FLIGHT DELAY PREDICTION

Project Report- NoSQL Database- BDM 1113

Abstract

This NoSQL project utilizes MongoDB as the database to store and manage large volumes of flight data, including historical flight information and weather data. The project focuses on building flight delay prediction models using various machine learning techniques, such as regression, decision trees. The models will be trained on historical data and updated with real-time data to improve accuracy. The ultimate goal of the project is to provide airlines and passengers with more accurate and timely information about flight delays.

Group 5

Ahmed Abdulrahim

Bimsara Geethachapa Siman Meru Pathiranage

Jasleen Kaur

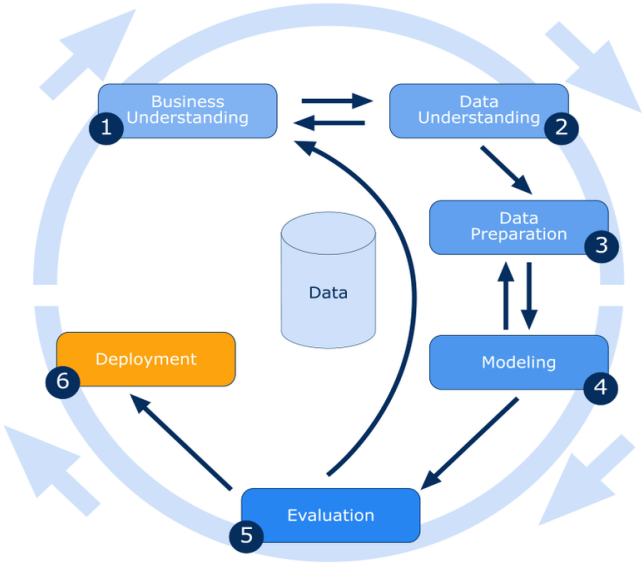
Simran

Simranjeet Kaur

Table of Contents

1. Methodology

In this project we are using the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework to guide the data mining process. . CRISP-DM (Cross-Industry Standard Process for Data Mining) is a widely used methodology for data mining, which provides a structured approach to planning, executing, and evaluating data mining projects. It is a process model that outlines six phases of the data mining process. These phases are:



1. Business Understanding
2. Data Understanding
3. Data Preparation
4. Modeling
5. Evaluation
6. Deployment

We are applying the CRISP-DM framework to predict flight delays. We started with the Business Understanding phase, where we identified the problem and defined the project goal. Next, in the Data Understanding phase, we gathered and explored historical flight data and weather information. Then, we moved to the Data Preparation phase, where we cleaned, transformed, and prepared the data for modeling. The Modeling phase involved building and testing several machine learning models to predict flight delays.

We then evaluated the performance of the different models in the Evaluation phase and selected the best one. Finally, in the Deployment phase, we will implement the model and integrate it into the airline's operations to help prevent and minimize the impact of flight delays .

1.1 Business Understanding phase

In this phase, we need to identify the business problem (e.g., flight delays) and define the project's goal. This phase helps ensure that the project's objectives align with the business needs.

1.2 Data Understanding phase

In this phase, needs to gather and explore historical flight data and weather data to gain a better understanding of the data they will be working with. And it involves identifying potential data quality issues and deciding which data to include or exclude from the analysis.

1.3 Data Preparation phase

In this phase, it needs to transform, clean, and prepare the data for modeling. This can involve tasks such as filling in missing values, removing duplicates, and scaling variables. The goal of this phase is to ensure that the data is in a suitable format for modeling.

1.4 Modeling phase

In this phase, needs to build and tests several machine learning models to predict flight delays. This can involve tasks such as feature selection, hyperparameter tuning, and cross-validation. The goal of this phase is to find the best-performing model.

1.5. Evaluation phase

In this phase, needs to compare the performance of the different models using appropriate metrics such as accuracy, precision, recall, or F1 score. The goal of this phase is to select the best model for deployment.

1.6 Deployment phase

In this phase, needs to implement the model and integrate it into the airline's operations. This can involve tasks such as building a user interface, setting up automated data pipelines, and monitoring the model's performance in production. The goal of this phase is to help prevent and minimize the impact of flight delays.

2. Business Understanding

Flight delays have been a persistent problem in the airline industry for many years, causing significant financial losses and inconvenience for both airlines and passengers. By developing an accurate prediction model, we can help airlines and passengers make informed decisions and take proactive measures to minimize the impact of delays.

Flight delays can cause missed connections, lost profits, and higher operational expenses for airlines. Delayed flights also cause inconvenience and frustration for passengers. The aim of this project is to predict which flights are likely to be delayed using historical flight data and weather information. The goal is to help airlines and passengers take proactive measures to minimize the impact of delays on their operations and travel plans. By identifying which flights are at high risk of being delayed, airlines can proactively adjust their schedules, allocate resources more effectively, and inform passengers of potential delays in advance. This can help reduce the number of missed connections, improve operational efficiency, and minimize the financial impact of delays. Additionally, passengers can also benefit from this prediction model by being able to plan their travel more efficiently. By knowing in advance that their flight is likely to be delayed, they can make alternate arrangements, such as booking a different flight or making other travel arrangements, to minimize the impact of the delay on their schedule.

Using historical flight data and weather information, we can identify patterns and trends that are indicative of delays. By analyzing this data using machine learning techniques such as logistic regression and random forest classifiers, we can build a model that can accurately predict which flights are likely to be delayed.

This model can then be used by airlines to optimize their operations, allocate resources more efficiently, and minimize the financial impact of delays. Additionally, passengers can benefit by being able to plan their travel more effectively and avoid the frustration and inconvenience of unexpected delays. Overall, the development of an accurate prediction model has the potential to improve the efficiency and profitability of airlines, while also improving the travel experience for passengers. The aim of this project is to create a prediction model that can help airlines and passengers mitigate the impact of flight delays and improve the overall travel experience.

3. Data Understanding

Data understanding is an essential step in our project, and it involves getting a comprehensive understanding of the data that will be used in the project. In our case, the data includes flight information, departure and arrival times, airline information, and weather data.

The first step in data understanding is to collect the data from various sources, including airline and weather data providers. We need to obtain as much data as possible to get accurate and complete information that can be used in your predictive model.

Once we have the data, we can perform data cleaning and data preprocessing to ensure that the data is accurate, consistent, and in a usable format. This process involves removing any duplicates, missing values, or errors in the data. We may also need to transform or normalize the data to make it consistent with other data sources.

Data understanding is a crucial step in your project, and it involves collecting, cleaning, preprocessing, exploring, and analyzing the data to ensure that it is accurate, complete, and in a usable format. The insights gained from this process will form the foundation for the predictive model that we will develop in the next steps of the project.

3.1 Major factors considered to select dataset

past flight information: A predictive model must be built using past information on flight delays. The times of the flights' departure as well as any delays should be included in this data.

Weather information: Delays in flights can be significantly impacted by the weather. It's crucial to incorporate weather data into our study, such as temperature, precipitation, and wind speed.

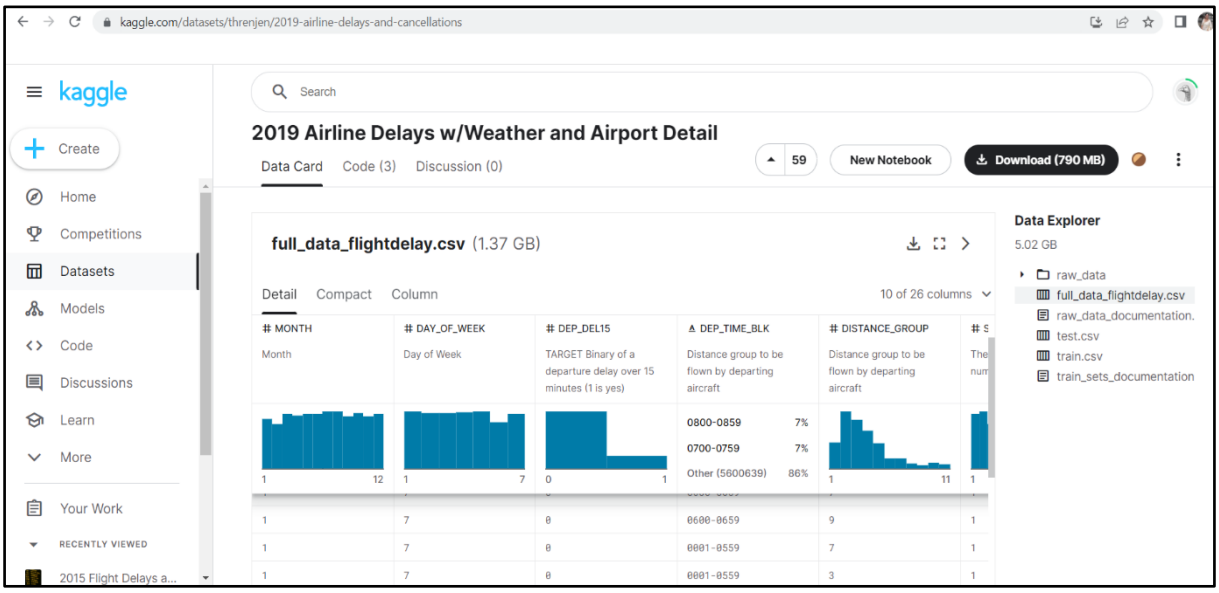
Data relevant to the airport should be included, such as the number of runways, the volume of air traffic, and the availability of ground employees, as different airports may experience distinct delay patterns.

Information about airlines: Different airlines may have different policies and processes, which may have an impact on flight delays. Include airline-specific information, such as the quantity of flights, if possible.

Time-related elements : The risk of a flight delay might vary depending on the time of day, the day of the week, and the season. For instance, there may be a higher likelihood of delays on flights during busy travel periods like holidays or the weekend.

3.2 Overview of Flight Delay dataset from Kaggle

In this project we use following dataset from **Kaggle**.



