Introduction to Data Science (S2-22_DSECLZG532)-ASSIGNMENT

Group No 80

Group Member Names:

1. **LOSSHIKA S N** (2023DC04228)
2. ~~NATARAJAN R~~
3. ~~NEESHU PRAVEEN SRIVASTAVA~~
4. ~~VIPIN LUTHRA~~

NOTE: No contributions made by any of the team members (2-4).

100% Contribution individually done by LOSSHIKA.

1. Business Understanding

Students are expected to identify an analytical problem of your choice. You have to detail the Business Understanding part of your problem under this heading which basically addresses the following questions.

1. What is the business problem that you are trying to solve?
2. What data do you need to answer the above problem?
3. What are the different sources of data?
4. What kind of analytics task are you performing?

Score: 1 Mark in total (0.25 mark each)

```
--------------Type the answers below this line--------------

1. The business problem at hand is predicting whether a flight will be delayed or
not, using a dataset of flight delays and cancellations from 2019 to 2023. Delays
in flight operations can result in significant financial costs for airlines,
operational challenges for airports, and inconvenience for passengers.
   The goal is to develop a model that can accurately classify whether a flight
will be delayed, enabling airlines to take proactive measures, such as adjusting
schedules, reallocating resources, or communicating effectively with passengers.
Accurate delay prediction can also enhance customer satisfaction and improve
airport operations, ultimately reducing the negative impact of delays on both the
airline's profitability and the passenger experience.

2. To address the above flight delay prediction problem, essential data includes
flight details (flight number, airline, departure/arrival times, taxi times),
delay-related information (departure/arrival delays, cancellation status, delay
reasons), and weather conditions (temperature, wind, visibility). Additionally,
temporal factors (day of the week, seasonality) are needed to capture relevant
patterns affecting flight delays.

3. Source data related to flight delays and cancellations for January 2019 – August
2023 is retrieved from the U.S. Department of Transportation's (DOT) On-Time :
Reporting Carrier On-Time Performance (1987-present) application. The source data
was downloaded in subsets by month and joined by year. The collected data on the
whole is kept as a public dataset in Kaggle.
```

```
4. The primary analytics task is CLASSIFICATION. The goal is to predict a binary
outcome: whether a flight will be delayed (Yes/No). In addition to classification,
feature selection and exploratory data analysis (EDA) will be performed to
understand the important factors contributing to delays.
```

2. Data Acquisition

For the problem identified , find an appropriate data set (Your data set must be unique with minimum **20 features and 10k rows**) from any public data source.

2.1 Download the data directly

In [1]:

```
# ##---------Type the code below this line------------------##
import kaggle

dataset_path = 'patrickzel/flight-delay-and-cancellation-dataset-2019-2023'
filename = 'flights_sample_3m.csv'

kaggle.api.dataset_download_file(dataset_path, file_name = filename, path = '.')
```

```
Dataset URL: https://www.kaggle.com/datasets/patrickzel/flight-delay-and-cancellati
on-dataset-2019-2023 (https://www.kaggle.com/datasets/patrickzel/flight-delay-and-c
ancellation-dataset-2019-2023)
```

Out[1]: False

2.2 Code for converting the above downloaded data into a dataframe

In [1]:

```
##---------Type the code below this line------------------##
import pandas as pd
import zipfile as zp

zipfilename = 'flights_sample_3m.csv.zip'

with zp.ZipFile(zipfilename, 'r') as zip_ref:
    zip_ref.extractall('.')

df = pd.read_csv(zipfilename)
```

2.3 Confirm the data has been correctly by displaying the first 5 and last 5 records.

In [3]:
```python
##---------Type the code below this line------------------##

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

print("\nFirst 5 records :")
display(df.head())
print("\nLast 5 records :")
display(df.tail())
```

```
First 5 records :
```

	FL_DATE	AIRLINE	AIRLINE_DOT	AIRLINE_CODE	DOT_CODE	FL_NUMBER	ORIGIN	ORIGIN_CITY
0	2019-01-09	United Air Lines Inc.	United Air Lines Inc.: UA	UA	19977	1562	FLL	Fort Lauderdale, FL
1	2022-11-19	Delta Air Lines Inc.	Delta Air Lines Inc.: DL	DL	19790	1149	MSP	Minneapolis, MN
2	2022-07-22	United Air Lines Inc.	United Air Lines Inc.: UA	UA	19977	459	DEN	Denver, CO
3	2023-03-06	Delta Air Lines Inc.	Delta Air Lines Inc.: DL	DL	19790	2295	MSP	Minneapolis, MN
4	2020-02-23	Spirit Air Lines	Spirit Air Lines: NK	NK	20416	407	MCO	Orlando, FL

```
Last 5 records :
```

	FL_DATE	AIRLINE	AIRLINE_DOT	AIRLINE_CODE	DOT_CODE	FL_NUMBER	ORIGIN	ORIGI
2999995	2022-11-13	American Airlines Inc.	American Airlines Inc.: AA	AA	19805	1522	JAX	Jacks
2999996	2022-11-02	American Airlines Inc.	American Airlines Inc.: AA	AA	19805	1535	ORD	Chic
2999997	2022-09-11	Delta Air Lines Inc.	Delta Air Lines Inc.: DL	DL	19790	2745	HSV	Hunts
2999998	2019-11-13	Republic Airline	Republic Airline: YX	YX	20452	6134	BOS	Bost
2999999	2019-06-15	Southwest Airlines Co.	Southwest Airlines Co.: WN	WN	19393	2823	LGB	Long

2.4 Display the column headings, statistical information, description and statistical summary of the data.

In [4]:
```
##---------Type the code below this line------------------##
print("\nCOLUMN HEADINGS:\n")
print(df.columns)
print("\nINFORMATION AND DESCRIPTION:\n")
print(df.info())
print("\nSTATISTICAL SUMMARY:")
display(df.describe(include='all')
        .apply(lambda s: s.apply(lambda x: '{0:.5f}'.format(x)
                                 if isinstance(x, (int, float)) else x)))
```

COLUMN HEADINGS:

```
Index(['FL_DATE', 'AIRLINE', 'AIRLINE_DOT', 'AIRLINE_CODE', 'DOT_CODE',
       'FL_NUMBER', 'ORIGIN', 'ORIGIN_CITY', 'DEST', 'DEST_CITY',
       'CRS_DEP_TIME', 'DEP_TIME', 'DEP_DELAY', 'TAXI_OUT', 'WHEELS_OFF',
       'WHEELS_ON', 'TAXI_IN', 'CRS_ARR_TIME', 'ARR_TIME', 'ARR_DELAY',
       'CANCELLED', 'CANCELLATION_CODE', 'DIVERTED', 'CRS_ELAPSED_TIME',
       'ELAPSED_TIME', 'AIR_TIME', 'DISTANCE', 'DELAY_DUE_CARRIER',
       'DELAY_DUE_WEATHER', 'DELAY_DUE_NAS', 'DELAY_DUE_SECURITY',
       'DELAY_DUE_LATE_AIRCRAFT'],
      dtype='object')
```

