Flight Delay Analysis

```
import pandas as pd
# Mount google drive
from google.colab import drive
drive.mount('/content/drive')
#Import Libraries that I will use
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns # For additional visualizations
import gc # for garbage collection the colab ran over RAM in my runtime gc.collect()
# Import excel files into df from google drive
file_path = '/content/drive/My Drive/Dataset/'

# **1. Importing Data**
df=None
airlines_df = pd.read_excel(file_path+'Airlines.xlsx')
airports_df = pd.read_excel(file_path+'airports.xlsx')
runways_df = pd.read_excel(file_path+'runways.xlsx')
# Check import is correct
# Investigate data structures (modify column names if needed)
print(airlines_df.info())
print(airlines_df.shape)
print(airlines_df.head())
print(airports_df.info())
print(airports_df.shape)
print(airports_df.head())
print(runways_df.info())
print(runways_df.shape)
print(runways_df.head())

# dropping unnecesary columns
runways_df = runways_df.drop(['le_ident', 'le_latitude_deg','le_longitude_deg', 'le_elevation_ft', 'le_heading_degT',
       'le_displaced_threshold_ft', 'he_ident', 'he_latitude_deg','he_longitude_deg', 'he_elevation_ft', 'he_heading_degT',
       'he_displaced_threshold_ft'], axis = 1)

airports_df =airports_df.drop(['continent', 'iso_country', 'iso_region','municipality', 'gps_code','local_code', 'home_link',
                       'wikipedia_link', 'keywords'], axis=1)

# Join 1:  merge the runways and airport data.
df = pd.merge(airports_df, runways_df,how='left', left_on = "ident", right_on = "airport_ident")
df.head()
# Derive 'number_of_runways' Feature Engineering
df['number_of_runways'] = df.groupby('ident')['id_y'].transform('count')
df.isna().sum()
df.shape
df.drop(['id_x','id_y','surface'], axis=1, inplace=True)
# 2. Delete rows where 'Type' == 'closed'
df = df[df['type'] != 'closed']

# 3. Delete rows with null 'iata_code'
df = df[df['iata_code'].notnull()]
#  drop nulls in certain columns:
df['elevation_ft'] = df['elevation_ft'].fillna(0)  # Replace nulls in 'elevation_ft' with 0
df['length_ft'] = df['length_ft'].fillna(df['length_ft'].mean())  # Impute with mean
df['width_ft'] = df['width_ft'].fillna(df['length_ft'].mean())  # Impute with mean
# Impute nulls in 'lighted' column with the most frequent value
df['lighted'] = df['lighted'].fillna(df['lighted'].mode()[0])
# Now drop the nulls
df.dropna(inplace=True)
df.shape
```

```
df.info()
df['iata_code'].nunique()
df.drop_duplicates(subset='iata_code', inplace=True)

#Join 2: join Airlnes on df airportfrm<>iata code
df.reset_index(inplace=True) # rest indices
airlines_df.reset_index(inplace=True)
airlines_df.info()
df_airrun=df.copy()
df = airlines_df.merge(df, how='left', left_on='AirportFrom', right_on='iata_code')
df.shape
df.columns

#merged_df.rename(columns={'id': 'airport_id', 'type': 'airport_type'}, inplace=True)  # Rename for clarity
df=df.drop(['index_x', 'id', 'Flight',
       'index_y', 'ident',  'name', 'latitude_deg', 'longitude_deg',
        'scheduled_service',  'airport_ref',
       'airport_ident'  ],axis=1)
# make a temp df
df_airrun.columns
df_airrun=df_airrun.drop(['index', 'ident', 'type', 'name', 'latitude_deg', 'longitude_deg',
        'scheduled_service',  'airport_ref',
         'closed',
       ],axis=1)
df_airrun.drop_duplicates(subset='iata_code', inplace=True)
df_airrun.isna().sum()
df_airrun.info()
# Join 3: airports_df <-> airlines_df (on arrival airport)
df = df.merge(df_airrun, how='left', left_on='AirportTo', right_on='iata_code', suffixes=('_from', '_to'))
df.isna().sum()
# Now drop the nulls
df.dropna(inplace=True)
df.info()
df.columns
df=df.drop(['closed'],axis=1)
# Upload to excel as pgm crashing
df.to_csv(file_path + 'df_output_air.csv')


# Columns to convert to integers
float_to_int_cols = ['DayOfWeek', 'Time', 'Length',
       'Delay', 'elevation_ft_from',
       'length_ft_from', 'width_ft_from', 'lighted_from',
       'number_of_runways_from', 'elevation_ft_to',
        'length_ft_to', 'width_ft_to', 'lighted_to',
       'number_of_runways_to']

# Columns to convert to strings
obj_to_str_cols = ['Airline', 'AirportFrom', 'AirportTo',  'iata_code_from',
        'iata_code_to']

# Apply transformations
for col in float_to_int_cols:
    df[col] = df[col].astype(int)

for col in obj_to_str_cols:
    df[col] = df[col].astype(str)
# Check Data
df.isnull().sum()
df.info()
df.head()
df.shape
df['Airline'].unique()
# Assuming your DataFrame is named "df"
count_co_records = df[df['Airline'] == 'CO'].shape[0]
print(f"There are {count_co_records} records with the 'Airline' value as 'CO'.")

```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 518556 entries, 0 to 518555
Data columns (total 9 columns):
 #   Column       Non-Null Count   Dtype
---  ------       --------------   -----
```

```
 0   id           518556 non-null  int64
 1   Airline      518556 non-null  object
 2   Flight       518556 non-null  int64
 3   AirportFrom  518556 non-null  object
 4   AirportTo    518556 non-null  object
 5   DayOfWeek    518556 non-null  int64
 6   Time         518556 non-null  int64
 7   Length       518556 non-null  int64
 8   Delay        518556 non-null  int64
dtypes: int64(6), object(3)
memory usage: 35.6+ MB
None
(518556, 9)
   id Airline  Flight AirportFrom AirportTo  DayOfWeek  Time  Length  Delay
0   1      CO     269         SFO       IAH          3    15     205      1
1   2      US    1558         PHX       CLT          3    15     222      1
2   3      AA    2400         LAX       DFW          3    20     165      1
3   4      AA    2466         SFO       DFW          3    20     195      1
4   5      AS     108         ANC       SEA          3    30     202      0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 73805 entries, 0 to 73804
Data columns (total 18 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   id                 73805 non-null  int64
 1   ident              73805 non-null  object
 2   type               73805 non-null  object
 3   name               73805 non-null  object
 4   latitude_deg       73805 non-null  float64
 5   longitude_deg      73805 non-null  float64
 6   elevation_ft       59683 non-null  float64
 7   continent          38086 non-null  object
 8   iso_country        73546 non-null  object
 9   iso_region         73805 non-null  object
 10  municipality       68739 non-null  object
 11  scheduled_service  73805 non-null  object
 12  gps_code           42996 non-null  object
 13  iata_code          9160 non-null   object
 14  local_code         32975 non-null  object
 15  home_link          3492 non-null   object
 16  wikipedia_link     10705 non-null  object
 17  keywords           13951 non-null  object
dtypes: float64(3), int64(1), object(14)
memory usage: 10.1+ MB
None
(73805, 18)
       id ident           type                          name  latitude_deg  \
0    4983  NTGA  medium_airport                   Anaa Airport    -17.352600
1   27342  YARY   small_airport                Arrabury Airport    -26.696390
2    3176  HEAR  medium_airport  El Arish International Airport     31.078565
3  324756   AAD   small_airport                   Adado Airport      6.095802
4    2060  DABB  medium_airport      Annaba Rabah Bitat Airport     36.826781
```

Double-click (or enter) to edit

Web scraping

Airline experience feature engg and extraction using beautiful Soup fir web scrapping.