Here is a brief description of each column present in Flight Delay dataset:

- MONTH: The numerical month (1-12) of the flight.
- DAY_OF_WEEK: The numerical day of the week (1-7) of the flight.
- DEP_DEL15: A binary indicator (0/1) of whether the flight departed more than 15 minutes after the scheduled departure time.
- DEP_TIME_BLK: The time block of the scheduled departure time (e.g. "0600-0659" for departures scheduled between 6:00-6:59am).
- DISTANCE_GROUP: The distance group (in 250-mile intervals) of the flight.
- SEGMENT_NUMBER: The segment number of the flight.
- CONCURRENT_FLIGHTS: The number of other flights scheduled to depart within 2 hours of the given flight.

- NUMBER_OF_SEATS: The total number of seats available on the aircraft for the flight.
- CARRIER_NAME: The name of the airline carrier for the flight.
- AIRPORT_FLIGHTS_MONTH: The total number of flights departing from the airport in the given month.
- AIRLINE_FLIGHTS_MONTH: The total number of flights operated by the airline carrier in the given month.
- AIRLINE_AIRPORT_FLIGHTS_MONTH: The total number of flights operated by the airline carrier departing from the airport in the given month.
- AVG_MONTHLY_PASS_AIRPORT: The average monthly passenger traffic at the departing airport.
- AVG_MONTHLY_PASS_AIRLINE: The average monthly passenger traffic for the airline carrier.
- FLT_ATTENDANTS_PER_PASS: The number of flight attendants on the aircraft per available seat.
- GROUND_SERV_PER_PASS: The number of ground service personnel on the aircraft per available seat.
- PLANE_AGE: The age of the aircraft in years.
- DEPARTING_AIRPORT: The three-letter code of the departing airport.
- LATITUDE: The latitude coordinate of the departing airport.
- LONGITUDE: The longitude coordinate of the departing airport.
- PREVIOUS_AIRPORT: The three-letter code of the previous airport for the flight.
- PRCP: The total precipitation (in inches) for the day of the flight at the departing airport.
- SNOW: The total snowfall (in inches) for the day of the flight at the departing airport.
- SNWD: The total snow depth (in inches) for the day of the flight at the departing airport.
- TMAX: The maximum temperature (in degrees Fahrenheit) for the day of the flight at the departing airport.
- AWND: The average daily wind speed (in miles per hour) for the day of the flight at the departing airport.

4. Data Preparation

Data preparation is a critical step in any data analysis project. In the case of flight data, it is essential to ensure that the data is clean, accurate, and formatted correctly before importing it into MongoDB. This process involves several steps, such as identifying missing or erroneous data, removing duplicates, and transforming data types and values as necessary.

To design an appropriate data model and schema for storing the flight data in MongoDB, it is essential to consider the specific requirements of the analysis. This may involve normalizing or denormalizing the data, defining relationships between collections, and establishing indexes for efficient querying.

Once the data is cleaned and the schema is designed, it can be imported into MongoDB using tools such as PyMongo or MongoDB Compass. This process involves mapping the data to the defined schema and loading it into the database.

MongoDB's aggregation framework is a powerful tool that allows for data processing and analysis. It enables the user to perform complex operations on the data, such as filtering, sorting, grouping, and joining. The aggregation pipeline allows for data transformations, such as merging, splitting, and reshaping data, making it more accessible for analysis.

As data preparation is a crucial step in the data analysis process, especially for flight data. It involves identifying missing or erroneous data, removing duplicates, and transforming data types and values as necessary. The appropriate data model and schema must be designed to store the flight data in MongoDB, considering the specific requirements of the analysis. Finally, MongoDB's aggregation framework can be used for powerful data transformations and querying capabilities.

4.1 Processing data steps

1. Import necessary libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.utils.validation import column_or_1d
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
```

2.Create a database and collection

```
from pymongo import MongoClient

# Set up the MongoDB client
client = MongoClient("mongodb+srv://enoch:admin@mycluster.ixjzbgw.mongodb.net/test")

# Access the database and collection
db = client.get_database("Flight_DB")
collection = db.get_collection("Flight_prediction")
```

3.Process data- Convert Object ID to String and create Data frame

```
# Convert ObjectId to string
data = list(collection.find())
for d in data:
    d['_id'] = str(d['_id'])

# Create DataFrame
df = pd.DataFrame(data)

# Print the dataframe
print(df)
```

	_id	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK
0	643782950014c5f5fcbb4e8b	2	6	0	1400-1459
1	643782950014c5f5fcbb4e85	9	7	0	0001-0559
2	643782950014c5f5fcbb4e86	11	3	0	0900-0959
3	643782950014c5f5fcbb4e87	4	5	1	0900-0959
4	643782950014c5f5fcbb4e91	8	5	0	1000-1059
...	...	...	...	...	...
299995	643783b60014c5f5fcbfe242	NaN	NaN	NaN	NaN
299996	643783b60014c5f5fcbfe245	NaN	NaN	NaN	NaN
299997	643783b60014c5f5fcbfe249	NaN	NaN	NaN	NaN
299998	643783b60014c5f5fcbfe250	NaN	NaN	NaN	NaN
299999	643783b60014c5f5fcbfe25d	NaN	NaN	NaN	NaN

DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS
2	5	22	143
2	1	2	50
9	2	20	180
7	2	23	169
11	2	37	170
...	...	...	...
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN
NaN	NaN	NaN	NaN

	CARRIER_NAME	...	PLANE_AGE
	Southwest Airlines Co.	...	13
	American Eagle Airlines Inc.	...	14
	Delta Air Lines Inc.	...	2
	United Air Lines Inc.	...	21
	United Air Lines Inc.	...	29
	...	...	...
	NaN	...	NaN
	NaN	...	NaN
	NaN	...	NaN
	NaN	...	NaN
	NaN	...	NaN

4. Create a dictionary with old and new column names and Rename columns using the dictionary

```python
# Create a dictionary with old and new column names
columns_dict = {
    "_id":"ID",
    "MONTH": "month",
    "DAY_OF_WEEK": "day_of_week",
    "DEP_DEL15": "dep_delayed_15min",
    "DEP_TIME_BLK": "dep_time_block",
    "DISTANCE_GROUP": "distance_group",
    "SEGMENT_NUMBER": "segment_number",
    "CONCURRENT_FLIGHTS": "concurrent_flights",
    "NUMBER_OF_SEATS": "number_of_seats",
    "CARRIER_NAME": "carrier_name",
    "AIRPORT_FLIGHTS_MONTH": "airport_flights_month",
    "AIRLINE_FLIGHTS_MONTH": "airline_flights_month",
    "AIRLINE_AIRPORT_FLIGHTS_MONTH": "airline_airport_flights_month",
    "AVG_MONTHLY_PASS_AIRPORT": "avg_monthly_pass_airport",
    "AVG_MONTHLY_PASS_AIRLINE": "avg_monthly_pass_airline",
    "FLT_ATTENDANTS_PER_PASS": "flt_attendants_per_pass",
    "GROUND_SERV_PER_PASS": "ground_serv_per_pass",
    "PLANE_AGE": "plane_age",
    "DEPARTING_AIRPORT": "departing_airport",
    "LATITUDE": "latitude",
    "LONGITUDE": "longitude",
    "PREVIOUS_AIRPORT": "previous_airport",
    "PRCP": "precipitation",
    "SNOW": "snow",
    "SNWD": "snow_depth",
    "TMAX": "max_temperature",
    "AWND": "avg_wind_speed"
}

# Rename the columns using the dictionary
df2 = df.rename(columns=columns_dict)
```

ID	month	day_of_week	dep_delayed_15min	dep_time_block	distance_group	segment_number	concurrent_flights	number_of_seats	carrier_name	...	plane_age	departing_airport	latitude	longit
64e8b	2	6	0	1400-1459	2	5	22	143	Southwest Airlines Co.	...	13	Los Angeles International	33.942	-118
64e85	9	7	0	0001-0559	2	1	2	50	American Eagle Airlines Inc.	...	14	Memphis International	35.05	-89
64e86	11	3	0	0900-0959	9	2	20	180	Delta Air Lines Inc.	...	2	Detroit Metro Wayne County	42.217	-83
64e87	4	5	1	0900-0959	7	2	23	169	United Air Lines Inc.	...	21	Newark Liberty International	40.696	-74
64e91	8	5	0	1000-1059	11	2	37	170	United Air Lines Inc.	...	29	San Francisco International	37.619	-122

5.Fill all missing values with Zero

```python
df2.fillna(0, inplace=True)
```

6.count the number of null values in each column and print the number of missing values in each column

```
# count the number of null values in each column
missing_values = df2.isnull().sum()

# print the number of missing values in each column
print(missing_values)
```

```
ID                                0
month                             0
day_of_week                       0
dep_delayed_15min                 0
dep_time_block                    0
distance_group                    0
segment_number                    0
concurrent_flights                0
number_of_seats                   0
carrier_name                      0
airport_flights_month             0
airline_flights_month             0
airline_airport_flights_month     0
avg_monthly_pass_airport          0
avg_monthly_pass_airline          0
flt_attendants_per_pass           0
ground_serv_per_pass              0
plane_age                         0
departing_airport                 0
latitude                          0
longitude                         0
previous_airport                  0
precipitation                     0
snow                              0
snow_depth                        0
max_temperature                   0
avg_wind_speed                    0
dtype: int64
```

7. Define a dictionary with the new column names and their data types

```
# Define a dictionary with the new column names and their data types
columns_dict = {
    'ID': 'str',
    'airport_flights_month': 'int',
    'airline_flights_month': 'int',
    'airline_airport_flights_month': 'int',
    'avg_monthly_pass_airline': 'int',
    'avg_monthly_pass_airport': 'int',
    'carrier_name': 'str',
    'concurrent_flights': 'int',
    'day_of_week': 'int',
    'departing_airport': 'str',
    'dep_delayed_15min': 'int',
    'dep_time_block': 'str',
    'distance_group': 'int',
    'flt_attendants_per_pass': 'float',
    'ground_serv_per_pass': 'float',
    'latitude': 'float',
    'longitude': 'float',
    'month': 'int',
    'number_of_seats': 'int',
    'plane_age': 'float',
    'precipitation': 'float',
    'previous_airport': 'str',
    'segment_number': 'int',
    'snow': 'float',
    'snow_depth': 'float',
    'max_temperature': 'float',
    'avg_wind_speed': 'float'
}