INFORMATION AND DESCRIPTION:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3000000 entries, 0 to 2999999
Data columns (total 32 columns):
 #   Column                   Dtype  
---  ------                   -----  
 0   FL_DATE                  object 
 1   AIRLINE                  object 
 2   AIRLINE_DOT              object 
 3   AIRLINE_CODE             object 
 4   DOT_CODE                 int64  
 5   FL_NUMBER                int64  
 6   ORIGIN                   object 
 7   ORIGIN_CITY              object 
 8   DEST                     object 
 9   DEST_CITY                object 
 10  CRS_DEP_TIME             int64  
 11  DEP_TIME                 float64
 12  DEP_DELAY                float64
 13  TAXI_OUT                 float64
 14  WHEELS_OFF               float64
 15  WHEELS_ON                float64
 16  TAXI_IN                  float64
 17  CRS_ARR_TIME             int64  
 18  ARR_TIME                 float64
 19  ARR_DELAY                float64
 20  CANCELLED                float64
 21  CANCELLATION_CODE        object 
 22  DIVERTED                 float64
 23  CRS_ELAPSED_TIME         float64
 24  ELAPSED_TIME             float64
 25  AIR_TIME                 float64
 26  DISTANCE                 float64
 27  DELAY_DUE_CARRIER        float64
 28  DELAY_DUE_WEATHER        float64
 29  DELAY_DUE_NAS            float64
 30  DELAY_DUE_SECURITY       float64
 31  DELAY_DUE_LATE_AIRCRAFT  float64
dtypes: float64(19), int64(4), object(9)
memory usage: 732.4+ MB
None
```

STATISTICAL SUMMARY:

	FL_DATE	AIRLINE	AIRLINE_DOT	AIRLINE_CODE	DOT_CODE	FL_NUMBER	ORIGIN
count	3000000	3000000	3000000	3000000	3000000.00000	3000000.00000	3000000
unique	1704.00000	18.00000	18.00000	18.00000	nan	nan	380.00000
top	2019-07-25	Southwest Airlines Co.	Southwest Airlines Co.: WN	WN	nan	nan	ATL
freq	2379	576470	576470	576470	nan	nan	153556
mean	nan	nan	nan	nan	19976.29410	2511.53552	nan
std	nan	nan	nan	nan	377.28462	1747.25804	nan
min	nan	nan	nan	nan	19393.00000	1.00000	nan
25%	nan	nan	nan	nan	19790.00000	1051.00000	nan
50%	nan	nan	nan	nan	19930.00000	2152.00000	nan
75%	nan	nan	nan	nan	20368.00000	3797.00000	nan
max	nan	nan	nan	nan	20452.00000	9562.00000	nan

2.5 Write your observations from the above.

1. Size of the dataset
2. What type of data attributes are there?
3. Is there any null data that has to be cleaned?

Score: 2 Marks in total (0.25 marks for 2.1, 0.25 marks for 2.2, 0.5 marks for 2.3, 0.25 marks for 2.4, 0.75 marks for 2.5)

```
--------------Type the answers below this line--------------

1. Size of the dataset: The dataset consists of 3000000 rows and 32 columns totally
comprising 96000000 values(elements).

2. In this dataset, we have three types of data attributes:
   a) NOMINAL - categorical
   b) INTERVAL - numerical
   c) RATIO - numerical

3. YES. The dataset has null values that has to be cleaned.
```

3. Data Preparation

```
If input data is numerical or categorical, do 3.1, 3.2 and 3.4
If input data is text, do 3.3 and 3.4
```

3.1 Check for

- duplicate data
- missing data
- data inconsistencies

In [2]:
```python
##---------Type the code below this line------------------##
import numpy as np

# - DUPLICATE DATA
duplicate_rows = df[df.duplicated()]

if duplicate_rows.empty:
    print("NO DUPLICATE DATA FOUND IN THE DATASET\n")
else:
    print("Duplicate data: ",duplicate_rows.shape[0],"rows\n")
    print(duplicate_rows,"\n")

# - MISSING DATA
missing_data = df.isnull().sum()
missing_columns = missing_data[missing_data>0]

if missing_columns.empty:
    print("NO MISSING DATA FOUND IN THE DATASET\n")
else:
    print("Missing data found in: ",missing_columns.shape[0],"columns\n")
    missing_percentage = round((df.isnull().sum() / len(df)) * 100,2)
    missing_col_percent = missing_percentage[missing_percentage>0]
    missing_info = pd.concat([missing_columns, missing_col_percent],axis=1)
    missing_info.columns = ['Missing_row_count','Missed_data_%']
    print(missing_info,"\n")

# - DATA INCONSISTENCIES
print("\nData Inconsistencies Checks:\n")

# Check whether time is in the provided format (hhmm) and valid datatype
decimal_arr_time = df[df['ARR_TIME'].notnull() % 1 != 0.0]
decimal_dep_time = df[df['DEP_TIME'].notnull() % 1 != 0.0]

if len(decimal_arr_time) == 0 and len(decimal_dep_time) == 0:
    print("INCONSISTENT - Invalid datatype float for columns",end=" ")
    print("with TIME info (in format hhmm)")
else:
    print("CONSISTENT - Valid datatype and format")

# Check if distance between two airports is 0 or negative
zero_distance_flights = df[df['DISTANCE'] <= 0]
if len(zero_distance_flights) > 0:
    print("INCONSISTENT - Invalid distance between two airports")
else:
    print("CONSISTENT - There are no zero/negative distance flights.")

def convert_to_minutes(time):
    hours = time // 100
    minutes = time % 100
    return hours * 60 + minutes

# Check if the TAXI_OUT VALUES in minutes are correct
df['DEP_TIME_MIN'] = df['DEP_TIME'].apply(convert_to_minutes)
df['WHEELS_OFF_MIN'] = df['WHEELS_OFF'].apply(convert_to_minutes)
df['DEP_W_OFF_DIFF'] = df['WHEELS_OFF_MIN'] - df['DEP_TIME_MIN']

df.loc[df['DEP_W_OFF_DIFF'] < 0, 'DEP_W_OFF_DIFF'] += 24 * 60

df['IS_T_OUT_CLOSE'] = np.isclose(df['DEP_W_OFF_DIFF'], df['TAXI_OUT'],
                                  rtol=1e-2, atol=1e-2, equal_nan=True)
if all(df['IS_T_OUT_CLOSE']):
    print("CONSISTENT",end=" - ")
    print("TAXI_OUT matches with Wheels off & Departure time difference")
else:
    inconsistent_taxi_out = df[~df['IS_T_OUT_CLOSE']].index
```

