date"].astype('datetime64[ns]')
merge1=merge1.merge(weather2018, how="inner",left_on=['FL_DATE','ORIGIN'],␣
 ↪right_on=['date','cities'])
merge1.drop(labels=['cities','date'],axis=1, inplace=True)
```

```
final=pd.merge(merge1, weather2018, how="inner", left_on=['FL_DATE','DEST'],␣
 ↪right_on=['date','cities'])
final.drop(labels=['cities','date'],axis=1, inplace=True)


# In[9]:


final


# In[31]:


final1 = final
#create delay lag 1 (AR(1))
final1.sort_values(["OP_CARRIER", "OP_CARRIER_FL_NUM",␣
 ↪"ORIGIN","DEST","FL_DATE"], inplace=True, ascending=True)

final1['delay_ar1'] = final1['delay']
final1['delay_ar1'] = final1['delay_ar1'].shift() #create

final1['OP_CARRIER_ar1'] = final1['OP_CARRIER']
final1['OP_CARRIER'] = final1['OP_CARRIER'].shift()
final1['OP_CARRIER_FL_NUM_ar1'] = final1['OP_CARRIER_FL_NUM']
final1['OP_CARRIER_FL_NUM_ar1'] = final1['OP_CARRIER_FL_NUM_ar1'].shift()
final1['ORIGIN_ar1'] = final1['ORIGIN']
final1['ORIGIN_ar1'] = final1['ORIGIN_ar1'].shift()
final1['DEST_ar1'] = final1['DEST']
final1['DEST_ar1'] = final1['DEST_ar1'].shift()

final1 = final1.drop(final1[(final1['OP_CARRIER'] != final1['OP_CARRIER_ar1'])␣
 ↪|                               (final1['OP_CARRIER_FL_NUM'] !=␣
 ↪final1['OP_CARRIER_FL_NUM_ar1']) |                               ␣
 ↪(final1['ORIGIN'] != final1['ORIGIN_ar1']) |                            ␣
 ↪(final1['DEST'] != final1['DEST_ar1']) ].index) #.index   | means or

final1 = final1.
 ↪drop(['OP_CARRIER_ar1','OP_CARRIER_FL_NUM_ar1','ORIGIN_ar1','DEST_ar1'],axis=1)
final1


# In[44]:


# Drop columns that we don't need
```

```
df = final1.drop(['FL_DATE', 'OP_CARRIER', 'OP_CARRIER_FL_NUM', 'ORIGIN',␣
 ↪'DEST','week',
                    'TAXI_OUT', 'WHEELS_OFF', 'WHEELS_ON', 'TAXI_IN',
                    'CANCELLED', 'CANCELLATION_CODE', 'DIVERTED',
                    'ARR_TIME', 'ARR_DELAY', 'ACTUAL_ELAPSED_TIME', 'AIR_TIME',
                    'DEP_DELAY', 'DISTANCE', 'week_Friday', 'carrier_9E'], axis␣
 ↪= 1) # Drop the real arrival time and the dummy variable to avoid␣
 ↪colineararity
df.head(5)


# In[45]:


df_train = df.iloc[0:int(0.8*len(df)),:]
df_test = df.iloc[int(0.8*len(df)):,:]
df_train.shape, df_test.shape


# In[46]:


X_train = df_train.drop(['delay'], axis = 1)
y_train = df_train['delay']

X_test = df_test.drop(['delay'], axis = 1)
y_test = df_test['delay']


# In[47]:


# Baseline model
default_false = np.sum(df_train['delay'] == 0)
default_true = np.sum(df_train['delay'] == 1)


# In[48]:


# Accuracy of baseline model based on training data:
ACC = default_false/(default_false + default_true)
ACC


# In[49]:
```

```python
# Compute accuracy of baseline on testing:
test_default_false = np.sum(df_test['delay'] == 0)
test_default_true = np.sum(df_test['delay'] == 1)
test_ACC = test_default_false/(test_default_false + test_default_true)
test_ACC