```python
import requests
from bs4 import BeautifulSoup

def get_airline_experience(airline_name, url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find all relevant tables
    info_tables = soup.find_all('table', class_='wikitable')

    if info_tables:
        for table in info_tables:
            # Search for the airline row within each table
            for row in table.find_all('tr'):
                cells = row.find_all('td')
                if cells and airline_name in cells[2].text: #index 2 holds IATA code
                    founded_year_str = cells[6].text.strip()  # this index holds Founded open and check url

                    try:
                        founded_year = int(founded_year_str)
                        return 2024 - founded_year  # Return as soon as a match is found
                    except ValueError:
                        print('Founding year format error for:', airline_name)

        # If no match is found in any of the tables:
        print('Error: Airline not found','   ', airline_name)
        return None
    else:
        print('Error: No info tables found')
        return None

# Set URL of website
airline_list_url = "https://en.wikipedia.org/wiki/List_of_airlines_of_the_United_States"
airline_experience = {}

# Assuming you have a DataFrame called 'airlines_df' with an 'airline_name' column
for airline in airlines_df['Airline'].unique():
    experience = get_airline_experience(airline, airline_list_url)
    airline_experience[airline] = experience

# Add airline experience to the merged dataset
df['airline_experience'] = df['Airline'].map(airline_experience)
# Checking and testing
df.tail()
df['airline_experience'].isna().sum()
# Find out airlines for which data na in website
grouped_df =df[df['airline_experience'].isna()].groupby('Airline').size().reset_index(name='Count')

# Print the websites where data NA
print(grouped_df)

# Assigning values manually as deleting these will be huge loss of data
df.loc[df['Airline'] == 'CO', 'airline_experience'] =87
df.loc[df['Airline'] == 'US', 'airline_experience'] = 87
df.loc[df['Airline'] == 'EV', 'airline_experience'] = 37
df.isna().sum()
df.info()
df.shape
# df=merged_df.copy() #This copies df the original one is unchanged
df.dropna(inplace=True)
df.isnull().sum()
df.shape
df.info()
```

```
Error: Airline not found     CO
Error: Airline not found     US
Error: Airline not found     EV
  Airline  Count
0      CO  21118
1      EV  27983
2      US  34500
<class 'pandas.core.frame.DataFrame'>
Int64Index: 518494 entries, 0 to 518493
Data columns (total 25 columns):
 #   Column                 Non-Null Count   Dtype
```

```
---  ------                   --------------   -----
 0   Airline                  518494 non-null  object
 1   AirportFrom              518494 non-null  object
 2   AirportTo                518494 non-null  object
 3   DayOfWeek                518494 non-null  int64
 4   Time                     518494 non-null  int64
 5   Length                   518494 non-null  int64
 6   Delay                    518494 non-null  int64
 7   type_from                518494 non-null  object
 8   elevation_ft_from        518494 non-null  int64
 9   iata_code_from           518494 non-null  object
 10  length_ft_from           518494 non-null  int64
 11  width_ft_from            518494 non-null  int64
 12  lighted_from             518494 non-null  int64
 13  number_of_runways_from   518494 non-null  int64
 14  elevation_ft_to          518494 non-null  int64
 15  iata_code_to             518494 non-null  object
 16  airport_ident            518494 non-null  object
 17  length_ft_to             518494 non-null  int64
 18  width_ft_to              518494 non-null  int64
 19  lighted_to               518494 non-null  int64
 20  number_of_runways_to     518494 non-null  int64
 21  type_to                  518494 non-null  object
 22  elevation_ft             518494 non-null  float64
 23  iata_code                518494 non-null  object
 24  airline_experience       518494 non-null  float64
dtypes: float64(2), int64(14), object(9)
memory usage: 102.9+ MB
<class 'pandas.core.frame.DataFrame'>
Int64Index: 518494 entries, 0 to 518493
Data columns (total 25 columns):
 #   Column                   Non-Null Count   Dtype
---  ------                   --------------   -----
 0   Airline                  518494 non-null  object
 1   AirportFrom              518494 non-null  object
 2   AirportTo                518494 non-null  object
 3   DayOfWeek                518494 non-null  int64
 4   Time                     518494 non-null  int64
 5   Length                   518494 non-null  int64
 6   Delay                    518494 non-null  int64
 7   type_from                518494 non-null  object
 8   elevation_ft_from        518494 non-null  int64
 9   iata_code_from           518494 non-null  object
 10  length_ft_from           518494 non-null  int64
 11  width_ft_from            518494 non-null  int64
 12  lighted_from             518494 non-null  int64
 13  number_of_runways_from   518494 non-null  int64
```

Passenger data departure airport from url

```
# Define function to process passenger info
def get_airport_tfc(airport_name, url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')

    # Find all relevant tables
    info_tables = soup.find_all('table',class_='wikitable')

    if info_tables:
        for table in info_tables:
            # Search for the airline row within each table
            for row in table.find_all('tr'):
                cells = row.find_all('td')
                if cells and airport_name in cells[2].text: #index 2 holds IATA code
                    passenger_tfc_str = cells[6].text.strip()  # this index holds Founded open and check url

                    try:
                        passenger_tfc = int(passenger_tfc_str.replace(',', ''))
                        return passenger_tfc  # Return as soon as a match is found
                    except ValueError:
                        print('Passenger tfc format error for:', airport_name)

        # If no match is found in any of the tables:
        print('Error: Passenger data not found','   ', airport_name)
        return None
    else:
        print('Error: No info tables found')
        return None

# Set URL of website
airline_list_url = "https://en.wikipedia.org/wiki/List_of_the_busiest_airports_in_the_United_States"
airport_tfc = {}

# Assuming you have a DataFrame called 'airlines_df' with an 'airline_name' column
for airport in df['AirportFrom'].unique():
    tfc = get_airport_tfc(airport, airline_list_url)
    airport_tfc[airport] = tfc

notnull_count = sum(1 for value in airport_tfc.values() if value is not None)
list_notnull = [key for key, value in airport_tfc.items() if value is not None]
# Add airline passenger to the merged dataset departure
df['passenger_tfc_from'] = df['AirportFrom'].map(airport_tfc)
df.tail()
df['passenger_tfc_from'].isnull().sum()
df['passenger_tfc_from'].unique()
# Find out airlines for which data na in website
grouped_passenger_df = df[df['passenger_tfc_from'].isna()].groupby('AirportFrom').size().reset_index(name='Count')
#df['passenger_tfc_from'].fillna(df['passenger_tfc_from'].min(), inplace=True)
# Print the websites where data NA
print(grouped_passenger_df)