```
    print("INCONSISTENT",end=" - ")
    print("TAXI_OUT values mismatch with Wheels off & Departure time difference")

# Check if the TAXI_IN VALUES in minutes are correct
df['ARR_TIME_MIN'] = df['ARR_TIME'].apply(convert_to_minutes)
df['WHEELS_ON_MIN'] = df['WHEELS_ON'].apply(convert_to_minutes)
df['ARR_W_ON_DIFF'] = df['ARR_TIME_MIN'] - df['WHEELS_ON_MIN']

df.loc[df['ARR_W_ON_DIFF'] < 0, 'ARR_W_ON_DIFF'] += 24 * 60

df['IS_T_IN_CLOSE'] = np.isclose(df['ARR_W_ON_DIFF'], df['TAXI_IN'],
                                 rtol=1e-2, atol=1e-2, equal_nan=True)
if all(df['IS_T_IN_CLOSE']):
    print("CONSISTENT",end=" - ")
    print("TAXI_IN matches with Wheels ON & Arrival time difference")
else:
    inconsistent_taxi_in = df[~df['IS_T_IN_CLOSE']].index
    print("INCONSISTENT",end=" - ")
    print("TAXI_IN values mismatch with Wheels on & Arrival time difference")

df = df.drop(columns=['DEP_TIME_MIN','WHEELS_OFF_MIN',
                      'DEP_W_OFF_DIFF','IS_T_OUT_CLOSE',
                      'ARR_TIME_MIN','WHEELS_ON_MIN',
                      'ARR_W_ON_DIFF','IS_T_IN_CLOSE'])

# Check whether no CANCELLED/DIVERTED flight data is present
if all(~(df.CANCELLED.astype(bool))):
    print("CONSISTENT - CANCELLED flight data not found")
elif all(~(df.DIVERTED.astype(bool))):
    print("CONSISTENT - DIVERTED flight data not found")
else:
    print("INCONSISTENT - CANCELLED and DIVERTED flights data are present (misleadin
```

```
NO DUPLICATE DATA FOUND IN THE DATASET

Missing data found in:  17 columns

                         Missing_row_count  Missed_data_%
DEP_TIME                             77615           2.59
DEP_DELAY                            77644           2.59
TAXI_OUT                             78806           2.63
WHEELS_OFF                           78806           2.63
WHEELS_ON                            79944           2.66
TAXI_IN                              79944           2.66
ARR_TIME                             79942           2.66
ARR_DELAY                            86198           2.87
CANCELLATION_CODE                  2920860          97.36
CRS_ELAPSED_TIME                        14            NaN
ELAPSED_TIME                         86198           2.87
AIR_TIME                             86198           2.87
DELAY_DUE_CARRIER                  2466137          82.20
DELAY_DUE_WEATHER                  2466137          82.20
DELAY_DUE_NAS                      2466137          82.20
DELAY_DUE_SECURITY                 2466137          82.20
DELAY_DUE_LATE_AIRCRAFT            2466137          82.20


Data Inconsistencies Checks:

INCONSISTENT - Invalid datatype float for columns with TIME info (in format hhmm)
CONSISTENT - There are no zero/negative distance flights.
CONSISTENT - TAXI_OUT matches with Wheels off & Departure time difference
INCONSISTENT - TAXI_IN values mismatch with Wheels on & Arrival time difference
INCONSISTENT - CANCELLED and DIVERTED flights data are present (misleading data)
```

3.2 Apply techiniques

- to remove duplicate data
- to impute or remove missing data
- to remove data inconsistencies

In [3]:
```
##---------Type the code below this line------------------##
import datetime

print("Dataset size before applying techniques: ",df.shape)

# - REMOVE MISSING DATA
#   Remove columns if missing data is above 80%
columns_to_drop = missing_col_percent[missing_col_percent>80].index
df = df.drop(columns=columns_to_drop)

#   Remove rows from columns with lesser missing values
rows_to_drop = missing_col_percent[~(missing_col_percent>80)].index
df = df.dropna(subset=rows_to_drop)

# - REMOVE DATA INCONSISTENCIES

# datatype inconsistencies
def format_time(hhmm):
    hhmm = "{0:04d}".format(int(hhmm))
    hour = int(hhmm[0:2])
    minute = int(hhmm[2:4])
    if hour == 24:
        hour = 0
    time = datetime.time(hour, minute)
    return time

df['CRS_DEP_TIME'] = df['CRS_DEP_TIME'].apply(format_time)
df['DEP_TIME'] = df['DEP_TIME'].apply(format_time)
df['WHEELS_OFF'] = df['WHEELS_OFF'].apply(format_time)
df['WHEELS_ON'] = df['WHEELS_ON'].apply(format_time)
df['CRS_ARR_TIME'] = df['CRS_ARR_TIME'].apply(format_time)
df['ARR_TIME'] = df['ARR_TIME'].apply(format_time)

# actual & computed value mismatch inconsistency
valid_indices = [i for i in inconsistent_taxi_in if i in df.index]
df = df.drop(index=valid_indices)

# misleading info inconsistency
df = df.drop(df[df.CANCELLED == True].index)
df = df.drop(df[df.DIVERTED == True].index)
df = df.reset_index(drop=True)
df = df.drop(columns=['CANCELLED','DIVERTED'])

print("Dataset size after removing missing and inconsistent data: ",df.shape)
```

```
Dataset size before applying techniques:  (3000000, 32)
Dataset size after removing missing and inconsistent data:  (2913799, 24)
```

Remove redundant attributes

In [4]:
```
redundant_attributes = ['AIRLINE_DOT','AIRLINE_CODE','DOT_CODE',
                        'ORIGIN_CITY','DEST_CITY']
df = df.drop(columns=redundant_attributes)
print("Dataset size after removing redundant attributes: ",df.shape)
```

```
Dataset size after removing redundant attributes:  (2913799, 19)
```

Create derived attributes

In [5]:
```
# Derive year, quarter, month and day from Flight date
df['FL_DATE'] = pd.to_datetime(df['FL_DATE'])

df['JOURNEY_YEAR'] = df['FL_DATE'].dt.year
df['JOURNEY_QTR'] = df['FL_DATE'].dt.quarter
df['JOURNEY_MONTH'] = df['FL_DATE'].dt.month
df['JOURNEY_DAY'] = df['FL_DATE'].dt.day