# Loop through the columns and update the data types
for c in columns_dict.keys():
    df2[c] = df2[c].astype(columns_dict[c])
```

8. Loop through the columns and update the data types

```
# Print the schema to check the data types
df2.dtypes
```

```
ID                                object
month                              int64
day_of_week                        int64
dep_delayed_15min                  int64
dep_time_block                    object
distance_group                     int64
segment_number                     int64
concurrent_flights                 int64
number_of_seats                    int64
carrier_name                      object
airport_flights_month              int64
airline_flights_month              int64
airline_airport_flights_month      int64
avg_monthly_pass_airport           int64
avg_monthly_pass_airline           int64
flt_attendants_per_pass          float64
ground_serv_per_pass             float64
plane_age                        float64
departing_airport                 object
latitude                         float64
longitude                        float64
previous_airport                  object
precipitation                    float64
snow                             float64
snow_depth                       float64
max_temperature                  float64
avg_wind_speed                   float64
dtype: object
```

9. Select a subset of columns from the Data Frame "df2" and assigns it to a new Data Frame called "df_subset"

```
df_subset = df2[["_id","airline_airport_flights_month","airline_flights_month", "carrier_name","airport_flights_month","avg_monthly_pass_airline","avg_monthly_pass_airport","concurr
df_subset.head()
```

Python

	airline_airport_flights_month	airline_flights_month	airport_flights_month	avg_monthly_pass_airline	avg_monthly_pass_airport	carrier_name	concurrent_flights	day_of_week	departing_airport	dep_
0	3015	94922	15964	13382999	2780593	Southwest Airlines Co.	22	6	Los Angeles International	
1	222	26473	2050	1204766	191927	American Eagle Airlines Inc.	2	7	Memphis International	
2	4894	79989	13199	12460183	1486066	Delta Air Lines Inc.	20	3	Detroit Metro Wayne County	
3	4919	51763	11588	8501631	1708599	United Air Lines Inc.	23	5	Newark Liberty International	
4	6234	55706	16527	8501631	1908862	United Air Lines Inc.	37	5	San Francisco International	

5 rows × 26 columns

10. Create a new column "dep_time_block_int_label" that maps the categorical variable "dep_time_block" to numerical values.

```
value_counts = df_subset["dep_time_block"].value_counts().sort_index(ascending=True)
value_counts
value_to_num = {value: i for i, value in enumerate(value_counts.index)}
# Create a new column in the DataFrame containing the unique number for each value
df_subset["dep_time_block_int_label"] = df_subset["dep_time_block"].map(value_to_num)
df_subset.head()
```

eparting_airport	dep_delayed_15min	...	number_of_seats	plane_age	precipitation	previous_airport	segment_number	snow	snow_depth	max_temperature	avg_wind_speed	dep_time_block_int_label
Los Angeles International	0	...	143	13.0	1.45	Stapleton International	5	0.0	0.0	60.0	11.41	10
Memphis International	0	...	50	14.0	0.00	NONE	1	0.0	0.0	94.0	7.61	1
Detroit Metro Wayne County	0	...	180	2.0	0.55	Logan International	2	0.0	0.0	56.0	21.70	5
Newark Liberty International	1	...	169	21.0	0.81	Los Angeles International	2	0.0	0.0	62.0	9.17	5
San Francisco International	0	...	170	29.0	0.00	Logan International	2	0.0	0.0	73.0	13.65	6

11. Create new column "carrier_name_int_label", which maps the categorical variable "carrier_name" to unique numerical values.

```
#Here we are taking the unique values from "Carrier Name" an assigning a value to them.

value_counts = df_subset["carrier_name"].value_counts().sort_index(ascending=True)
value_counts
```

```
# Create a dictionary to map each unique value to a unique number
value_to_num = {value: i for i, value in enumerate(value_counts.index)}

# Create a new column in the DataFrame containing the unique number for each value
df_subset["carrier_name_int_label"] = df_subset["carrier_name"].map(value_to_num)
df_subset.head()
```

[27] Python

ng_airport	dep_delayed_15min	...	plane_age	precipitation	previous_airport	segment_number	snow	snow_depth	max_temperature	avg_wind_speed	dep_time_block_int_label	carrier_name_int_label
.os Angeles ıternational	0	...	13.0	1.45	Stapleton International	5	0.0	0.0	60.0	11.41	10	15
Memphis ıternational	0	...	14.0	0.00	NONE	1	0.0	0.0	94.0	7.61	1	4
:troit Metro yne County	0	...	2.0	0.55	Logan International	2	0.0	0.0	56.0	21.70	5	7
vark Liberty ıternational	1	...	21.0	0.81	Los Angeles International	2	0.0	0.0	62.0	9.17	5	17
n Francisco ıternational	0	...	29.0	0.00	Logan International	2	0.0	0.0	73.0	13.65	6	17

12. Create a new column called "departing_airport_int_label", which maps the categorical variable "departing_airport" to unique numerical values.

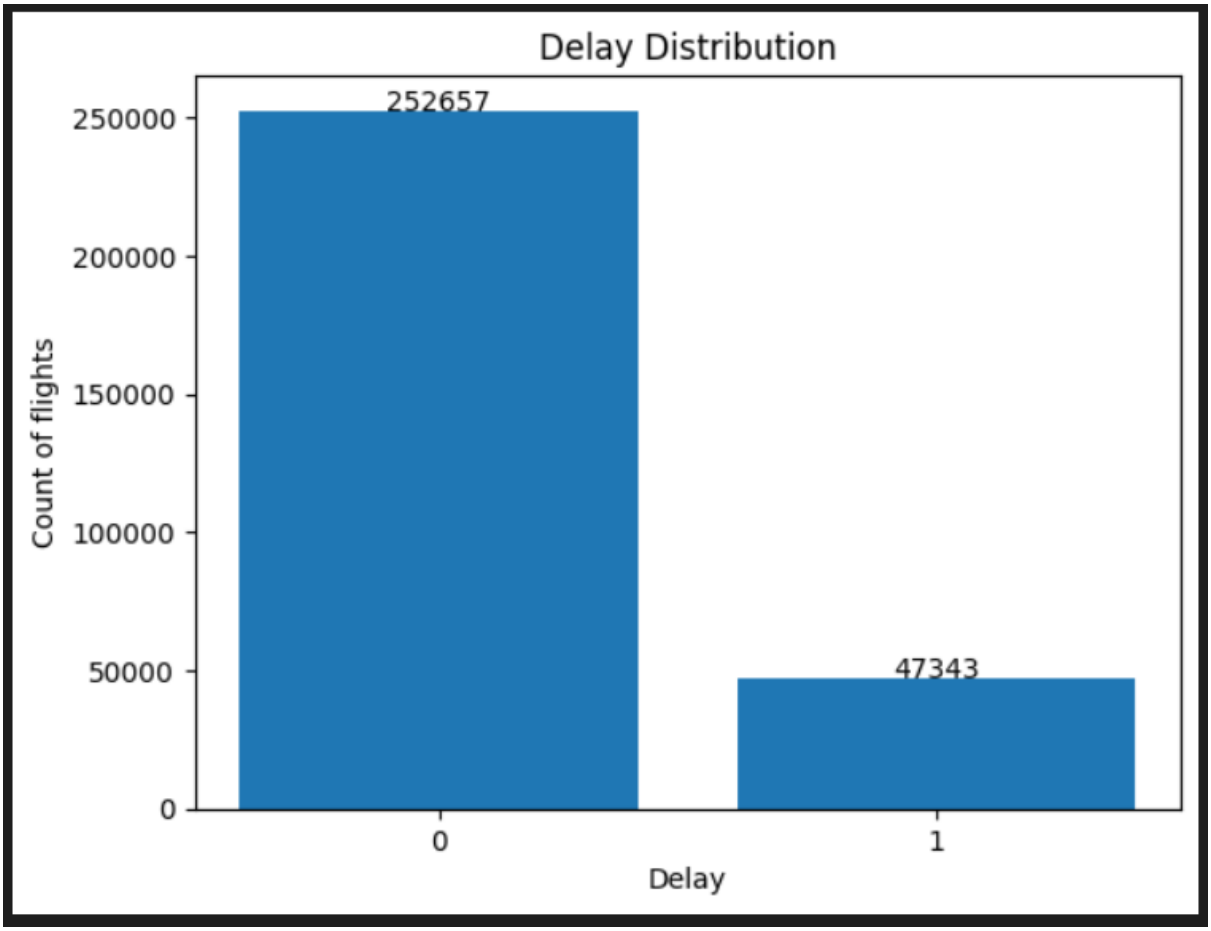
```
# Get the value counts of each unique value in "DEPARTING_AIRPORT"

#We want to use this as a feature but we have to convert it from strings to an int
value_counts = df_subset["departing_airport"].value_counts().sort_index(ascending=True)
value_counts

#Here we are taking the unique values from "Departing Airport" an assigning a value to them.