# Consider dropping- data too large so will make cat variable
#df.dropna(inplace=True)
df.isnull().sum()
# I have 2 lakh nulls in the passenger field. As no data from website
# Will make a categorical variable hub to classify airport to large, medium, small
# Categorize airports as large/medium hubs based on passenger traffic
df['hub_type_from'] = np.where(df['passenger_tfc_from'] >= 8000000, 'large',
                               np.where(df['passenger_tfc_from'] >= 2000000, 'medium', 'small'))

# Troble shoot immute later
#df['passenger_tfc_from'].fillna(df['passenger_tfc_from'].min(), inplace=True)
# Find outhub summary
grouped_hub_df = df[df['hub_type_from'].isna()].groupby('AirportFrom').size().reset_index(name='Count')

# Print the websites where data NA
print(grouped_hub_df)
# Count the occurrences of each hub type
hub_type_counts_from = df['hub_type_from'].value_counts()

# Print the counts
print(hub_type_counts_from)
df['hub_type_from'].unique()
```

```
74 df.shape
75 df.info()
76 df['Airline'].dtype
77 df['Delay'].dtype
78 df.head()
79 print(df['Delay'].unique())
80 df['Delay'] = df['Delay'].astype(bool)
```

```
    Error: Passenger data not found    FAI
    Error: Passenger data not found    BQN
    Error: Passenger data not found    PSE
    Error: Passenger data not found    BIS
    Error: Passenger data not found    IYK
    Error: Passenger data not found    GFK
    Error: Passenger data not found    GSO
    Error: Passenger data not found    LMT
    Error: Passenger data not found    DLH
    Error: Passenger data not found    FAR
    Error: Passenger data not found    MFE
    Error: Passenger data not found    VPS
    Error: Passenger data not found    MAF
    Error: Passenger data not found    LWS
    Error: Passenger data not found    RST
    Error: Passenger data not found    ALB
    Error: Passenger data not found    DSM
    Error: Passenger data not found    MSN
    Error: Passenger data not found    PNS
    Error: Passenger data not found    BHM
    Error: Passenger data not found    LIT
    Error: Passenger data not found    SAV
    Error: Passenger data not found    ICT
    Error: Passenger data not found    ECP
    Error: Passenger data not found    DHN
    Error: Passenger data not found    MGM
    Error: Passenger data not found    CAE
    Error: Passenger data not found    PWM
    Error: Passenger data not found    ACV
    Error: Passenger data not found    EKO
    Error: Passenger data not found    RIC
    Error: Passenger data not found    BTR
    Error: Passenger data not found    HRL
    Error: Passenger data not found    MYR
    Error: Passenger data not found    TUS
    Error: Passenger data not found    SBN
    Error: Passenger data not found    CAK
    Error: Passenger data not found    TVC
    Error: Passenger data not found    DAY
    Error: Passenger data not found    MFR
    Error: Passenger data not found    BTV
    Error: Passenger data not found    TLH
    Error: Passenger data not found    TYS
    Error: Passenger data not found    CHA
    Error: Passenger data not found    LRD
    Error: Passenger data not found    BRO
    Error: Passenger data not found    CRP
    Error: Passenger data not found    LAN
    Error: Passenger data not found    PVD
    Error: Passenger data not found    FWA
    Error: Passenger data not found    OKC
    Error: Passenger data not found    ORF
    Error: Passenger data not found    AEX
    Error: Passenger data not found    SYR
    Error: Passenger data not found    SHV
    Error: Passenger data not found    VLD
    Error: Passenger data not found    FAT
    Error: Passenger data not found    BZN
```

passenger data arr airport

```
# Add airline passenger to the merged dataset departure
df['passenger_tfc_to'] = df['AirportTo'].map(airport_tfc)
df.tail()
df['passenger_tfc_to'].isnull().sum()
df['passenger_tfc_to'].unique()
# Find out airlines for which data na in website
grouped_passenger_to_df = df[df['passenger_tfc_to'].isna()].groupby('AirportTo').size().reset_index(name='Count')

# Print the websites where data NA
print(grouped_passenger_to_df)

# Consider dropping- data too large so will make cat variable
#df.dropna(inplace=True)
df.isnull().sum()
# I have 2 lakh nulls in the passenger field. As no data from website
# Will make a categorical variable hub to classify airport to large, medium, small
# Categorize airports as large/medium hubs based on passenger traffic
df['hub_type_to'] = np.where(df['passenger_tfc_to'] >= 8000000, 'large',
                             np.where(df['passenger_tfc_to'] >= 2000000, 'medium', 'small'))
# Trouble shoot will immute later
#df['passenger_tfc_to'].fillna(df['passenger_tfc_to'].min(), inplace=True)
# Find outhub summary
grouped_hub_to_df = df[df['hub_type_to'].isna()].groupby('AirportTo').size().reset_index(name='Count')

# Print the websites where data NA
print(grouped_hub_to_df)
# Count the occurrences of each hub type
hub_type_counts_to = df['hub_type_to'].value_counts()

# Print the counts
print(hub_type_counts_to)
```

```
    AirportTo  Count
0         ABE    257
1         ABI    211
2         ABR      2
3         ABY     87
4         ACT     59
..        ...    ...
221       VPS    633
222       WRG     59
223       XNA   1123
224       YAK     57
225       YUM    338

[226 rows x 2 columns]
Empty DataFrame
Columns: [AirportTo, Count]
Index: []
large     334086
medium     99490
small      84918
Name: hub_type_to, dtype: int64
```

Upload webscrapped file

This is to keeo copy of webscraped df

```
# Upload to csv for continuing proj from here
# Only passenger from and to field has na values
df.to_csv(file_path + 'df_wescraped_air.csv')
print(df.info())
df.shape
# The shape of the df is similar to Airline table. If majorly different then the joins are incorrect.
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 518494 entries, 0 to 518493
Data columns (total 29 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   Airline                518494 non-null  object
 1   AirportFrom            518494 non-null  object
 2   AirportTo              518494 non-null  object
 3   DayOfWeek              518494 non-null  int64
 4   Time                   518494 non-null  int64
```