# Derive the difference between CRS_ELAPSED_TIME and ELAPSED_TIME
df['ELAPSED_TIME_DIFF'] = df['ELAPSED_TIME'] - df['CRS_ELAPSED_TIME']

print("Dataset size after creating derived attributes: ",df.shape)
```

```
Dataset size after creating derived attributes:  (2913799, 24)
```

Remove derived attributes

In [6]:
```
derived_attributes = ['FL_DATE', # derived year,qtr,month,day
                      'CRS_DEP_TIME','DEP_TIME', # derived DEP_DELAY
                      'WHEELS_OFF','WHEELS_ON', # derived AIR_TIME
                      'CRS_ARR_TIME','ARR_TIME', # derived ARR_DELAY
                      'CRS_ELAPSED_TIME','ELAPSED_TIME'] # derived ELAPSED_TIME_DIFF
df = df.drop(columns=derived_attributes)
print("Dataset size after removing derived attributes: ",df.shape)
```

```
Dataset size after removing derived attributes:  (2913799, 15)
```

3.3 Encode categorical data

In [7]:
```
##---------Type the code below this line------------------##
from sklearn.preprocessing import LabelEncoder

categorical_columns = ['FL_NUMBER', 'AIRLINE', 'ORIGIN', 'DEST']
label_encoders = {}
for column in categorical_columns:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

print(len(categorical_columns),"categorical attributes encoded using Label encoder")
```

```
4 categorical attributes encoded using Label encoder
```

3.4 Report

Mention and justify the method adopted

- to remove duplicate data, if present
- to impute or remove missing data, if present
- to remove data inconsistencies, if present

OR for textdata

- How many tokens after step 3?
- how may tokens after stop words filtering?

If the any of the above are not present, then also add in the report below.

Score: 2 Marks (based on the dataset you have, the data prepreation you had to do and report typed, marks will be distributed between 3.1, 3.2, 3.3 and 3.4)

##---------Type the code below this line------------------##

DATA PREPARATION REPORT

1. Duplicate Data: No duplicate data was found in the dataset.
2. Missing Data: Missing data was identified in 17 columns.

   - **DROP COLUMNS**
     - 6 columns have more than 80% missing data. So Dropping those columns is appropriate. Other methods like imputing values will not be a good option since that would not be a correct measure & dropping missing rows will cause huge loss in this case.
   - **DROP ROWS WITH MISSING VALUES**
     - The remaining 11 columns had less than 3% missing data. I didn't choose imputing because the missing data was found majorly in time representing columns and can't take any statistical method to replace. So rows containing these missing values were deleted. This seems appropriate since deleting 3% of rows will not affect this large dataset of 3 million rows.
3. Data Inconsistencies:

   - Invalid Data Types:
     - The time-related columns had incorrect data types and were
       - **CONVERSION TO DATE-TIME FORMAT (hh:mm:ss)** The float value which was in format hhmm is handled and converted to a proper date-time format.
   - Negative Distance: NOT FOUND (CONSISTENT)
     - Distance between airports was verified to avoid any negative values.
   - Taxi Times Accuracy:
     - Inconsistencies were found in taxi times (taxi-in time). Identified 3 rows.
       - **DROP ROWS WITH INCONSISTENT VALUE** This seems appropriate since deleting just 3 rows will not affect this large dataset of 3 million rows.
   - Cancelled & Diverted Flights:
     - Rows representing cancelled and diverted flights were found which could skew delay analysis.
       - **DROP ROWS AND THEN COLUMNS** This seems appropriate to delete data that might mislead the actual analysis. After dropping rows representing them, the actual columns is unnecessary, hence dropped the two associated columns.
4. Redundant Attributes:

   - Identified 5 redundant attributes where the attribute is actually being represented in the other attribute of the dataset.

        - AIRLINE --> 'AIRLINE_DOT','AIRLINE_CODE','DOT_CODE'
        - ORIGIN --> 'ORIGIN_CITY'
        - DEST --> 'DEST_CITY'
            - **DROPPED 5 REDUNDANT ATTRIBUTES** by keeping the actual 3 attributes only to simplify and reduce data dimensionality.

5. Derived Attributes:

    - **DERIVED 5 NEW ATTRIBUTES** in total from existed 2 attributes which is in the more meaningful format for the model to learn.
        - FL_DATE --> 'JOURNEY_YEAR','JOURNEY_QTR', 'JOURNEY_MONTH', 'JOURNEY_DAY'
        - CRS_ELAPSED_TIME, ELAPSED_TIME --> 'ELAPSED_TIME_DIFF'
    - **DROPPED 9 DERIVED ATTRIBUTES** related to delay differences including these 3, which were denoting the available attributes.

6. Categorical data Encoding:

    - **LABEL ENCODING**
        - This is suitable for tree-based models where the algorithm can handle integer encoding effectively.
        - Avoided one-hot encoding due to large number of unique values in attributes which result in high dimensionality.

RESULT:

- The number of columns was reduced from 32 to 15.
- The dataset size was reduced by nearly 86,000 rows (3%) due to the removal of rows with missing and inconsistent data.

.

3.5 Identify the target variables.

- Separate the data from the target such that the dataset is in the form of (X,y) or (Features, Label)
- Discretize / Encode the target variable or perform one-hot encoding on the target or any other as and if required.
- Report the observations

Score: 1 Mark

In [8]:
```python
##---------Type the code below this line------------------##

# Binary Discretization
status = []
for value in df['ARR_DELAY']:
    if value <= 10:
        status.append(0)
    else:
        status.append(1)

df['FL_DELAY_STATUS'] = status
print("Target Variable FL_DELAY_STATUS computed on performing Binary discretization"
print(df['FL_DELAY_STATUS'].value_counts())

# (X,y) = (Features, Label)
X = df.drop(labels=['FL_DELAY_STATUS'], axis=1)
y = df['FL_DELAY_STATUS']
```

```
Target Variable FL_DELAY_STATUS computed on performing Binary discretization
FL_DELAY_STATUS
0    2293863
1     619936
Name: count, dtype: int64
```

```
Observations:

- The FL_DELAY_STATUS target variable, after binary discretization, shows a class
imbalance with 2,293,863 flights (0) classified as on-time and 619,936 flights (1)
classified as delayed.
- This imbalance indicates that a large majority of flights were on time, while a
smaller fraction (~20%) were delayed which is common in real time.
```

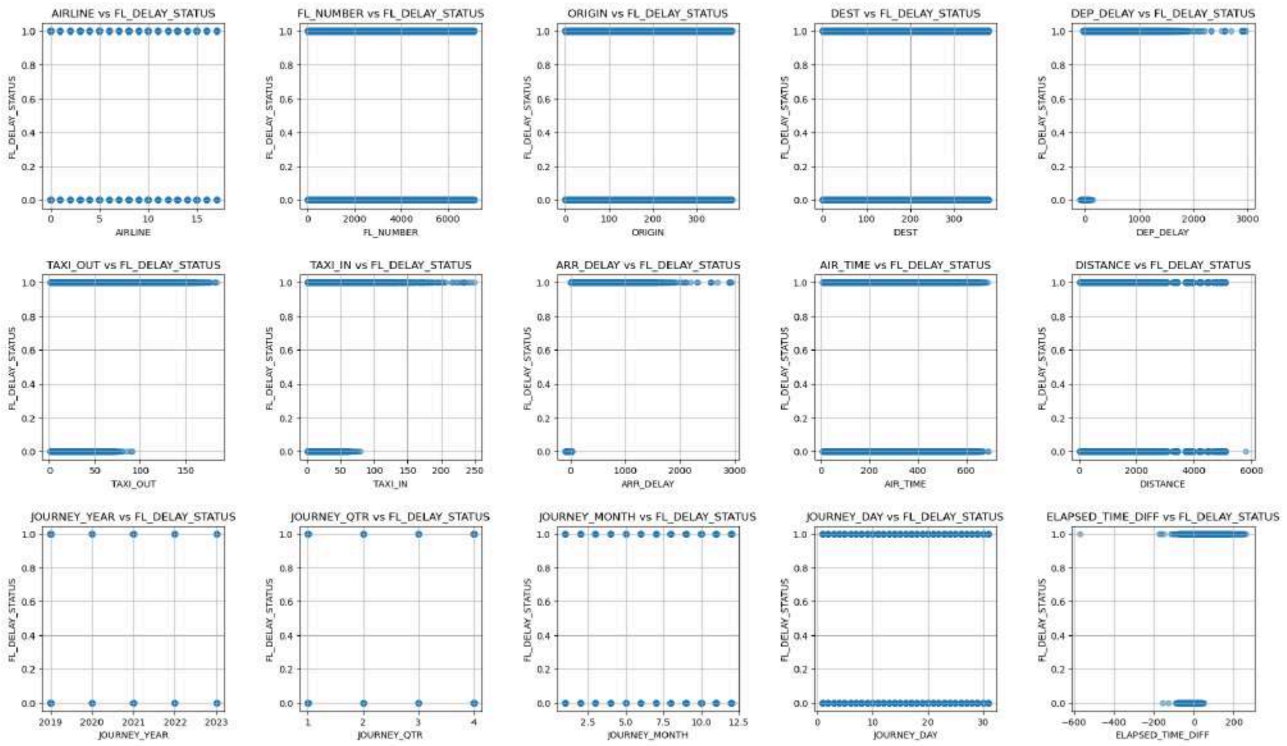
4. Data Exploration using various plots

4.1 Scatter plot of each quantitative attribute with the target.

Score: 1 Mark

In [63]:

```
import matplotlib.pyplot as plt

quantitative_columns=X.select_dtypes(include=['float64', 'int64', 'int32']).columns

fig, axes = plt.subplots(3, 5, figsize=(20, 12))
fig.tight_layout(pad=5.0)
axes = axes.flatten()

for i, column in enumerate(quantitative_columns):
    ax = axes[i]
    ax.scatter(df[column], df['FL_DELAY_STATUS'], alpha=0.5)
    ax.set_title(f'{column} vs FL_DELAY_STATUS')
    ax.set_xlabel(column)
    ax.set_ylabel('FL_DELAY_STATUS')
    ax.grid(True)

plt.show()
```



4.2 EDA using visuals

- Use (minimum) 2 plots (pair plot, heat map, correlation plot, regression plot...) to identify the optimal set of attributes that can be used for classification.
- Name them, explain why you think they can be helpful in the task and perform the plot as well. Unless proper justification for the choice of plots given, no credit will be awarded.

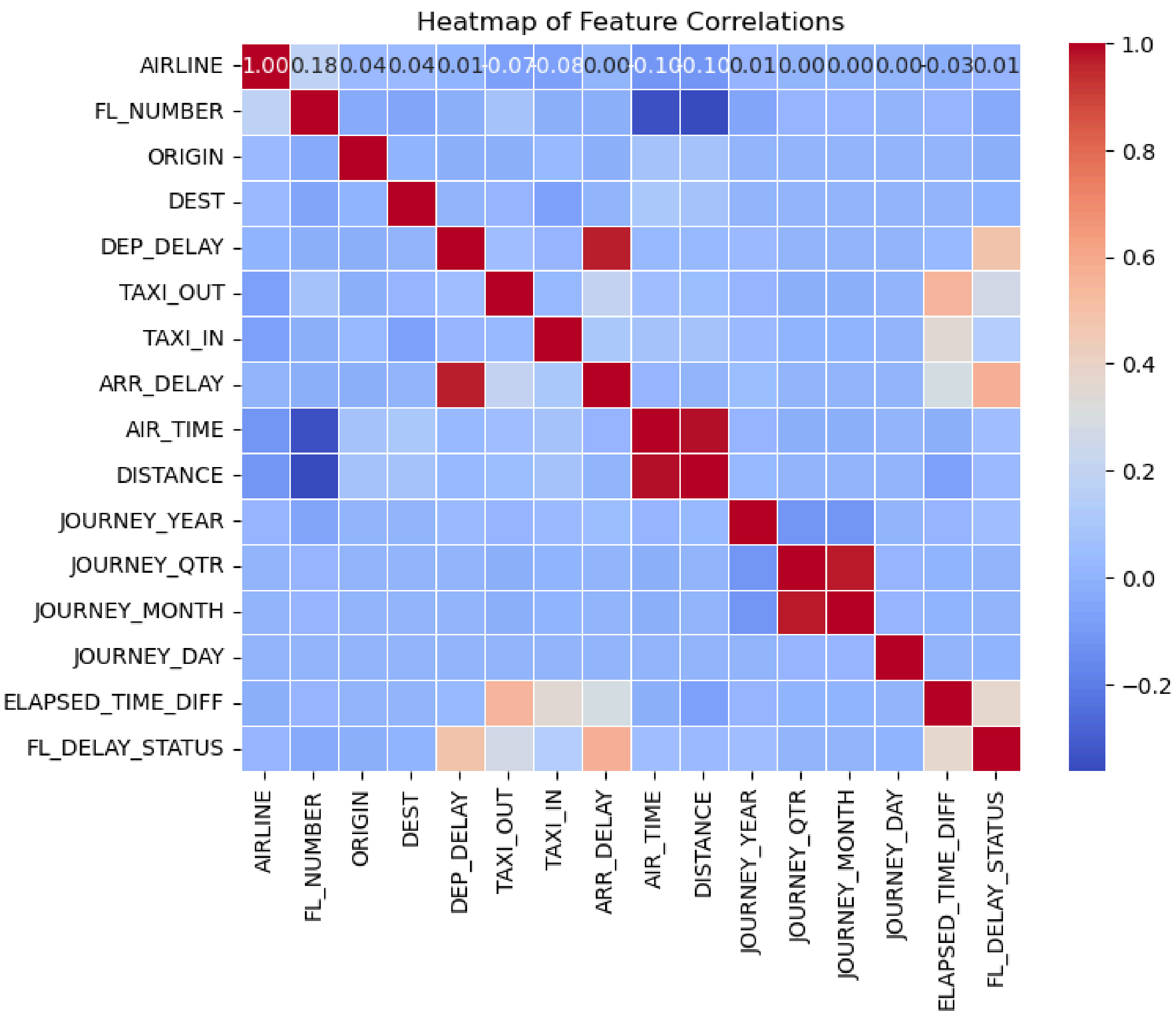
Score: 2 Marks