# Create a dictionary to map each unique value to a unique number
value_to_num = {value: i for i, value in enumerate(value_counts.index)}

# Create a new column in the DataFrame containing the unique number for each value
df_subset["departing_airport_int_label"] = df_subset["departing_airport"].map(value_to_num)
df_subset.head()
```

delayed_15min	...	precipitation	previous_airport	segment_number	snow	snow_depth	max_temperature	avg_wind_speed	dep_time_block_int_label	carrier_name_int_label	departing_airport_int_label
0	...	1.45	Stapleton International	5	0.0	0.0	60.0	11.41	10	15	43
0	...	0.00	NONE	1	0.0	0.0	94.0	7.61	1	4	47
0	...	0.55	Logan International	2	0.0	0.0	56.0	21.70	5	7	18
1	...	0.81	Los Angeles International	2	0.0	0.0	62.0	9.17	5	17	53
0	...	0.00	Logan International	2	0.0	0.0	73.0	13.65	6	17	79

13. Create a plot shows the count of flights for each category ("delayed" or "not delayed") on the x-axis, and the number of flights on the y-axis.

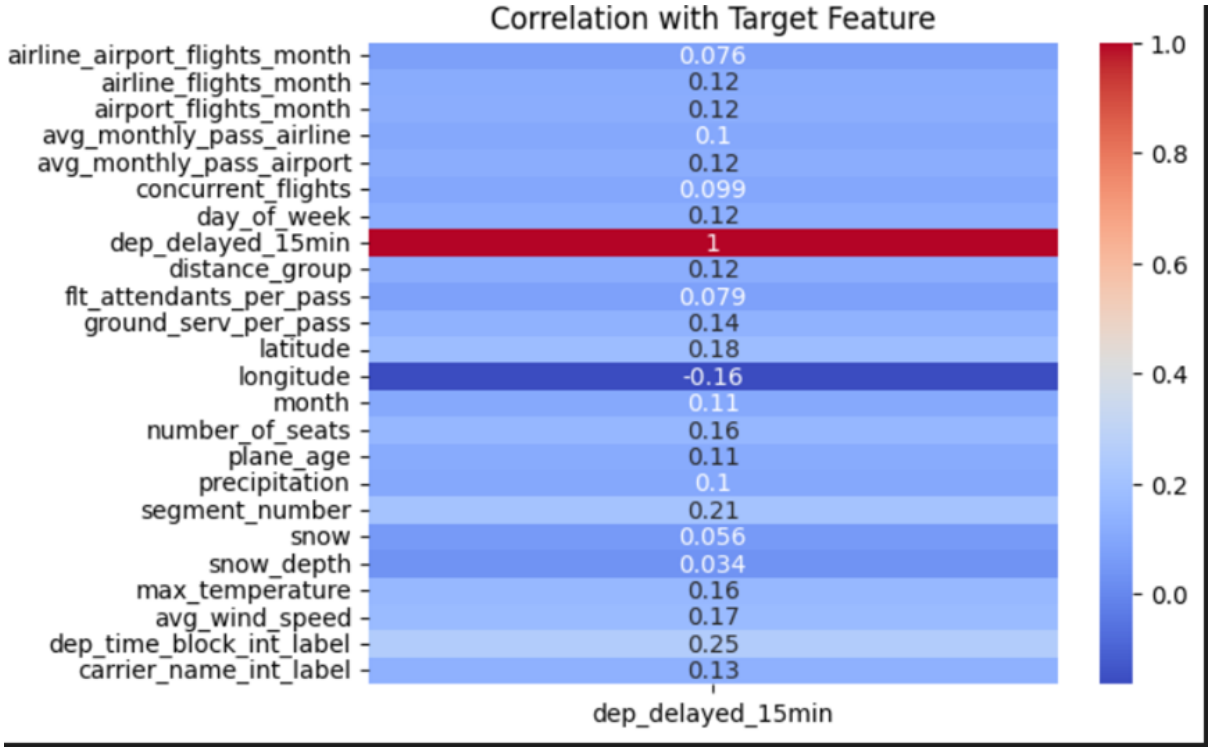
```
import matplotlib.pyplot as plt

values = df_subset["dep_delayed_15min"].value_counts()

# Create a new figure
plt.figure()

# Add a bar plot of the data
plt.bar(values.index, values.values)

# Add the value of each bar to the plot
for i, v in enumerate(values.values):
    plt.text(i, v, str(v), ha='center')

# Set the x-axis tick labels
plt.xticks(list(values.index))

# Add a title and axis labels
plt.title("Delay Distribution")
plt.xlabel("Delay")
plt.ylabel("Count of flights")

# Show the plot
plt.show()
```

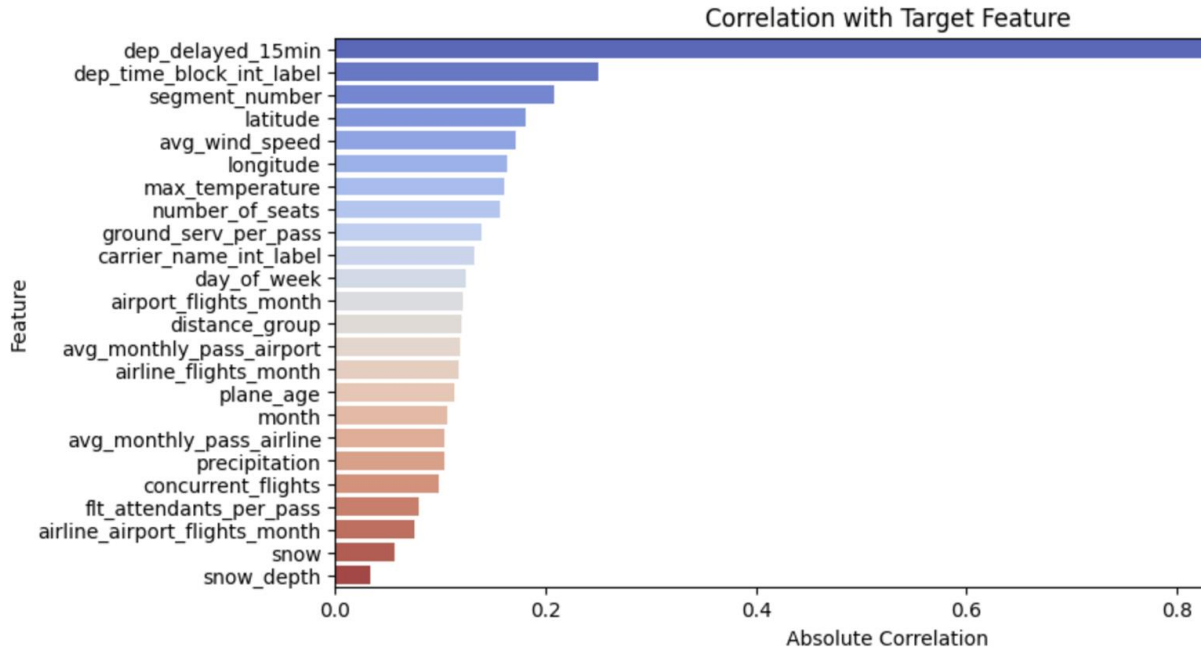
The code creates a bar plot of the distribution of delays in the "dep_delayed_15min" column of a pandas DataFrame named "df_subset". It uses matplotlib to create the plot, with the x-axis representing the different delay categories (0 and 1, for on-time or delayed, respectively) and the y-axis representing the count of fli



14. Generates a heatmap that displays the correlation of each feature in the df_subset DataFrame with the target feature (dep_delayed_15min).

```
#Here we are trying to find a correlation between our subsetted features
#and our target fetaure

# Calculate the correlation matrix between all pairs of features
corr_matrix = df_subset.corr()
# Get the correlation of each feature with the target feature
target_corr = corr_matrix["dep_delayed_15min"].iloc[:-1]
# Plot the target correlation as a heatmap
sns.heatmap(target_corr.to_frame(), annot=True, cmap='coolwarm')
# Add a title
plt.title("Correlation with Target Feature")
# Show the plot
plt.show()
```





This visualize the correlation between all features and a target feature, and here we can identify which features are most strongly associated with the target feature. In here dep_delayed_15min, dep_time_block_int_lable, segment_number are the features that directly affected to target.

5. Modeling

The process of analyzing the data stored in MongoDB involves using MongoDB's aggregation pipeline. This pipeline is a set of stages that can be used to perform complex data analysis tasks, such as grouping, filtering, and sorting. By applying these stages to the flight data, we can identify patterns and trends that can help us understand why flights are delayed and what factors are most likely to contribute to delays.

Once we have analyzed the data and identified key patterns and trends, we can then use this information to build a predictive model using Python and related tools. This model will use historical flight data and weather information to predict which flights are likely to be delayed. By using machine learning algorithms such as logistic regression and random forest classifier, we can train the model to recognize patterns and make predictions based on the input data.

The goal of building this predictive model is to help airlines and passengers take proactive measures to minimize the impact of flight delays on their operations and travel plans. By identifying which flights are at high risk of being delayed, airlines can adjust their schedules and allocate resources more effectively, while passengers can make alternate arrangements to minimize the impact of the delay on their schedule.

Overall, the process of analyzing flight data and building a predictive model involves a combination of data analysis and machine learning techniques. By leveraging these tools, we can gain insights into flight delays and develop models that can help mitigate their impact on airlines and passengers alike.

Two machine learning models will be used to predict flight delays: **logistic regression** and **random forest classifier**. Logistic regression is a statistical method that models the relationship between a binary dependent variable (in this case, whether a flight is delayed or not) and one or more independent variables (such as weather data, airline carrier, etc.). Random forest classifier, on the other hand, is an ensemble learning method that creates multiple decision trees and combines their predictions to make a final prediction. Both models will be evaluated based on their accuracy in predicting flight delays.

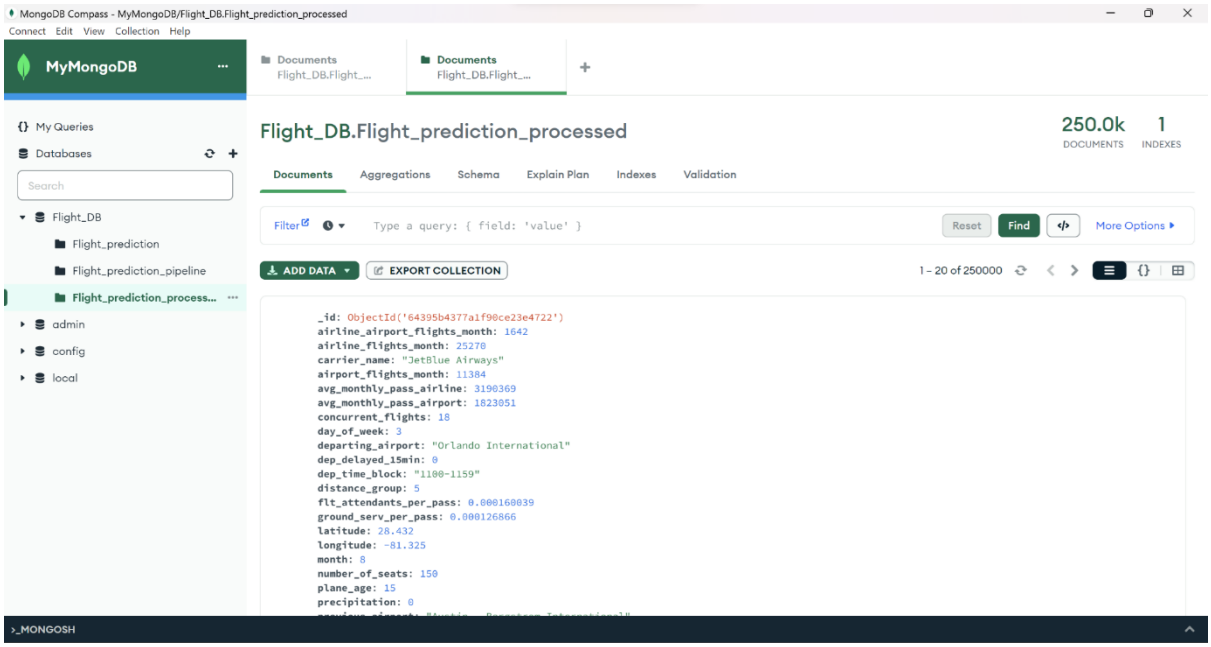
5.1 Modeling data steps

1. Import processed data to MongoDB

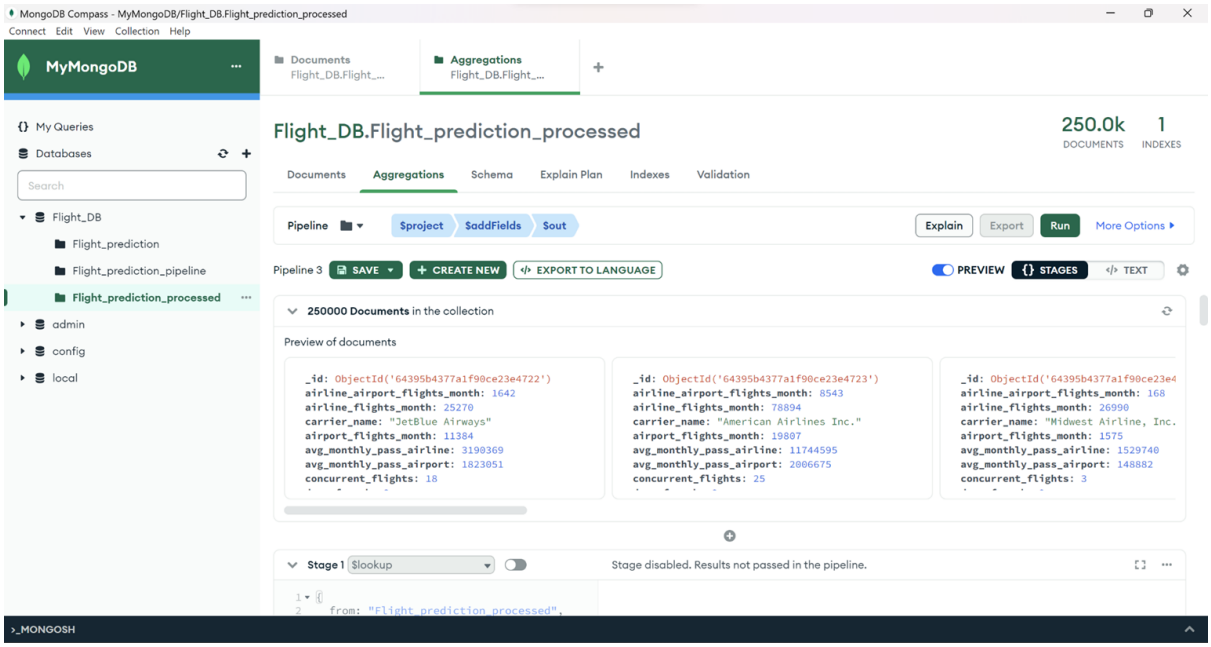
```
# Set up the MongoDB client
client = MongoClient("mongodb+srv://enoch:admin@mycluster.ixjzbgw.mongodb.net/test")