```
 5   Length                  518494 non-null  int64
 6   Delay                   518494 non-null  bool
 7   type_from               518494 non-null  object
 8   elevation_ft_from       518494 non-null  int64
 9   iata_code_from          518494 non-null  object
 10  length_ft_from          518494 non-null  int64
 11  width_ft_from           518494 non-null  int64
 12  lighted_from            518494 non-null  int64
 13  number_of_runways_from  518494 non-null  int64
 14  elevation_ft_to         518494 non-null  int64
 15  iata_code_to            518494 non-null  object
 16  airport_ident           518494 non-null  object
 17  length_ft_to            518494 non-null  int64
 18  width_ft_to             518494 non-null  int64
 19  lighted_to              518494 non-null  int64
 20  number_of_runways_to    518494 non-null  int64
 21  type_to                 518494 non-null  object
 22  elevation_ft            518494 non-null  float64
 23  iata_code               518494 non-null  object
 24  airline_experience      518494 non-null  float64
 25  passenger_tfc_from      433524 non-null  float64
 26  hub_type_from           518494 non-null  object
 27  passenger_tfc_to        433576 non-null  float64
 28  hub_type_to             518494 non-null  object
dtypes: bool(1), float64(4), int64(13), object(11)
memory usage: 115.2+ MB
None
```

Load Webscraped file from Gdrive. Midcourse start as Web scraping is time consuming.

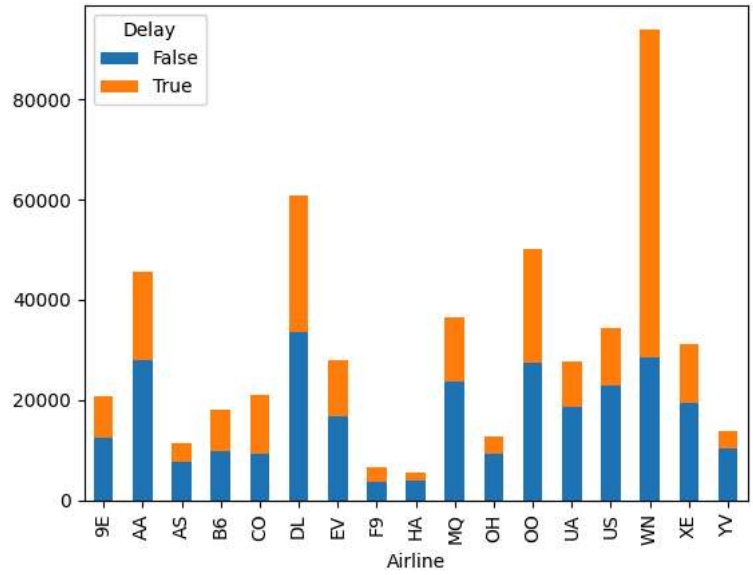
```
1 df= None
2
3 # read file into df
4 df=pd.read_csv(file_path+'df_wescraped_air.csv')
5 print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 518494 entries, 0 to 518493
Data columns (total 30 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Unnamed: 0              518494 non-null  int64
 1   Airline                 518494 non-null  object
 2   AirportFrom             518494 non-null  object
 3   AirportTo               518494 non-null  object
 4   DayOfWeek               518494 non-null  int64
 5   Time                    518494 non-null  int64
 6   Length                  518494 non-null  int64
 7   Delay                   518494 non-null  bool
 8   type_from               518494 non-null  object
 9   elevation_ft_from       518494 non-null  int64
 10  iata_code_from          518494 non-null  object
 11  length_ft_from          518494 non-null  int64
 12  width_ft_from           518494 non-null  int64
 13  lighted_from            518494 non-null  int64
 14  number_of_runways_from  518494 non-null  int64
 15  elevation_ft_to         518494 non-null  int64
 16  iata_code_to            518494 non-null  object
 17  airport_ident           518494 non-null  object
 18  length_ft_to            518494 non-null  int64
 19  width_ft_to             518494 non-null  int64
 20  lighted_to              518494 non-null  int64
 21  number_of_runways_to    518494 non-null  int64
 22  type_to                 518494 non-null  object
 23  elevation_ft            518494 non-null  float64
 24  iata_code               518494 non-null  object
 25  airline_experience      518494 non-null  float64
 26  passenger_tfc_from      433524 non-null  float64
 27  hub_type_from           518494 non-null  object
 28  passenger_tfc_to        433576 non-null  float64
 29  hub_type_to             518494 non-null  object
dtypes: bool(1), float64(4), int64(14), object(11)
memory usage: 115.2+ MB
None
```

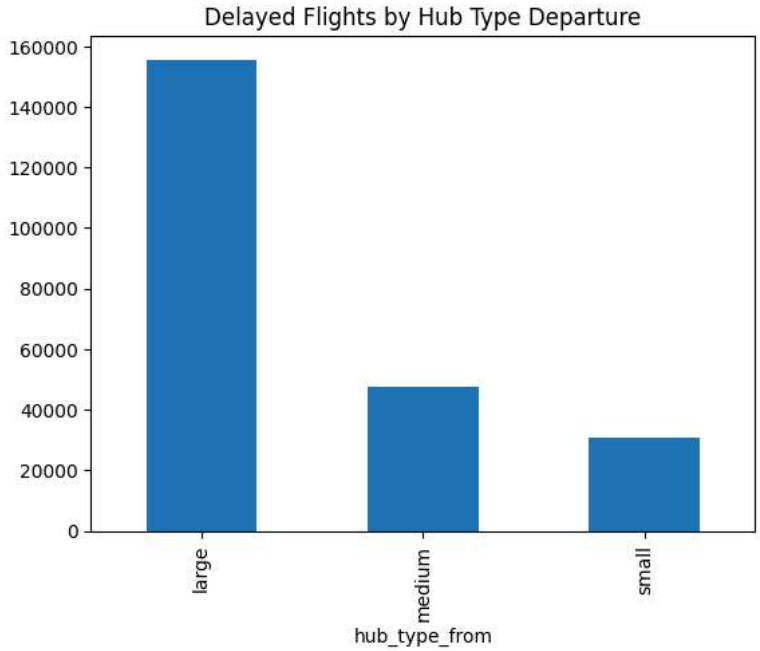
Visualisation

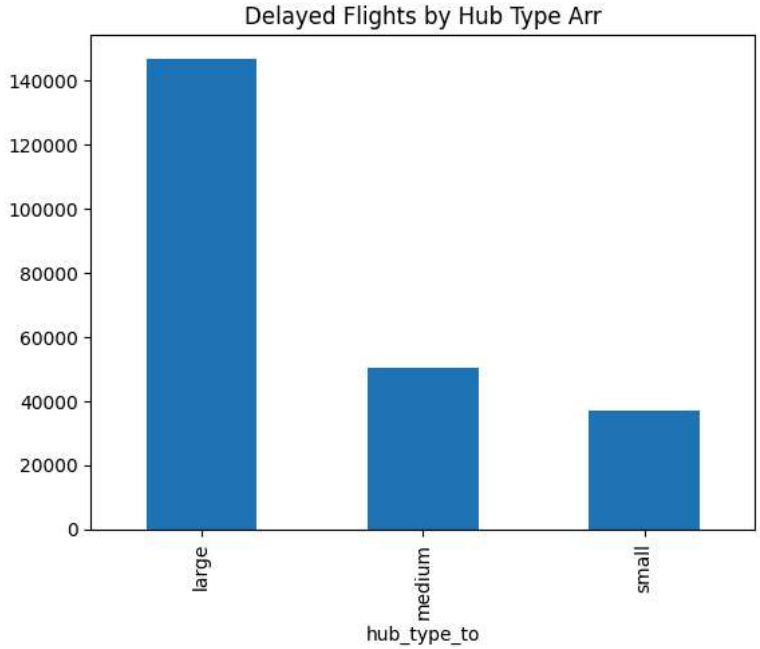
```