```
EDA JUSTIFICATION:

- HEATMAP:
            A heatmap provides a clear and compact summary of how different
variables relate to each other. It is a very simple representation where we can
quickly identify positive, negative and no correlations.

  Task: To identify non-correlated attributes and delete those to train better
models.

- PAIR PLOTS:
```

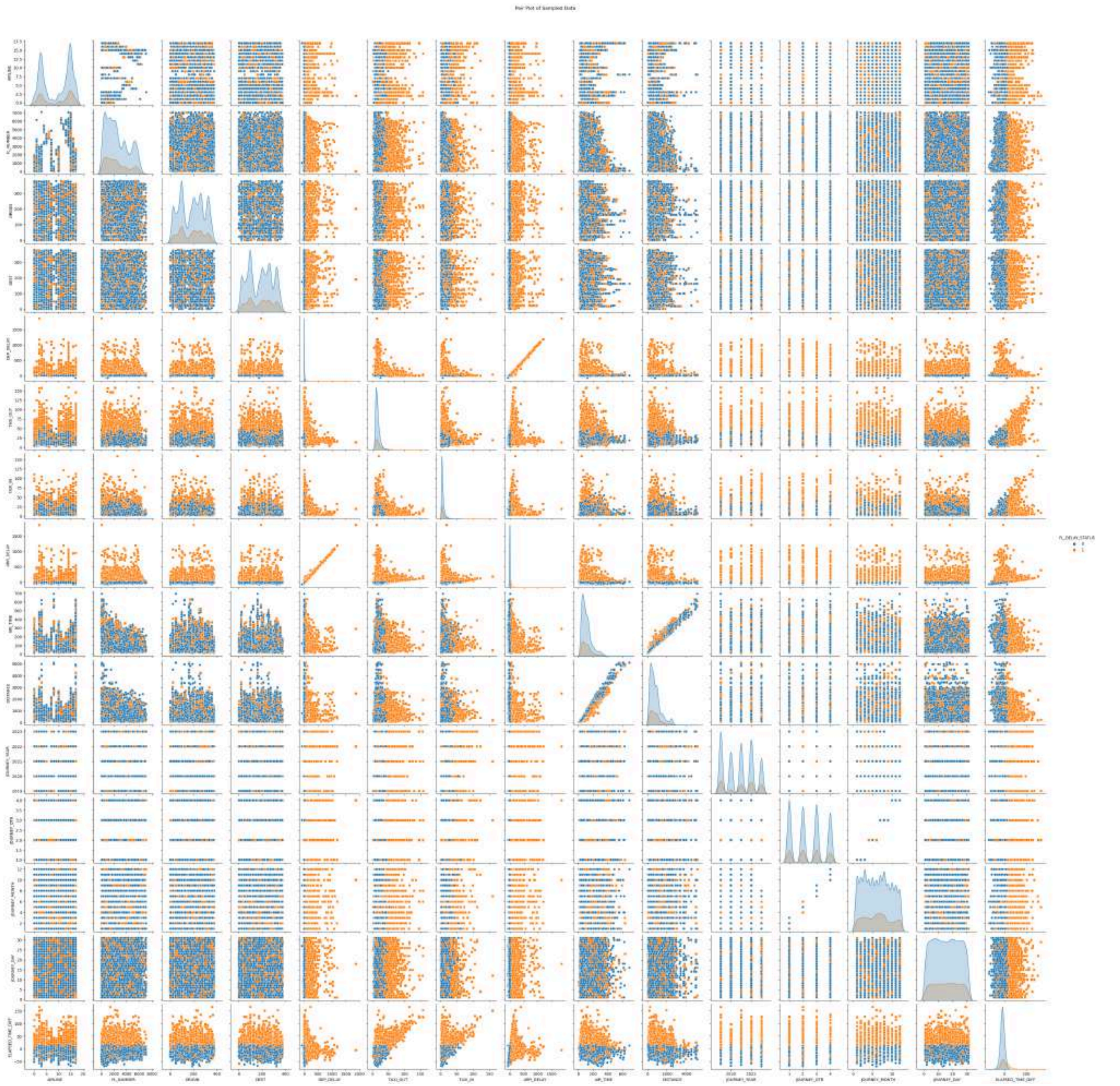
Pair plots are useful for visualizing the relationships between multiple features simultaneously. They show both the distribution of individual variables (diagonal plots) and pairwise relationships (scatter plots). This is similar to heatmap with respect to features so we can easily spot potential correlations, clusters, or trends across several feature pairs.

Task: Can identify both correlations and clusters as well which will be useful in analysing clusters.

In [64]:
```
import seaborn as sns

quantitative_features = X.select_dtypes(include=['float64', 'int64','int32']).copy()
quantitative_features['FL_DELAY_STATUS'] = y
correlation_matrix = quantitative_features.corr()

plt.figure(figsize=(8, 6))
sns.heatmap(correlation_matrix,annot=True,cmap='coolwarm',fmt='.2f',linewidths=0.5)
plt.title('Heatmap of Feature Correlations')
plt.show()
```