# Access the database and collection
db = client.get_database("Flight_DB")
collection = db.get_collection("Flight_prediction_processed")

# Load your Pandas DataFrame into a dictionary format
data = df_subset.to_dict(orient='records')

# Insert the data into the MongoDB collection
collection.insert_many(data)
```

```
# Convert ObjectId to string
data = list(collection.find())
for d in data:
    d['_id'] = str(d['_id'])

# Create DataFrame
df_pipeline = pd.DataFrame(data)

# Print the dataframe
df_pipeline.head()
```

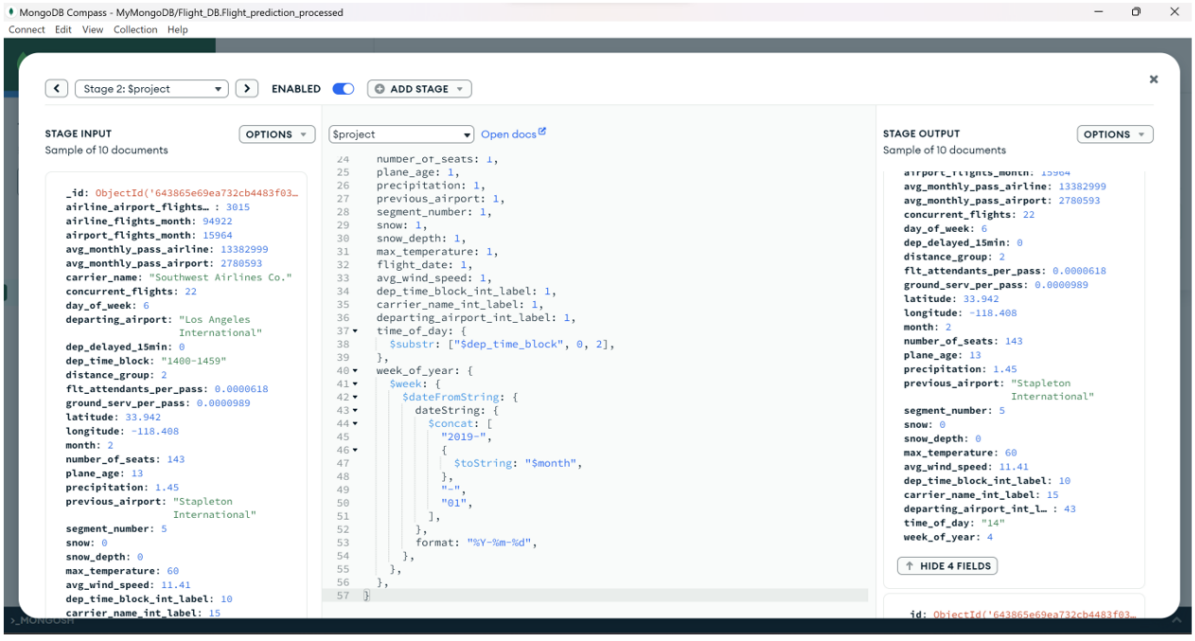
2. View of Flight_predcition_processed Collection, after Data has been processed



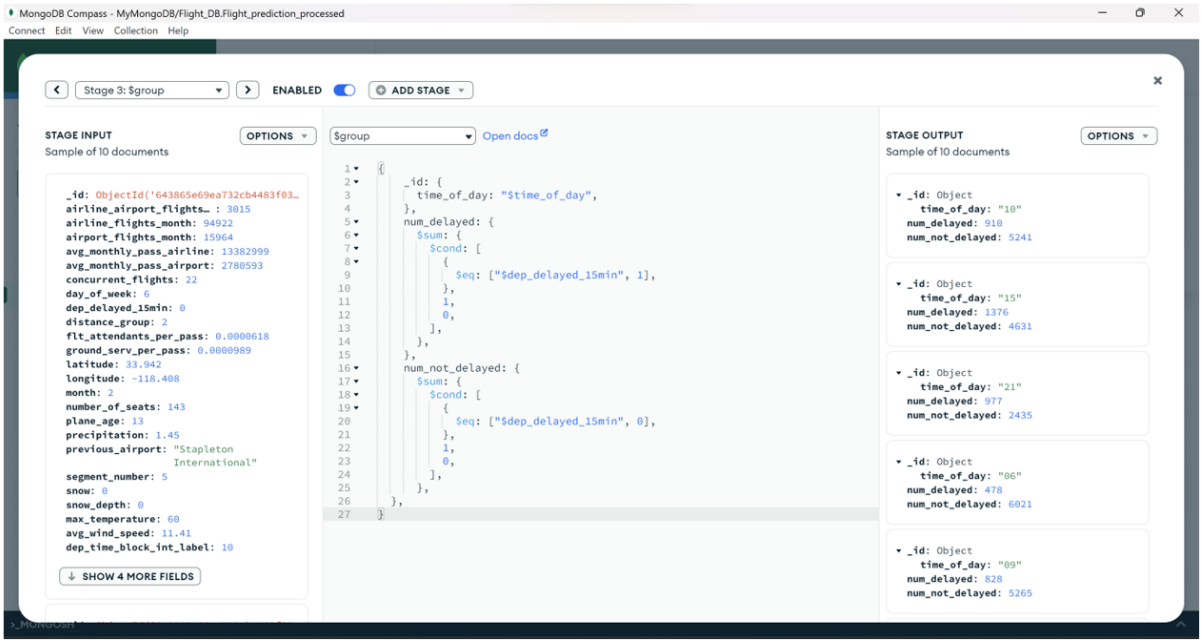
3. Using MongoDB aggregation pipeline



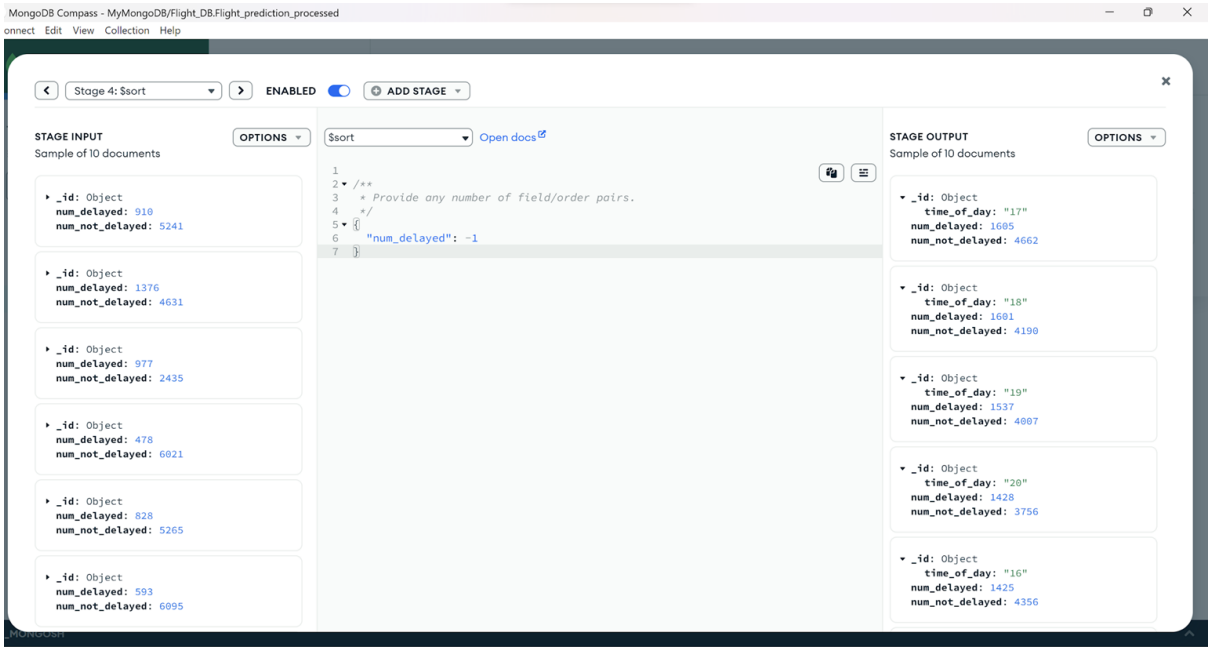
