```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'flight-delays:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F810%2F1496%2Fbundle%2Farchive.zip%3FX-Goog

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')



import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
flights=pd.read_csv("../input/flight-delays/flights.csv")
flights.head(10)
```

```
flights.shape
```

```
flights.isnull().sum() # Checking how many null values in each column in our data set
```

```
import seaborn as sns
```

```
sns.countplot(x="CANCELLATION_REASON",data=flights)
```

```
#Reason for Cancellation of flight: A - Airline/Carrier; B - Weather; C - National Air System; D - Security
#We can observe from graph easily that mostly Whether is responsible for delays of flight.
```

```
sns.countplot(x="MONTH",hue="CANCELLATION_REASON",data=flights)
```

```
flights.isnull().sum()*100/flights.shape[0]
```

```
df_sample=flights
```

```
plt.figure(figsize=(10, 10))
axis = sns.countplot(x=df_sample['ORIGIN_AIRPORT'], data =df_sample,
              order=df_sample['ORIGIN_AIRPORT'].value_counts().iloc[:20].index)
axis.set_xticklabels(axis.get_xticklabels(), rotation=90, ha="right")
plt.tight_layout()
plt.show()
```

```
axis = plt.subplots(figsize=(10,14))
sns.despine(bottom=True, left=True)
# Observations with Scatter Plot
sns.stripplot(x="ARRIVAL_DELAY", y="AIRLINE",
              data = df_sample, dodge=True, jitter=True
             )
plt.show()
```

```
axis = plt.subplots(figsize=(10,14))
Name = df_sample["AIRLINE"].unique()
size = df_sample["AIRLINE"].value_counts()
plt.pie(size,labels=Name,autopct='%5.0f%%')
plt.show()
```

```
axis = plt.subplots(figsize=(20,14))
sns.heatmap(df_sample.corr(),annot = True)
plt.show()
```

```
# Very High Correlation Between Arrival Delay and Departure Delay¶
#It shows that maximum of the Arrival Delays are due to the Departure Delays.
```

```
corr=df_sample.corr()
corr
```

```
varaibles_to_remove=['YEAR','FLIGHT_NUMBER',
       'TAIL_NUMBER', 'DEPARTURE_TIME', 'TAXI_OUT',
       'WHEELS_OFF', 'ELAPSED_TIME', 'AIR_TIME',
       'WHEELS_ON', 'TAXI_IN', 'ARRIVAL_TIME', 'DIVERTED', 'CANCELLED', 'CANCELLATION_REASON',
       'AIR_SYSTEM_DELAY', 'SECURITY_DELAY', 'AIRLINE_DELAY',
       'LATE_AIRCRAFT_DELAY', 'WEATHER_DELAY']
```

```
flights.drop(varaibles_to_remove,axis=1,inplace=True)
```

```
flights.columns
```

```
flights.drop('SCHEDULED_TIME',axis=1,inplace=True)
```

```
flights.drop('SCHEDULED_ARRIVAL',axis=1,inplace=True)
```

```
flights.columns
```

```
airport=pd.read_csv("../input/flight-delays/airports.csv")

airport.head()

flights.loc[~flights.ORIGIN_AIRPORT.isin(airport.IATA_CODE.values),'ORIGIN_AIRPORT']='OTHER'
flights.loc[~flights.DESTINATION_AIRPORT.isin(airport.IATA_CODE.values),'DESTINATION_AIRPORT']='OTHER'

flights.head()

flights.ORIGIN_AIRPORT.nunique()

flights.DESTINATION_AIRPORT.nunique()

flights.AIRLINE.nunique()

flights.columns

flights.shape

flights=flights.dropna()

flights.head()

row_indexes=flights[flights['DAY_OF_WEEK']==1].index

flights.loc[row_indexes,'DAY_OF_WEEK']="SUNDAY"

flights.head(40)

row_indexes=flights[flights['DAY_OF_WEEK']==2].index

flights.loc[row_indexes,'DAY_OF_WEEK']="MONDAY"

row_indexes=flights[flights['DAY_OF_WEEK']==3].index

flights.loc[row_indexes,'DAY_OF_WEEK']="TUESDAY"

row_indexes=flights[flights['DAY_OF_WEEK']==4].index

flights.loc[row_indexes,'DAY_OF_WEEK']="WEDNESDAY"

row_indexes=flights[flights['DAY_OF_WEEK']==5].index

flights.loc[row_indexes,'DAY_OF_WEEK']="THRUSDAY"

row_indexes=flights[flights['DAY_OF_WEEK']==6].index

flights.loc[row_indexes,'DAY_OF_WEEK']="FRIDAY"

row_indexes=flights[flights['DAY_OF_WEEK']==7].index

flights.loc[row_indexes,'DAY_OF_WEEK']="SATURDAY"