```
In [42]: df_subset = df.sample(n=50000, random_state=42)

         sns.pairplot(df_subset, hue='FL_DELAY_STATUS', markers=["o", "s"])
         plt.suptitle('Pair Plot of Sampled Data', y=1.02)
         plt.show()
```



5. Data Wrangling

5.1 Univariate Filters

Numerical and Categorical Data

- Identify top 5 significant features by evaluating each feature independently with respect to the target/other variable by exploring

1. Mutual Information (Information Gain)
2. Gini index
3. Gain Ratio
4. Chi-Squared test
5. Strenth of Association

(From the above 5 you are required to use only any **two**)

Score: 3 Marks

In [43]:
```
##---------Type the code below this line------------------##
from sklearn.feature_selection import mutual_info_classif

mutual_info = mutual_info_classif(X, y, discrete_features='auto')
mutual_info_series = pd.Series(mutual_info, index=X.columns)
mutual_info_series = mutual_info_series.sort_values(ascending=False)

# Top 5 features by Mutual Information
print("Top 5 features by Mutual Information:")
print(mutual_info_series.head(5))
```

```
Top 5 features by Mutual Information:
ARR_DELAY           0.520110
DEP_DELAY           0.279015
JOURNEY_YEAR        0.103315
ELAPSED_TIME_DIFF   0.098030
JOURNEY_QTR         0.095811
dtype: float64
```

In [45]:
```
from sklearn.preprocessing import KBinsDiscretizer

def gini_impurity(y):
    classes, counts = np.unique(y, return_counts=True)
    p = counts / counts.sum()
    return 1 - np.sum(p ** 2)

def gini_for_feature(feature, target):
    feature_values = np.unique(feature)
    gini_total = 0.0
    total_samples = len(target)
    for value in feature_values:
        target_split = target[feature == value]
        gini_split = gini_impurity(target_split)
        gini_total += (len(target_split) / total_samples) * gini_split
    return gini_total

def gini_for_all_features(df, target_column):
    features = df.columns.drop(target_column)
    target = df[target_column].values
    gini_scores = {}
    for feature in features:
        if np.issubdtype(df[feature].dtype, np.number):
            binner = KBinsDiscretizer(n_bins=5,encode='ordinal',strategy='uniform')
            feature_binned = binner.fit_transform(df[[feature]]).flatten()
        else:
            feature_binned = df[feature].values
        gini_scores[feature] = gini_for_feature(feature_binned, target)
    return pd.Series(gini_scores).sort_values()

gini_scores = gini_for_all_features(df, 'FL_DELAY_STATUS')
top_5_features = gini_scores.head(5)
print("Top 5 features by Gini Index:")
print(top_5_features)
```

```
Top 5 features by Gini Index:
TAXI_OUT        0.315154
JOURNEY_YEAR    0.330938
TAXI_IN         0.332416
ARR_DELAY       0.333285
DEP_DELAY       0.333307
dtype: float64
```

5.2 Report observations

Write your observations from the results of each method. Clearly justify your choice of the method.

Score 1 mark

##---------Type the code below this line------------------##

Observations:

- Mutual Information highlights ARR_DELAY as the most influential feature by a large margin, while Gini Index ranks TAXI_OUT as the most significant.
- ARR_DELAY and DEP_DELAY are consistently ranked as important by both Mutual Information and Gini Index, indicating they have a strong relationship with the target variable.
- JOURNEY_YEAR also appears in the top 5 in both methods, suggesting that the year of the journey is also relevant to predicting flight delays.

Choice : GINI INDEX

Justification:

- Gini Index is particularly sensitive to class imbalances and reflects the ability of a feature to separate the data into target classes.
- Mutual Information would likely be the more appropriate method in case of predicting the degree of delay. However, we are focusing on binary classification whether a flight will be delayed or not.
- So Gini Index could be a valuable method, especially in understanding factors like taxi times (TAXI_OUT and TAXI_IN), which play a role in whether a flight gets delayed.

6. Implement Machine Learning Techniques

Use any 2 ML tasks

1. Classification
2. Clustering
3. Association Analysis
4. Anomaly detection

You may use algorithms included in the course, e.g. Decision Tree, K-means etc. or an algorithm you learnt on your own with a brief explanation. A clear justification have to be given for why a certain algorithm was chosen to address your problem.

Score: 4 Marks (2 marks each for each algorithm)

In [9]:
```
# Removing columns based on correlation and relationships identified by EDA and data

least_correlated_attributes = ['AIRLINE', 'FL_NUMBER', 'ORIGIN', 'DEST',
                               'JOURNEY_QTR', 'JOURNEY_MONTH', 'JOURNEY_DAY']
df = df.drop(columns=least_correlated_attributes)
X = X.drop(columns=least_correlated_attributes)
print("Final Dataset size after removing least correlated attributes: ",df.shape)
```

```
Final Dataset size after removing least correlated attributes:  (2913799, 9)
```

6.1 ML technique 1 + Justification

```
ML TECHNIQUE  : DECISION TREE CLASSIFIER

TYPE          : SUPERVISED LEARNING - (1) Classification

EXPLANATION   :
    - Decision tree classifier is the classifier that will be trained to predict
the target.
    - Gini index is chosen as the splitting criterion in a way that minimizes
impurity and results in more homogeneous branches in the decision tree so that the
tree is optimized to reduce misclassification.
    - GridSearchCV is used to find the optimal hyperparameters and selects the best
model based on cross-validation scores that balance model performance and
complexity.
    - Here, 5-fold cross-validation is performed to ensure that the model is
validated on different subsets of the data, helping avoid overfitting.

JUSTIFICATION :
    Decision Trees provide a clear and easy-to-interpret model, allowing US to
understand how decisions (in this case, flight delays) are made based on the input
features.
```

    Decision Trees can capture complex, non-linear relationships between input features, which is crucial in the flight delay problem where multiple factors like weather, time of day, and operational conditions interact.
    Decision Trees do not require feature scaling or normalization, making preprocessing simpler compared to algorithms like KNN.

In [69]:
```
##---------Type the code below this line------------------##
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
                                                    random_state=42)

clf = DecisionTreeClassifier(criterion='gini', random_state=42)

param_grid = {
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 5]
}
grid_search = GridSearchCV(estimator=clf, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
best_tree_clf = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best Parameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 2}
```

6.2 ML technique 2 + Justification

ML TECHNIQUE : K-NEAREST NEIGHBOUR (KNN) CLASSIFIER

TYPE   : UNSUPERVISED LEARNING - (2) Clustering

EXPLANATION   :
    - KNN algorithm is used to classify data points based on the distance from their neighbors i.e. based on its similarity to other flights.
    - Feature scaling is done using StandardScaler to ensure that all features contribute equally to the distance calculation.
    - A pipline is used to combine preprocessing steps and model training into one streamlined process, so that scaling and KNN are applied together during training and testing.
    - GridSearchCV is used to find the optimal hyperparameters ensuring the best possible performance on the dataset without manually trying different combinations.
    - The kd-tree algorithm optimizes the neighbor search, making KNN computationally efficient, especially for large datasets.

JUSTIFICATION :
    KNN is a simple yet powerful instance-based learning algorithm based on the most similar data points from the training set. This is well-suited for problems like flight delay prediction, where flights with similar characteristics often share similar outcomes.
    KNN makes no assumption about the underlying data distribution, allowing it to perform well in scenarios with complex patterns.

In [11]:
```python
##---------Type the code below this line------------------##
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