4. Stage 1 of the pipeline



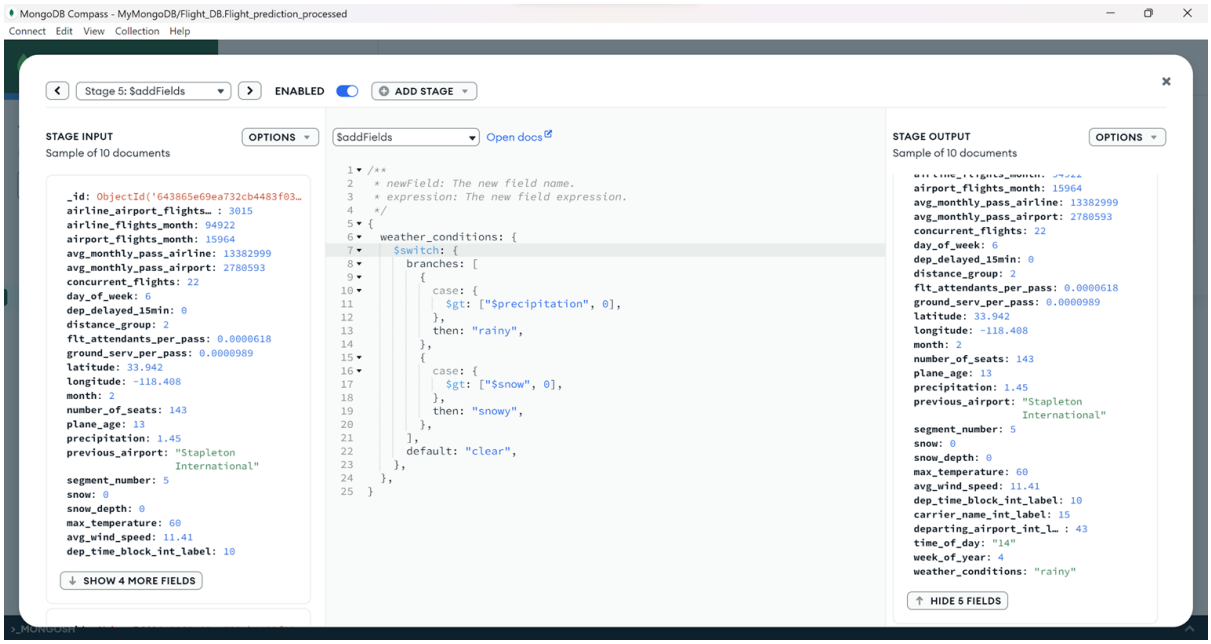
5.Stage 2 of the pipeline



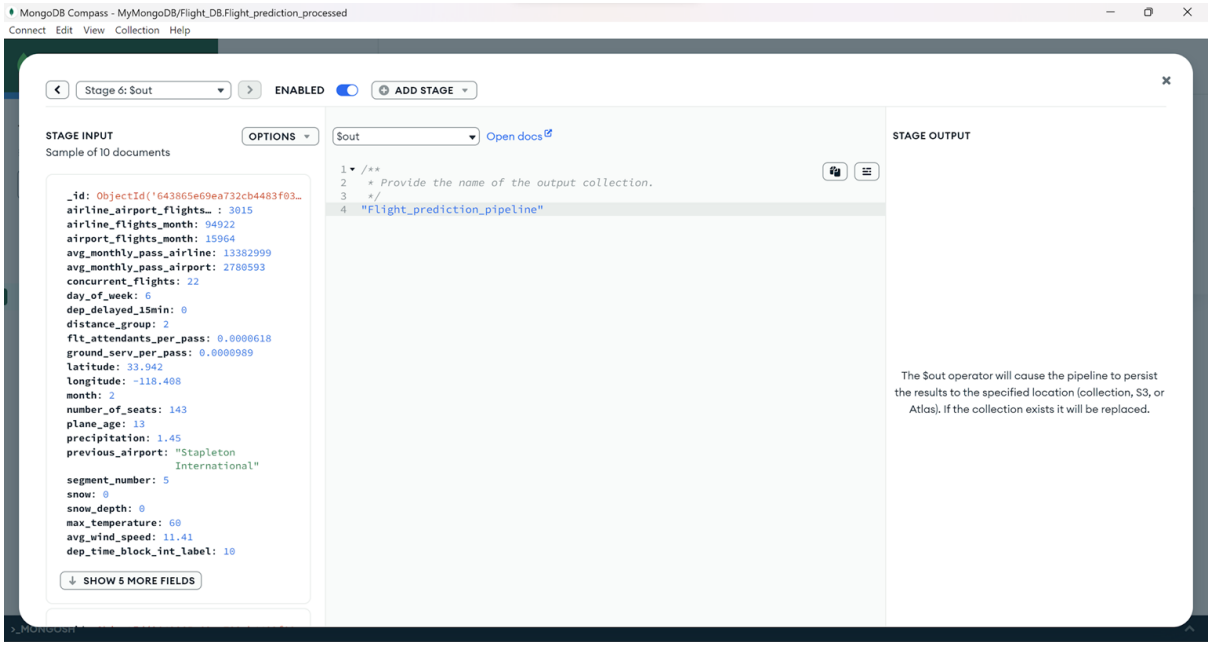
6. Stage 3 of the pipeline



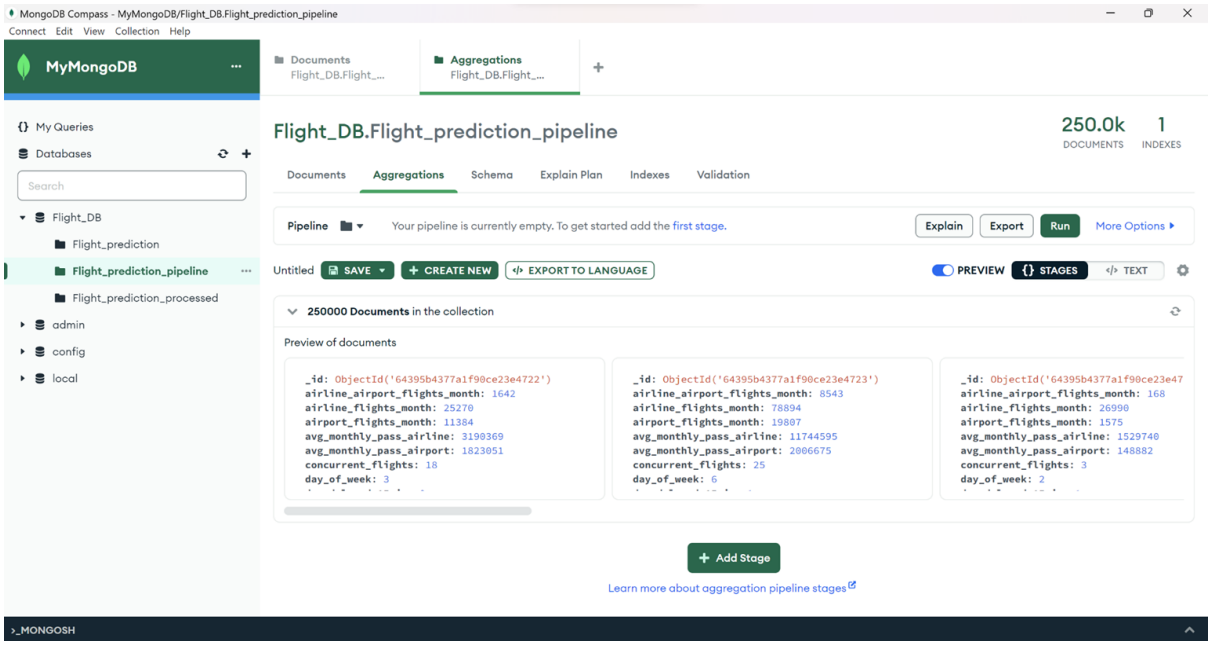
7. Stage 4 of the pipeline



8. Stage 5 of the pipeline



9. View of Flight_prediction_pipeline collection



10. Connect to the MongoDB database and retrieves data from a collection, converting it into a pandas DataFrame for further analysis or machine learning tasks

```
# Set up the MongoDB client
client = MongoClient("mongodb+srv://bigelk:bimZmiran@tutorial.wqhr5nb.mongodb.net/test")

# Access the database and collection
db = client.get_database("Flight_DB")
collection = db.get_collection("Flight_prediction_pipeline")