# In[50]:


# Logistic Regression Model

cols = df_train.columns[df_train.columns != "delay"]
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X=df_train[cols], y=df_train['delay'])


# In[51]:


# Accuracy (The accuracy is greater than the baseline model.)

print(model.score(X=df_train[cols], y=df_train['delay']))


# In[52]:


# Confusion matrix

from sklearn.metrics import confusion_matrix
y_test = df_test['delay']
y_pred = model.predict(df_test[cols])
cm = confusion_matrix(y_test, y_pred)
cm


# In[53]:


# ROC Curve

from sklearn.metrics import plot_roc_curve

plot_roc_curve(model, df_test[cols], df_test['delay'])
```

```
plt.plot([0, 1], [0, 1], color='navy', lw=3, linestyle='--')
plt.title('ROC Curve', fontsize=18)
plt.xlabel('FPR', fontsize=16)
plt.ylabel('TPR', fontsize=16)
plt.xlim([-0.01, 1.00])
plt.ylim([-0.01, 1.01])
plt.legend(loc='lower right', fontsize=10)
plt.show()
```

```
import pandas as pd
#from plotly import express as px
import numpy as np
import pandas as pd
#import plotly.io as pio
import matplotlib.pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import OneHotEncoder
import tensorflow as tf
#from tensorflow.keras.layers.experimental import preprocessing
from tensorflow.keras import layers
import warnings
warnings.filterwarnings('ignore')
from datetime import datetime
import re
import time
import datetime
```

rf_data

```
# in TestSet
rf_Accuracy = 0.626792873191393
rf_TPR = 0.6330190273412417
rf_FPR = 0.3746318752706799
```

Decision Tree

```
# in TestSet
decision_tree_Accuracy = 0.7779
decision_tree_TPR = 0.2213130722774206
decision_tree_FPR = 0.029932064303490952
```

Logistic Regression

```
# in TrainSet
logistic_regression_Accuracy_train = 0.8595082412404075
```

```
# in TestSet
#confusion_matrix
cm = confusion_matrix(y_test, y_pred)
cm = np.array([[645942,    518],
       [111566,  36439]])
logistic_regression_Accuracy = 0.8589188951055112
```

```
(645942 + 36439) / (645942 + 36439 +518+ 111566)
```

```
0.8589188951055112
```

```
645942 / (645942 + 518)
```

```
0.9991987129907496
```

```
111566 / (111566 + 36439)
```

```
0.7537988581466842
```

```
base_Accuracy = 0.8137048202249313
rf_Accuracy = 0.626792873191393
dt_Accuracy = 0.7779
lr_Accuracy = 0.8589188951055112
```

```
import matplotlib.pyplot as plt
```

```
plt.bar(["RandomForest","DecisionTree","LogisticRegression"],[rf_Accuracy,dt_Accuracy,lr_Accuracy])
plt.ylabel("Accuracy")
plt.title("Accuracy in testing set")
# plt.xticks(rotation = 90)
plt.savefig("Training Accuracy.png")
```

TPR

```
rf_TPR = 0.6330190273412417
dt_TPR = 0.2213130722774206
lr_TPR = 0.9991987129907496

plt.bar(["RandomForest","DecisionTree","LogisticRegression"],[rf_TPR,dt_TPR,lr_TPR])
plt.ylabel("TPR")
plt.title("TPR in testing set")
# plt.xticks(rotation = 90)
# plt.savefig("Training Accuracy.png")
```

```
Text(0.5, 1.0, 'TPR in testing set')
```

FPR

```
rf_FPR = 0.3746318752706799
dt_FPR = 0.029932064303490952
lr_FPR = 0.7537988581466842
plt.bar(["RandomForest","DecisionTree","LogisticRegression"],[rf_FPR,dt_FPR,lr_FPR])
plt.ylabel("FPR")
plt.title("FPR in testing set")
# plt.xticks(rotation = 90)
# plt.savefig("Training Accuracy.png")
```

```
Text(0.5, 1.0, 'FPR in testing set')
```

```
#train:
lr_acc_train = 0.8595082412404075
```