scaler = StandardScaler()
knn = KNeighborsClassifier()

param_grid = {
    'knn__n_neighbors': [3, 5],
    'knn__weights': ['uniform', 'distance'],
    'knn__p': [1, 2],
    'knn__algorithm': ['kd_tree']
}

pipeline = Pipeline([
    ('scaler', scaler),
    ('knn', knn)
])

grid_search = GridSearchCV(estimator=pipeline, param_grid=param_grid,
                           cv=5, n_jobs=-1, verbose=2)
grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_
print(f'Best Parameters: {best_params}')
best_knn = grid_search.best_estimator_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
Best Parameters: {'knn__algorithm': 'kd_tree', 'knn__n_neighbors': 5, 'knn__p': 1,
'knn__weights': 'distance'}
```

7. Conclusion

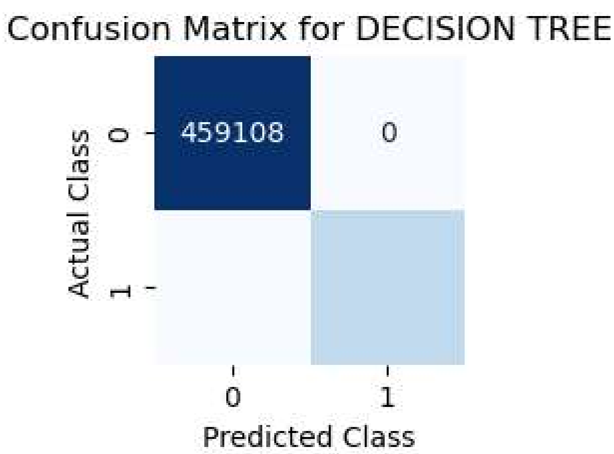
Compare the performance of the ML techniques used.

Derive values for preformance study metrics like accuracy, precision, recall, F1 Score, AUC-ROC etc to compare the ML algos and plot them. A proper comparision based on different metrics should be done and not just accuracy alone, only then the comparision becomes authentic. You may use Confusion matrix, classification report, Word cloud etc as per the requirement of your application/problem.

Score 1 Mark

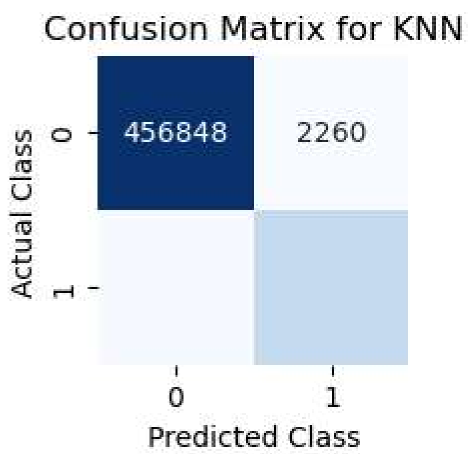
In [70]:
```python
##---------Type the code below this line------------------##
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print('-------------------')
print('   DECISION TREE')
print('-------------------')

y_pred = best_tree_clf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')
print(classification_report(y_test, y_pred))

print('Confusion Matrix:')
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

plt.figure(figsize=(2, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for DECISION TREE')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()
```

```
-------------------
   DECISION TREE
-------------------
Accuracy: 1.00
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    459108
           1       1.00      1.00      1.00    123652

    accuracy                           1.00    582760
   macro avg       1.00      1.00      1.00    582760
weighted avg       1.00      1.00      1.00    582760

Confusion Matrix:
[[459108      0]
 [     0 123652]]
```



In [14]:
```
##---------Type the code below this line------------------##
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

print('-------------------')
print('K-NEAREST NEIGHBOUR')
print('-------------------')

y_pred = best_knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')

print('Classification Report:')
print(classification_report(y_test, y_pred))

print('Confusion Matrix:')
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

plt.figure(figsize=(2, 2))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix for KNN')
plt.ylabel('Actual Class')
plt.xlabel('Predicted Class')
plt.show()
```

```
-------------------
K-NEAREST NEIGHBOUR
-------------------
Accuracy: 0.99
Classification Report:
              precision    recall  f1-score   support

           0       0.99      1.00      0.99    459108
           1       0.98      0.96      0.97    123652

    accuracy                           0.99    582760
   macro avg       0.99      0.98      0.98    582760
weighted avg       0.99      0.99      0.99    582760

Confusion Matrix:
[[456848   2260]
 [  4650 119002]]
```



8. Solution

What is the solution that is proposed to solve the business problem discussed in Section 1. Also share your learnings while working through solving the problem in terms of challenges, observations, decisions made etc.

Score 2 Marks

--------------Type the answers below this line--------------

```
##---------Type the answer below this line------------------##

SOLUTION :
    The proposed solution to solve the business problem of predicting whether a
flight will be delayed or not is to build and use a classification model (two
models have been implemented and evaluated: Decision Tree and K-Nearest Neighbors)
    By predicting potential delays, airlines and airport operators can take
preemptive measures on data-driven decision making to reduce delays, optimize
scheduling, and enhance passenger satisfaction.

CHALLENGES :
    - The Decision Tree model showed perfect accuracy on the training set,
indicating potential overfitting.
    - KNN, being computationally intensive, required optimization in terms of
hyperparameters and distance algorithms.

OBSERVATIONS :
    - Handling missing values and ensuring data consistency is very essential
because datasets often have incomplete entries or discrepancies that need proper
data cleaning before model training.
    - A significantly high number of redundant attributes have been identified.
    - Features like arrival delay, departure delay, and taxi times were highly
predictive of flight delays.
    - Temporal variables also had significant impacts on delay predictions,
highlighting the importance of including these factors in the model.

DECISIONS MADE:
    - Appropriate methods of handling missing and inconsistent data has been made
    - Selected and derived features that were most informative for predicting
flight delays.
    - The algorithm selection (both decision tree and KNN) has yielded an higher
accuracy
    - Used GridSearchCV to find the best model parameters and improve performance.
```

##NOTE All Late Submissions will incur a penalty of -2 marks. Do ensure on time submission to avoid penalty.

Good Luck!!!