# Convert ObjectId to string
data = list(collection.find())
for d in data:
    d['_id'] = str(d['_id'])

# Create DataFrame
df_pipeline = pd.DataFrame(data)

# Print the dataframe
df_pipeline.head()
```

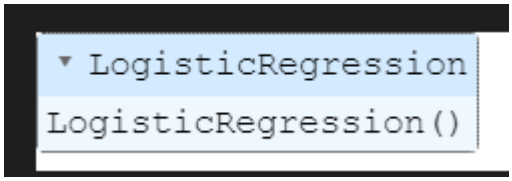
✓ 3.3s

	_id	airline_airport_flights_month	airline_flights_month	airport_flights_month	avg_monthly_pass_airline	avg_monthly_pass_airport	concurrent_flights	day_of_week	dep_delay
0	6438d3e1824330253b0add9d	1642	25270	11384	3190369	1823051	18	3	
1	6438d3e1824330253b0add9e	8543	78894	19807	11744595	2006675	25	6	
2	6438d3e1824330253b0add9f	168	26990	1575	1529740	148882	3	2	
3	6438d3e1824330253b0adda0	3251	17869	22775	1191889	2907365	42	6	
4	6438d3e1824330253b0adda1	1530	17201	13019	2688839	1823051	30	1	

5.2 Logistic Regression Modeling

1. Assigning X and Y: The feature variables are assigned to x and the target variable to y.
2. Splitting the data: The data is split into training and testing sets using the train_test_split function from the sklearn library. This allows the model to be trained on a subset of the data and evaluated on a separate subset.
3. Instantiating the model: A logistic regression model is instantiated using the LogisticRegression class from the sklearn library.
4. Fitting the model: The logistic regression model is fit to the training data using the fit method. This step involves finding the coefficients that best fit the training data and using them to make predictions on new data.

5. The following code generates predictions for the target variable based on the test data using the logistic regression model trained in previous code.



6. The predictions are stored in a pandas DataFrame along with the actual values of the target variable from the test data.
7. This DataFrame can be used to evaluate the accuracy of the model's predictions on the test data.

```
#Here we are combining Data predictions and

#Predict using the test data
predictions = model.predict(test_data)

#creating a DataFrame with Test Target and Predictions
result = pd.DataFrame(predictions, columns=['prediction'])
result["target"] = test_target.values
```

✓ 0.0s

```
result.head(1000)
```

✓ 0.0s

	prediction	target
0	0	0
1	0	0
2	0	0
3	0	0
4	0	0
...	...	...
995	0	1
996	0	0
997	0	0
998	0	0
999	0	1

1000 rows × 2 columns

5.3 Random Forest Classifier Modeling

1. We extracted the hour from the time_of_day column using a custom function and applied it to the dataframe.

```
# Define a function to extract the hour from the time_of_day string
def get_hour(time_str):
    return int(time_str)

# Apply the get_hour function to the time_of_day column
df_pipeline['time_of_day'] = df_pipeline['time_of_day'].apply(get_hour)

# One-hot encode the weather_conditions feature
df_pipeline = pd.get_dummies(df_pipeline, columns=['weather_conditions'])
```

✓ 0.1s

2. We encoded the weather_conditions column, creating new binary columns for each possible value.

```
# Define the feature columns
feature_cols = ['concurrent_flights', 'day_of_week', 'distance_group', 'flt_attendants_per_pass',
                'ground_serv_per_pass', 'month', 'number_of_seats', 'precipitation', 'snow', 'snow_depth',
                'max_temperature', 'avg_wind_speed', 'dep_time_block_int_label', 'carrier_name_int_label',
                'departing_airport_int_label', 'week_of_year', 'time_of_day', 'weather_conditions_clear',   'weather_conditions_rainy', 'weather_conditions_snowy']

# Define the target column
target_col = 'dep_delayed_15min'

# Create the X and y variables
X = df_pipeline[feature_cols]
y = df_pipeline[target_col]
```

✓ 0.0s

3. We defined the feature columns and target column for our predictive model, which we split into training and testing sets using the train_test_split function.

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

✓ 0.0s

4. We trained a RandomForestClassifier model with 100 decision trees on the training set and made predictions on the test set.

```
# Train a Random Forest classifier
clf = RandomForestClassifier(n_estimators=100, random_state=42)
clf.fit(X_train, y_train)
```

✓ 7.8s

```
RandomForestClassifier
RandomForestClassifier(random_state=42)
```

6. Evaluation

Evaluation is a crucial part of the CRISP-DM process to assess the accuracy and effectiveness of a model for predicting flight delays. In this case, the predicted outcomes are compared with the actual outcomes to determine the performance of the model. Evaluation techniques used for flight delay prediction may include confusion matrices, ROC AUC, precision, recall, and F1-score. By evaluating the model, it is possible to determine if it meets the required standards and can accurately predict flight delays. This helps to identify areas for improvement and refinement to achieve better results in terms of predicting flight delays.

6.1 Evaluation of Logistic Regression Model

6.1.1 Evaluation Matrices

```
#Testing the Accuracy of the prediction
accuracy = accuracy_score(test_target, predictions) * 100
precision = precision_score(test_target, predictions) * 100
recall = recall_score(test_target, predictions) * 100
f1 = f1_score(test_target, predictions) * 100
roc_auc = roc_auc_score(test_target, predictions) * 100
conf_matrix = confusion_matrix(test_target, predictions)

print("Accuracy: %.2f%%" % accuracy)
print("Precision: %.2f%%" % precision)
print("Recall: %.2f%%" % recall)
print("F1 Score: %.2f%%" % f1)
print("ROC AUC Score: %.2f%%" % roc_auc)
print("Confusion Matrix:\n", conf_matrix)
```

```
Accuracy: 81.00%
Precision: 49.59%
Recall: 1.92%
F1 Score: 3.69%
ROC AUC Score: 50.73%
Confusion Matrix:
 [[40318   185]
 [ 9315   182]]
```

	Predicted Negatives	Predicted Positive
Actual Negative	40267	177
Actual Positive	9383	173

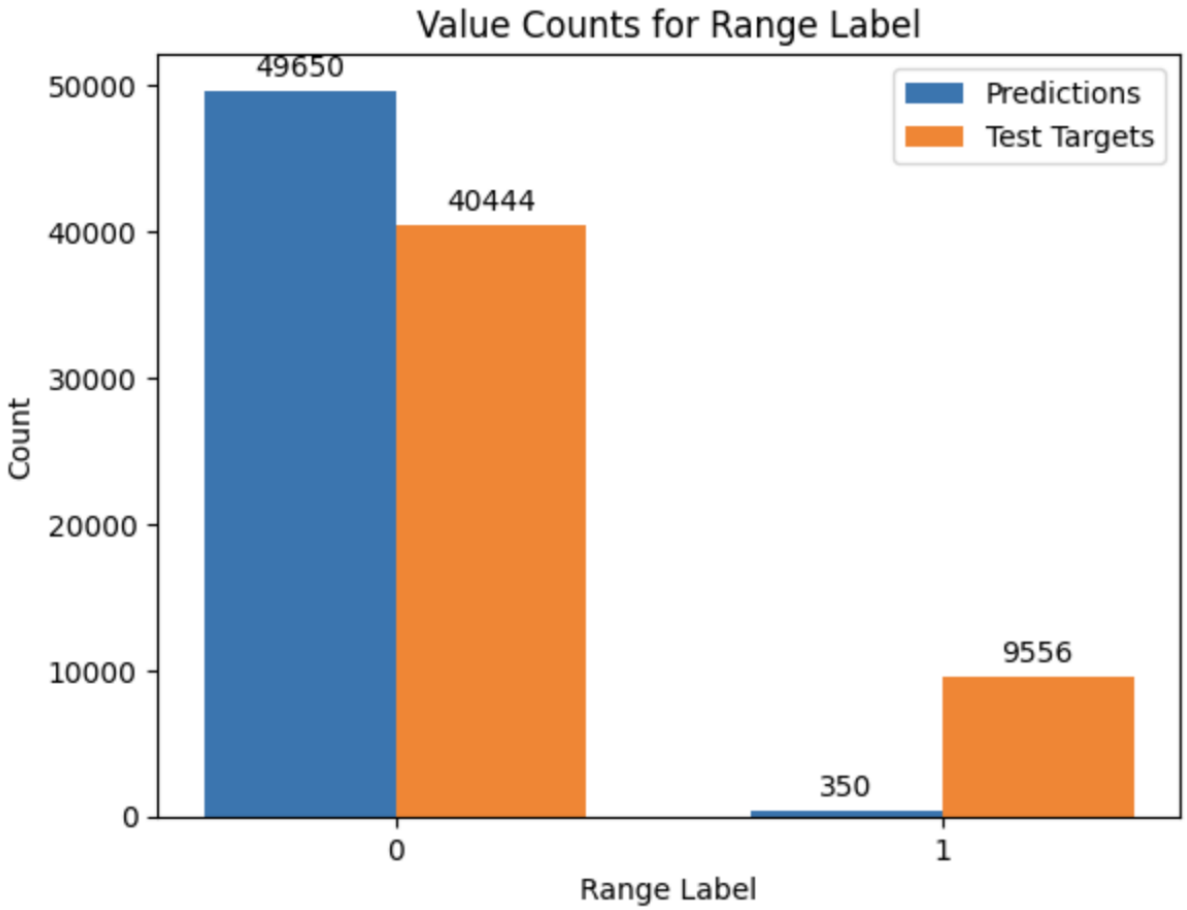
The accuracy of the model is 80.88%, which means that 80.88% of the predictions made by the model were correct. The precision score of 49.43% indicates that only about half of the positive predictions made by the model were actually true positives. The recall score of 1.81% indicates that the model correctly identified only a very small proportion of the actual positive cases. The F1 score, which takes into account both precision and recall, is 3.49%. The ROC AUC score of 50.69% indicates that the model's performance is only slightly better than random guessing. The confusion matrix shows that the model correctly classified 40267 negative cases and 173 positive cases, but incorrectly classified 9383 positive cases as negative and 177 negative cases as positive. Overall, the model's performance is poor and may require further refinement to improve its accuracy and predictive power.

6.1.2 Logistic Regression prediction Bar Chart