# Reset the index to ensure it starts from 0 and is consecutive
df.reset_index(drop=True, inplace=True)

# Add a new column named 'ID'
df['ID'] = df.index + 1  # Index starts from 0, so we add 1

df_pivot = pd.pivot_table(df, index='Airline', columns='Delay',values='ID',  aggfunc='count').fillna(0)
plt.figure(figsize=(10,7))
df_pivot.plot.bar(stacked=True)
plt.show()

```

```
<Figure size 1000x700 with 0 Axes>
```

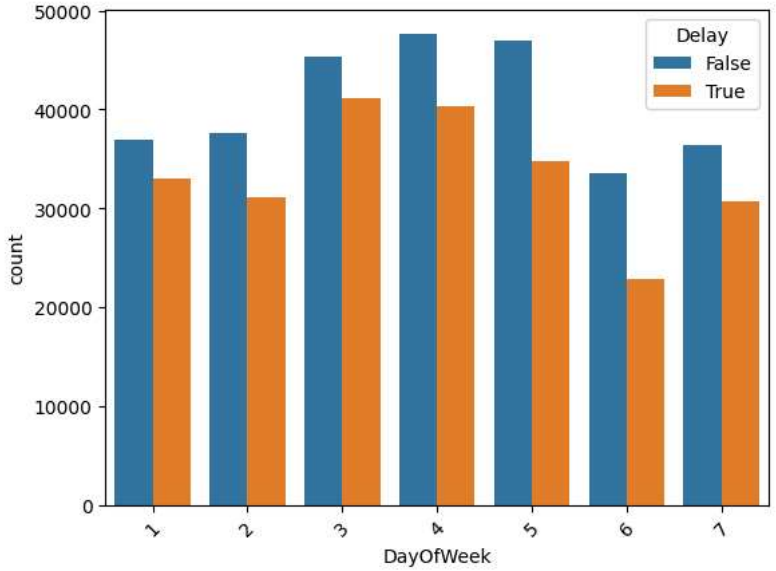



```
# Visualization: Delayed flights by hub type departure
delayed_by_hub_from = df[df['Delay'] == True].groupby('hub_type_from').size().plot(kind='bar')
plt.title('Delayed Flights by Hub Type Departure')
plt.show()
#Plot dealy vs Arr airfd hub type
delayed_by_hub_to = df[df['Delay'] == True].groupby('hub_type_to').size().plot(kind='bar')
plt.title('Delayed Flights by Hub Type Arr')
plt.show()
```



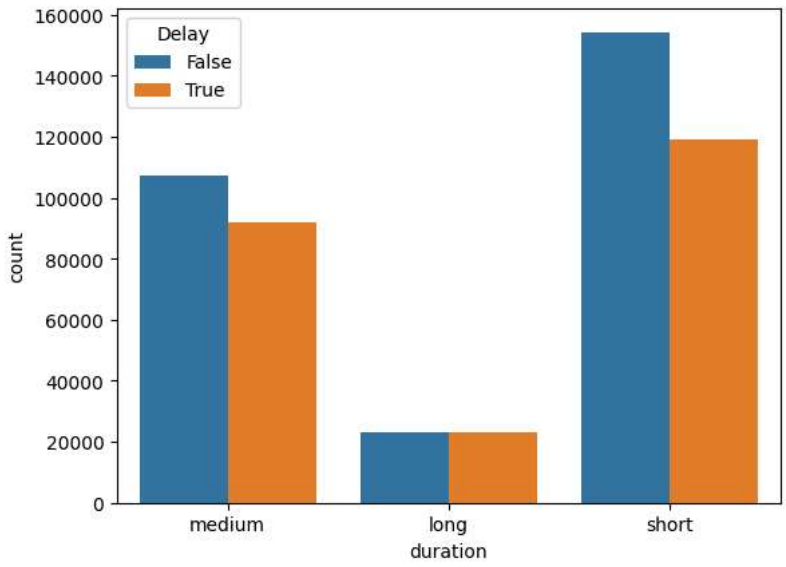



```
# Create a count plot of DayOfWeek vs Delay
sns.countplot(
    x = 'DayOfWeek',
    hue = 'Delay',
    data = df
)

# Rotate x labels to prevent overlapping
plt.xticks(rotation=45)
plt.show()
```

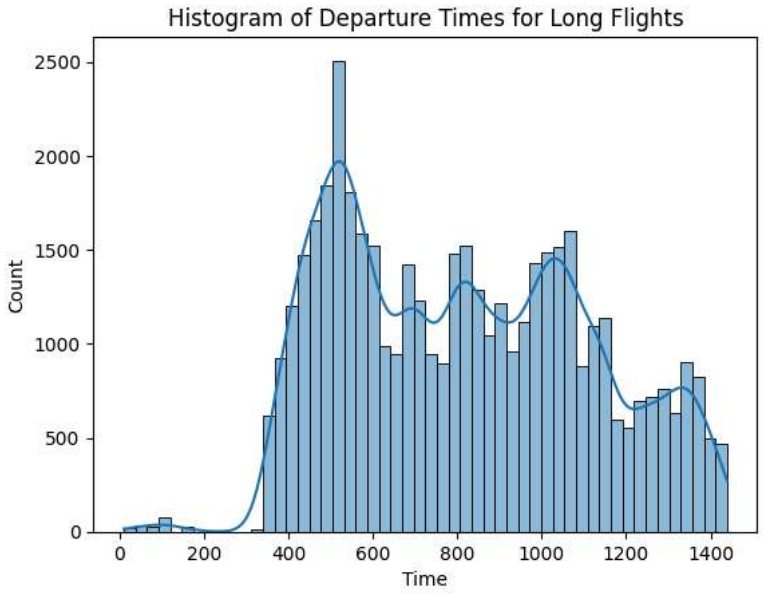


Feature engineering

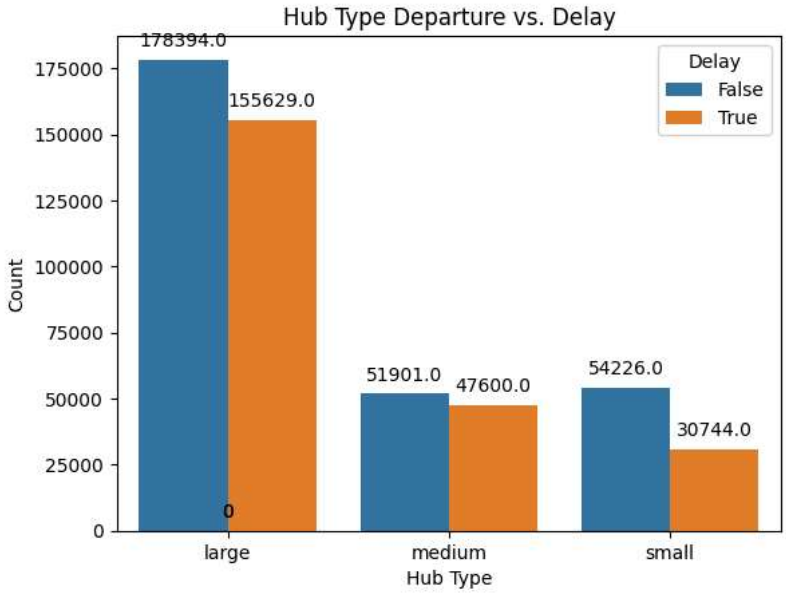
```
 1 # Create the 'duration' field
 2 def categorize_duration(row):
 3     if row['Length'] < 120:
 4         return 'short'
 5     elif 120 <= row['Length'] < 240:
 6         return 'medium'
 7     else:
 8         return 'long'
 9
10 df['duration'] = df.apply(categorize_duration, axis=1)
11
12 # Plot duration vs. delay
13 sns.countplot(x='duration', hue='Delay', data=df)
14 plt.show()
15
16
```



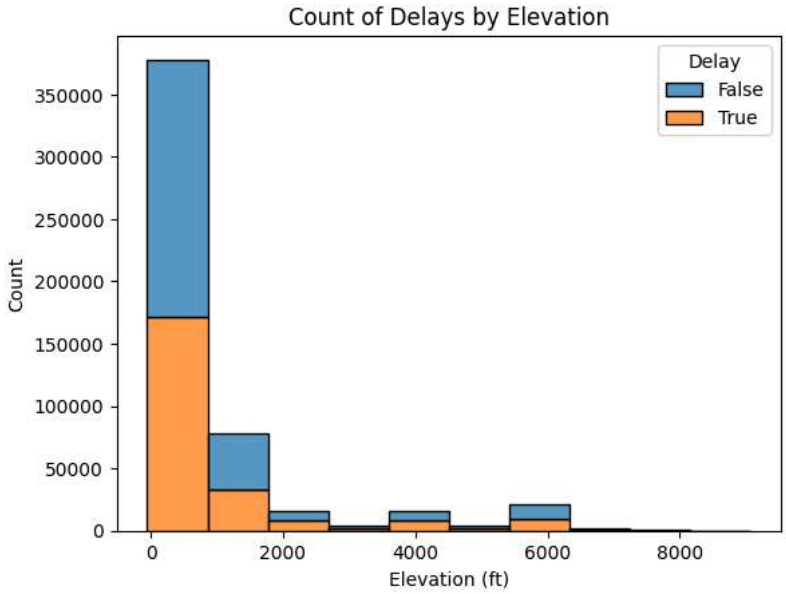
```
# Filter for long duration flights
long_flights = df[df['duration'] == 'long']