flights.head(40)

dd=pd.DataFrame(flights)

dums = ['AIRLINE','ORIGIN_AIRPORT','DESTINATION_AIRPORT','DAY_OF_WEEK']

df_cat=pd.get_dummies(dd[dums],drop_first=True)
```

```
df_cat.columns
```

```
dd.columns
```

```
dd.drop("AIRLINE",axis=1,inplace=True)
```

```
dd.drop("ORIGIN_AIRPORT",axis=1,inplace=True)
```

```
dd.drop("DESTINATION_AIRPORT",axis=1,inplace=True)
```

```
dd.drop("DAY_OF_WEEK",axis=1,inplace=True)
```

```
import pandas as pd
```

```
df=pd.concat([dd,df_cat],axis=1)
```

```
df.shape
```

```
df.head(10)
```

```
final_data=df
```

```
final_data = final_data.sample(n=100000)
```

```
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
final_data.head(10)
```

```
X=final_data.drop("DEPARTURE_DELAY",axis=1)
Y=final_data.DEPARTURE_DELAY
```

```
Y
```

```
X.head(10)
```

```
Y.head(10)
```

```
from sklearn.model_selection import train_test_split, cross_val_score, cross_val_predict
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=0)
```

```
from sklearn.ensemble import RandomForestRegressor
reg_rf = RandomForestRegressor()
reg_rf.fit(X_train,y_train)
```

```
y_pred = reg_rf.predict(X_test)
```

```
reg_rf.score(X_train,y_train)
```

```
reg_rf.score(X_test,y_test)
```

```
metrics.r2_score(y_test,y_pred)
```

```
print('MAE:', metrics.mean_absolute_error(y_test,y_pred))
print('MSE:', metrics.mean_squared_error(y_test,y_pred))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,y_pred)))
```

```
pp=pd.DataFrame({'Actual':y_test,'Predicted':y_pred})
pp.head(10)
```

```
from sklearn.model_selection import RandomizedSearchCV
```

```
#Randomized Search CV

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 10, stop = 200, num = 12)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 5, 10]


# Create the random grid

random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}


# Random search of parameters, using 5 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv =


rf_random.fit(X_train,y_train)


rf_random.best_params_


p=rf_random.predict(X_test)


metrics.r2_score(y_test,p)


print('MAE:', metrics.mean_absolute_error(y_test,p))
print('MSE:', metrics.mean_squared_error(y_test,p))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,p)))


zz=pd.DataFrame({'Actual':y_test,'Predicted':p})
zz.head(30)
```

Boosting technique applying

```
from sklearn.ensemble import GradientBoostingRegressor


gbr=GradientBoostingRegressor(random_state=0)


GBR=gbr.fit(X_train,y_train)


pre =GBR.predict(X_test)


print('MAE:', metrics.mean_absolute_error(y_test,pre))
print('MSE:', metrics.mean_squared_error(y_test,pre))
print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test,pre)))


metrics.r2_score(y_test,pre)


gg=pd.DataFrame({'Actual':y_test,'Predicted':pre})
gg.head(20)
```

```
def predict(MONTH, DAY,SCHEDULED_DEPARTURE,
       DISTANCE, ARRIVAL_DELAY,AIRLINE,ORIGIN_AIRPORT,DESTINATION_AIRPORT,DAY_OF_WEEK):
    AIRLINE_index = np.where(X.columns==AIRLINE)[0][0]
    ORIGIN_index = np.where(X.columns==ORIGIN_AIRPORT)[0][0]
    DESTINATION_index = np.where(X.columns==DESTINATION_AIRPORT)[0][0]
    DAY_OF_WEEK_index = np.where(X.columns==DAY_OF_WEEK)[0][0]
    x= np.zeros(len(X.columns))
    x[0] = MONTH
    x[1] = DAY
    x[2] = SCHEDULED_DEPARTURE
    x[3] = DISTANCE
    x[4] = ARRIVAL_DELAY
    if AIRLINE_index >=0:
        x[AIRLINE_index] = 1
    if ORIGIN_index >=0:
        x[ORIGIN_index] = 1
    if DESTINATION_index >=0:
        x[DESTINATION_index] = 1
    if  DAY_OF_WEEK_index >= 0:
        x[ DAY_OF_WEEK_index] = 1




    return gbr.predict([x])[0]


predict(5,6,1515,328,-8.0,'AIRLINE_OO','ORIGIN_AIRPORT_PHX','DESTINATION_AIRPORT_ABQ','DAY_OF_WEEK_TUESDAY')
```

Start coding or generate with AI.