```
# -----> Creating a bar chart to display Predictions and Test Targe

# Sample data
df_validate = result["prediction"].value_counts()
df_validate2 = test_target.value_counts()

# Get x and y values for the first set of bars
x = np.arange(len(df_validate.index))
y1 = df_validate.values

# Get x and y values for the second set of bars
y2 = df_validate2.values

# Set the width of each bar
width = 0.35

# Create the bar chart with two sets of bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, y1, width, label='Predictions')
rects2 = ax.bar(x + width/2, y2, width, label='Test Targets')

# Add value labels to the bars
for rect in rects1:
    height = rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')

for rect in rects2:
    height = rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
```

```
                textcoords="offset points",
                ha='center', va='bottom')

# Add labels and a title to the plot
plt.xlabel('Range Label')
plt.ylabel('Count')
plt.title('Value Counts for Range Label')

# Set the x-axis tick labels to be the range labels
plt.xticks(x, df_validate.index)

# Add a legend to the plot
plt.legend()

# Display the plot
plt.show()
✓ 0.1s
```

- We create a bar chart to compare the predictions made by our model with the actual target values in the test set.
- First, we use the **value_counts()** method to count the number of occurrences of each predicted label and each true label in the test set.
- We then plot two sets of bars side-by-side using the **bar()** method from Matplotlib.
- We use annotations to display the height of each bar on top of the bar itself.
- We label the x-axis as "Range Label" and the y-axis as "Count", and add a title to the plot.
- Finally, we set the x-axis tick labels to be the range labels, and add a legend to the plot.



6.2 Evaluation of Random Forest Classifier Model

6.2.1 Evaluation Matrices

```
# Get the value counts for the predicted and actual labels
df_validate = pd.DataFrame({"prediction": y_pred}).value_counts()
df_validate2 = pd.DataFrame({"test_target": y_test}).value_counts()

# Get x and y values for the first set of bars
x = np.arange(len(df_validate.index))
y1 = df_validate.values

# Get x and y values for the second set of bars
y2 = df_validate2.values

# Set the width of each bar
width = 0.35

# Create the bar chart with two sets of bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, y1, width, label='Predictions')
rects2 = ax.bar(x + width/2, y2, width, label='Test Targets')

# Add value labels to the bars
for rect in rects1:
    height = rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')
```

```
# Add labels and a title to the plot
plt.xlabel('Range Label')
plt.ylabel('Count')
plt.title('Value Counts for Range Label')

# Set the x-axis tick labels to be the range labels
plt.xticks(x, df_validate.index)

# Add a legend to the plot
plt.legend()

# Display the plot
plt.show()
```

	Predicted Negatives	Predicted Positive
Actual Negative	39378	1233
Actual Positive	8173	1216

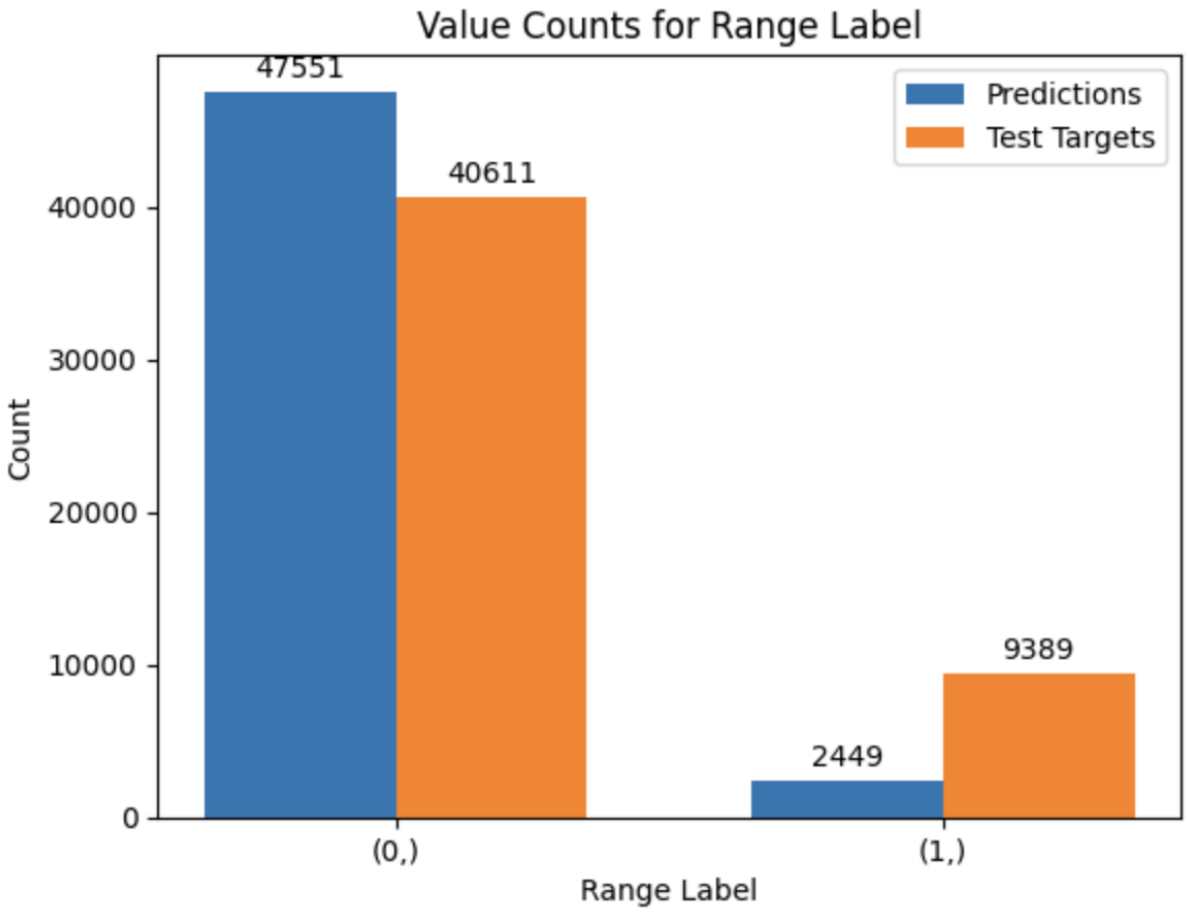
The accuracy of the model is 81.19%, which indicates that 81.19% of the predictions made by the model were correct. The precision score of 49.65% indicates that only about half of the positive predictions made by the model were actually true positives. The recall score of 12.95% indicates that the model correctly identified only a small proportion of the actual positive cases. The F1 score, which takes into account both precision and recall, is 20.54%. The ROC AUC score of 54.96% indicates that the model's performance is slightly better than random guessing. The confusion matrix shows that the model correctly classified 39378 negative cases and 1216 positive cases, but incorrectly classified 8173 positive cases as negative and 1233 negative cases as positive. Overall, the model's performance is better than the logistic regression model, but still has room for improvement to increase its accuracy and predictive power.

6.2.2 Logistic Regression prediction Bar Chart

```
# Get the value counts for the predicted and actual labels
df_validate = pd.DataFrame({"prediction": y_pred}).value_counts()
df_validate2 = pd.DataFrame({"test_target": y_test}).value_counts()

# Get x and y values for the first set of bars
x = np.arange(len(df_validate.index))
y1 = df_validate.values

# Get x and y values for the second set of bars
y2 = df_validate2.values

# Set the width of each bar
width = 0.35

# Create the bar chart with two sets of bars
fig, ax = plt.subplots()
rects1 = ax.bar(x - width/2, y1, width, label='Predictions')
rects2 = ax.bar(x + width/2, y2, width, label='Test Targets')

# Add value labels to the bars
for rect in rects1:
    height = rect.get_height()
    ax.annotate('{}'.format(height),
                xy=(rect.get_x() + rect.get_width() / 2, height),
                xytext=(0, 3),  # 3 points vertical offset
                textcoords="offset points",
                ha='center', va='bottom')
```

```
# Add labels and a title to the plot
plt.xlabel('Range Label')
plt.ylabel('Count')
plt.title('Value Counts for Range Label')

# Set the x-axis tick labels to be the range labels
plt.xticks(x, df_validate.index)

# Add a legend to the plot
plt.legend()

# Display the plot
plt.show()
```

- We create a bar chart to compare the predictions made by our model with the actual target values in the test set.
- First, we use the **value_counts()** method to count the number of occurrences of each predicted label and each true label in the test set.
- We then plot two sets of bars side-by-side using the **bar()** method from Matplotlib.
- We use annotations to display the height of each bar on top of the bar itself.
- We label the x-axis as "Range Label" and the y-axis as "Count", and add a title to the plot.
- Finally, we set the x-axis tick labels to be the range labels, and add a legend to the plot.



7. Future Enhancements

In summary, predicting flight delays is crucial in the airline industry, as it can help airlines optimize their schedules, reduce operational costs, and improve customer satisfaction. Machine learning techniques can be used to develop accurate flight delay prediction models, by analyzing various factors such as weather conditions, flight and airport data, and historical delays. In this context, the CRISP-DM process provides a structured approach to developing and implementing machine learning models for predicting flight delays.

We can further enhance our model with following suggestions.

- Increreasing dataset size: Collecting more data to train the model - Collecting more data related to flight delays such as weather data, flight schedules, and airport traffic can help the model to capture more patterns and provide better predictions.
- Feature engineering: Identifying new features - In addition to the existing features such as time of day, day of the week, and distance group, new features such as flight history, airport congestion, and airline operations can be engineered to provide better predictions.
- Incorporating external data sources: Utilizing external data sources - Incorporating external data sources such as flight traffic data, weather forecast data, and social media sentiment analysis can provide more insights into the possible causes of flight delays and help the model to make better predictions.
- Deploying the model: Implementing the model into a production environment - Once the model is developed, it needs to be integrated into a production environment where it can be used to make predictions in real-time. This requires a robust deployment pipeline that can handle different types of inputs, manage data flows, and scale to handle high volumes of requests.

Overall, these techniques can help to improve the accuracy and effectiveness of the flight delay prediction model, and make it more reliable and useful for the aviation industry.

8. References

- 2019 Airline Delays w/Weather and Airport Detail - https://www.kaggle.com/datasets/threnjen/2019-airline-delays-and-cancellations