# Create the histogram
sns.histplot(long_flights['Time'], kde=True)  # kde=True adds a density curve
plt.xlabel('Time')
plt.ylabel('Count')
plt.title('Histogram of Departure Times for Long Flights')
plt.show()
```

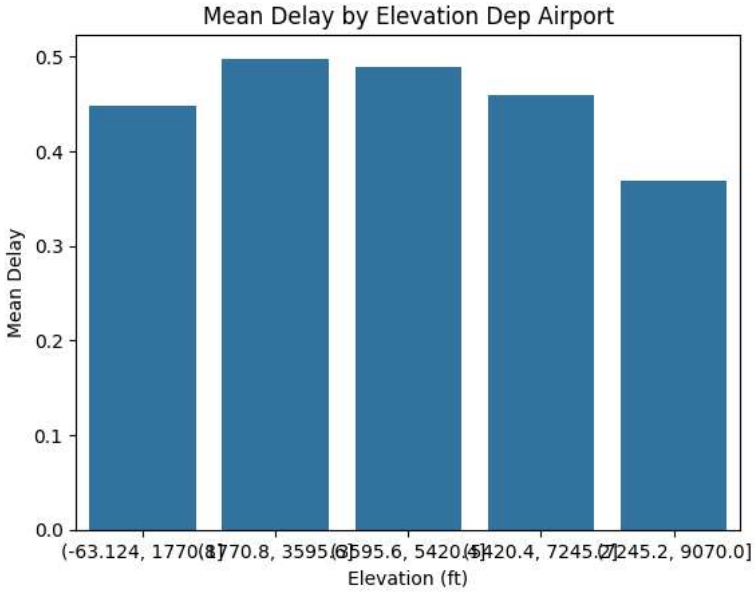



```
# Plot hub vs. delay
sns.countplot(x='hub_type_from', hue='Delay', data=df)
# Get the current axes object
ax = plt.gca()
# Add value labels
for p in ax.patches:
    height = p.get_height()
    ax.annotate(f'{height}',
                xy=(p.get_x() + p.get_width() / 2, height),
                xytext=(0, 5),  # Offset slightly above the bar
                textcoords='offset points',
                ha='center', va='bottom')
# Customize the plot
plt.xlabel('Hub Type')
plt.ylabel('Count')
plt.title('Hub Type Departure vs. Delay')
plt.legend(title='Delay', loc='upper right')
plt.show()

```



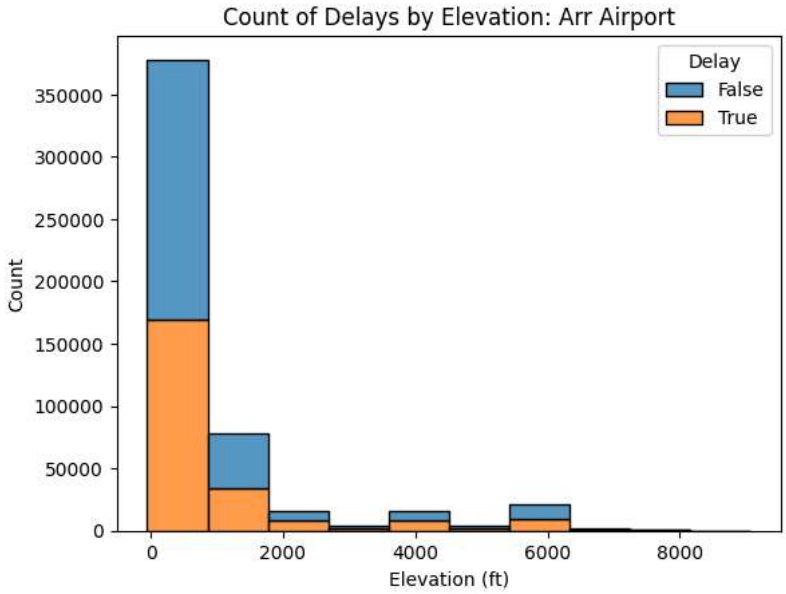

```
# Define the number of bins (adjust as needed)
num_bins = 10

# Create the histogram
sns.histplot(
    df,
    x = 'elevation_ft_from',
    hue = 'Delay',  # Separate histograms for delays vs non-delays
    bins = num_bins,
    multiple = 'stack'  # Stack the histograms on top of each other
)
plt.xlabel('Elevation (ft)')
plt.ylabel('Count')
plt.title('Count of Delays by Elevation')
plt.show()

# Calculate grouped means
# Define the number of bins (adjust as needed)
num_bins = 5

df_grouped = df.groupby(pd.cut(df['elevation_ft_from'], bins=num_bins))['Delay'].mean()

# Create the bar plot
sns.barplot(
    x = df_grouped.index.astype(str),  # Convert bin labels to string for plotting
    y = df_grouped.values
)
plt.xlabel('Elevation (ft)')
plt.ylabel('Mean Delay')
plt.title('Mean Delay by Elevation Dep Airport')
plt.figure(figsize=(12, 20))
plt.show()
# Hypothesis testing (example: effect of airport elevation on delays)
from scipy.stats import ttest_ind

high_elev =df[df['elevation_ft_from'] > 5000]
low_elev = df[df['elevation_ft_from'] <= 5000]

t_stat, p_value = ttest_ind(high_elev['Delay'], low_elev['Delay'], equal_var=False)

print("t-statistic:", t_stat)
print("p-value:", p_value)

```

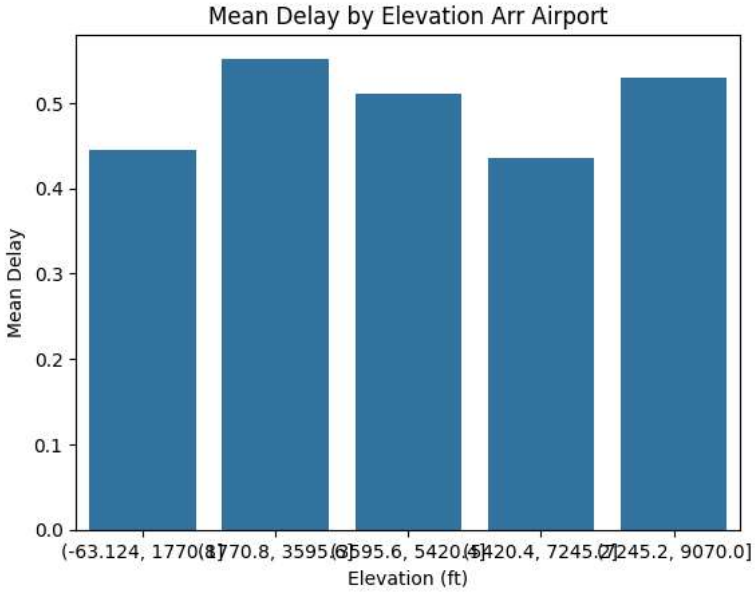





```
<Figure size 1200x2000 with 0 Axes>
t-statistic: 2.415435752905638
p-value: 0.015722462914667622
```

```
 1
 2 # Define the number of bins (adjust as needed)
 3 num_bins = 10
 4
 5 # Create the histogram
 6 sns.histplot(
 7     df,
 8     x = 'elevation_ft_to',
 9     hue = 'Delay',  # Separate histograms for delays vs non-delays
10     bins = num_bins,
11     multiple = 'stack'  # Stack the histograms on top of each other
12 )
13 plt.xlabel('Elevation (ft)')
14 plt.ylabel('Count')
15 plt.title('Count of Delays by Elevation: Arr Airport')
16 plt.show()
17
18 # Calculate grouped means
19 # Define the number of bins (adjust as needed)
20 num_bins = 5
21
22 df_grouped = df.groupby(pd.cut(df['elevation_ft_to'], bins=num_bins))['Delay'].mean()
23
24 # Create the bar plot (using barplot for flexibility)
25 sns.barplot(
26     x = df_grouped.index.astype(str),  # Convert bin labels to string for plotting
27     y = df_grouped.values
28 )
29 plt.xlabel('Elevation (ft)')
30 plt.ylabel('Mean Delay')
31 plt.title('Mean Delay by Elevation Arr Airport')
32 plt.figure(figsize=(12, 20))
33 plt.show()
34 # Hypothesis testing (example: effect of airport elevation on delays)
35 from scipy.stats import ttest_ind
36
37 high_elev =df[df['elevation_ft_to'] > 5000]
38 low_elev = df[df['elevation_ft_to'] <= 5000]
39
40 t_stat, p_value = ttest_ind(high_elev['Delay'], low_elev['Delay'], equal_var=False)
41
42 print("t-statistic:", t_stat)
43 print("p-value:", p_value)
44
```



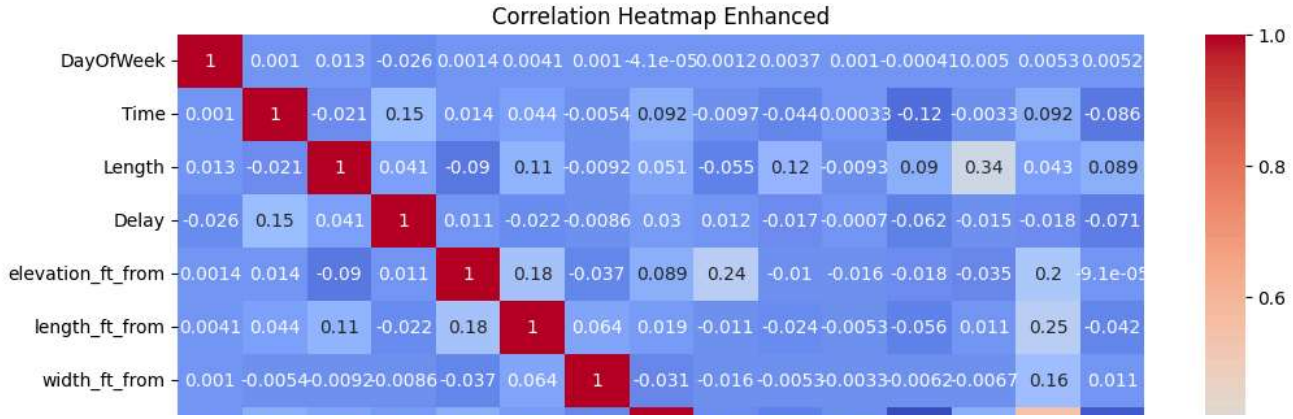



```
<Figure size 1200x2000 with 0 Axes>
t-statistic: 0.7498156315532386
p-value: 0.4533717588728631
```

Correlation

```
df.info()
df.columns
# Select relevant numerical features for correlation analysis
predictors = ['DayOfWeek', 'Time', 'Length',
       'Delay',  'elevation_ft_from',
       'length_ft_from', 'width_ft_from',
       'number_of_runways_from', 'elevation_ft_to',
       'length_ft_to', 'width_ft_to',
       'number_of_runways_to',
       'airline_experience', 'passenger_tfc_from',
       'passenger_tfc_to' ]
corr_matrix = df[predictors].corr()

# Heatmap visualization
import seaborn as sns
# Increase size
plt.figure(figsize=(12, 8))  # Adjust width and height as desired

# Select a colormap:
cmap = sns.color_palette("coolwarm", as_cmap=True)  # Try other palettes!

# Create the heatmap
sns.heatmap(corr_matrix, annot=True, cmap=cmap)
plt.title('Correlation Heatmap Enhanced')
plt.show()
# Convert the correlation matrix to a DataFrame (if it isn't already)
if not isinstance(corr_matrix, pd.DataFrame):
    corr_matrix = pd.DataFrame(corr_matrix)

# Export to Excel
corr_matrix.to_excel(file_path +'correlation_matrix.xlsx', index=True)
# 'index=True' will include the row and column labels
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 518494 entries, 0 to 518493
Data columns (total 32 columns):
 #   Column                  Non-Null Count   Dtype
---  ------                  --------------   -----
 0   Unnamed: 0              518494 non-null  int64
 1   Airline                 518494 non-null  object
 2   AirportFrom             518494 non-null  object
 3   AirportTo               518494 non-null  object
 4   DayOfWeek               518494 non-null  int64
 5   Time                    518494 non-null  int64
 6   Length                  518494 non-null  int64
 7   Delay                   518494 non-null  bool
 8   type_from               518494 non-null  object
 9   elevation_ft_from       518494 non-null  int64
 10  iata_code_from          518494 non-null  object
 11  length_ft_from          518494 non-null  int64
 12  width_ft_from           518494 non-null  int64
 13  lighted_from            518494 non-null  int64
 14  number_of_runways_from  518494 non-null  int64
 15  elevation_ft_to         518494 non-null  int64
 16  iata_code_to            518494 non-null  object
 17  airport_ident           518494 non-null  object
 18  length_ft_to            518494 non-null  int64
 19  width_ft_to             518494 non-null  int64
 20  lighted_to              518494 non-null  int64
 21  number_of_runways_to    518494 non-null  int64
 22  type_to                 518494 non-null  object
 23  elevation_ft            518494 non-null  float64
 24  iata_code               518494 non-null  object
 25  airline_experience      518494 non-null  float64
 26  passenger_tfc_from      433524 non-null  float64
 27  hub_type_from           518494 non-null  object
 28  passenger_tfc_to        433576 non-null  float64
 29  hub_type_to             518494 non-null  object
 30  ID                      518494 non-null  int64
 31  duration                518494 non-null  object
dtypes: bool(1), float64(4), int64(15), object(12)
memory usage: 123.1+ MB
```



Machine learning

```
 1 df.columns
 2 df['type_to'].unique()
 3 # Drop unnecessary columns from df
 4 df=df.drop(['passenger_tfc_from','passenger_tfc_to','elevation_ft','iata_code_from','iata_code_to','airport_ident'],axis=1)
 5 df=df.drop(['iata_code'],axis=1)
 6
 7 # Assuming 'AirportFrom' is already in the categorical_features list
 8 encoder = OrdinalEncoder()
 9 df['AirportFrom_encode'] = encoder.fit_transform(df[['AirportFrom']])
10 df['AirportFrom'].nunique()
11 df=df.drop(['AirportFrom_encode'],axis=1)
12
13 df.columns
14 df.shape
15
```

```
(518494, 25)
```

Label encode

I am encoding it out of the pipeline due to repeted errors

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()

df['Airline'] = le.fit_transform(df['Airline'])
df['AirportFrom'] = le.fit_transform(df['AirportFrom'])
df['AirportTo'] = le.fit_transform(df['AirportTo'])
df.head()
```

	Unnamed: 0	Airline	AirportFrom	AirportTo	DayOfWeek	Time	Length	Delay	type_from	elevation_ft_from	...	length_ft_to	width_ft_
0	0	4	251	133	3	15	205	True	large_airport	13	...	9000	1
1	1	13	215	59	3	15	222	True	large_airport	1135	...	7502	1
2	2	1	152	78	3	20	165	True	large_airport	125	...	9000	2
3	3	1	251	78	3	20	195	True	large_airport	13	...	9000	2
4	4	2	13	250	3	30	202	False	large_airport	152	...	9426	1

5 rows × 25 columns

Pipeline Model Fitting and testing

This is the main ML codeblock where ppl has been used to encode categorical data and model fitting is done in a loop. This is a very useful and resusable code 😍

```

from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OrdinalEncoder, StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split, StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_extraction import FeatureHasher
from sklearn.ensemble import RandomForestClassifier

# Assuming 'delay' is your target variable (binary: delayed or not)
X = df[['Airline', 'AirportFrom', 'AirportTo', 'DayOfWeek',
       'Time', 'Length', 'type_from', 'elevation_ft_from',
       'length_ft_from', 'width_ft_from', 'lighted_from',
       'number_of_runways_from', 'elevation_ft_to', 'length_ft_to',
       'width_ft_to', 'lighted_to', 'number_of_runways_to', 'type_to',
       'airline_experience', 'hub_type_from', 'hub_type_to']]
y = df['Delay']



# Identify Feature Types
ordinal_features = ['type_from','type_to','hub_type_from', 'hub_type_to']  # Replace with your actual ordinal feature
numerical_features = ['DayOfWeek','Time', 'Length', 'elevation_ft_from','length_ft_from', 'width_ft_from',
       'number_of_runways_from', 'elevation_ft_to', 'length_ft_to',
       'width_ft_to', 'number_of_runways_to',
       'airline_experience',]  # Your numerical features
airport_type_order = ['small_airport', 'medium_airport', 'large_airport']  # Order for 'type_from', 'type_to'
hub_type_order=['large', 'medium', 'small']
ordinal_order=[airport_type_order,airport_type_order,hub_type_order,hub_type_order]


# Build the Pipeline
preprocessor = ColumnTransformer(
    transformers=[

        ('oe_type', OrdinalEncoder(categories='auto'), ordinal_features),
        ('scaler', StandardScaler(), numerical_features)
    ],
    remainder='passthrough'
)



# Split into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)
X_train_transformed = preprocessor.fit_transform(X_train)
X_test_transformed = preprocessor.transform(X_test)

# Model Building
models = {
'Logistic Regression': LogisticRegression(solver='liblinear'),
    'Decision Tree': DecisionTreeClassifier(max_depth=5),  # Example depth control
    # Hyper tuning parameter set
    'Gradient Boosting': GradientBoostingClassifier(max_depth=8, n_estimators=100),
    # tuning hyper parameters based on grid search
    'Random Forest Classifier' : RandomForestClassifier(max_depth= 8, min_samples_split= 10, n_estimators=200)
}

# Stratified K-Fold for voting
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

for name, model in models.items():
    accuracy_scores = []
    for train_index, test_index in skf.split(X_train_transformed, y_train):
        X_train_fold, y_train_fold = X_train_transformed[train_index], y_train.iloc[train_index]
        model.fit(X_train_fold, y_train_fold)
        y_pred = model.predict(X_train_transformed[test_index])
        accuracy = accuracy_score(y_train.iloc[test_index], y_pred)
        accuracy_scores.append(accuracy)
```

```
74     print(f"{name} Average Accuracy: {np.mean(accuracy_scores):.3f}")
```

Plot the gradient booster

```
    Decision Tree Average Accuracy: 0.634
```

```
 1 import matplotlib.pyplot as plt
 2
 3 model = models['Gradient Boosting']
 4 feature_importances = model.feature_importances_
 5 feature_names = X.columns  # Or update with transformed column names
 6 sorted_indices = np.argsort(feature_importances)[::-1]  # Sort importances
 7
 8 plt.barh(feature_names[sorted_indices], feature_importances[sorted_indices])
 9 plt.xlabel("Feature Importance")
10 plt.title("Gradient Boosting Feature Importance")
11 plt.show()
12
13 #Print top features
14 print('Top 10 Features (Gradient Boosting):')
15 for i in sorted_indices[:10]:  # Print the top 10
16     print(f'{feature_names[i]}: {feature_importances[i]:.3f}')
```