FLIGHT DELAY PREDICTION

Group 5:

Ahmed

Jasleen

Simran

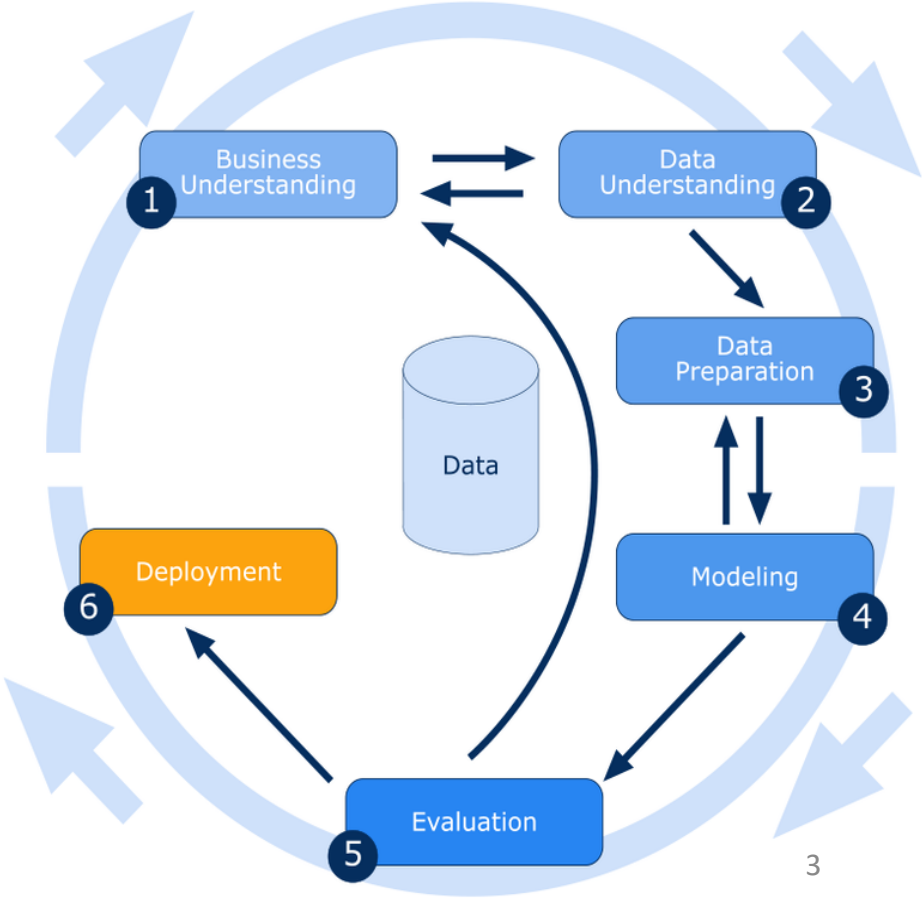
Bimsara

Content

- Methodology
- Business Understanding
- Data Understanding
- Data Preparation
- Modeling
- Evaluation
- Deployment

Methodology

- We are using the CRISP-DM (Cross-Industry Standard Process for Data Mining) framework to guide the data mining process.
- The framework consists of six phases:
    1. Business Understanding
    2. Data Understanding
    3. Data Preparation
    4. Modeling
    5. Evaluation
    6. Deployment



- **Business Understanding phase:** Involved identifying the problem and defining the project goal.
- **Data Understanding phase:** We gathered and explored historical flight data and weather information.
- **Data Preparation phase:** Involved transforming, and preparing the data for modeling.
- **Modeling phase:** Involved building and testing several machine learning models to predict flight delays.

- **Evaluation phase:** Involved comparing the performance of the different models and selecting the best one.

Business Understanding

- 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**.

- By identifying which flights are at high risk of being delayed, airlines can adjust their schedules, allocate resources more effectively, and inform passengers of potential delays in advance.
- Passengers can benefit from this prediction model by being able to plan their travel more efficiently and make alternate arrangements in advance to minimize the impact of delays on their schedule.
- The overall 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**.

Data Understanding

- The first step in the project is to collect a large dataset of flight information, including departure and arrival times, airline information, and weather data.
- The data can be obtained from various sources, including airline and weather data providers.

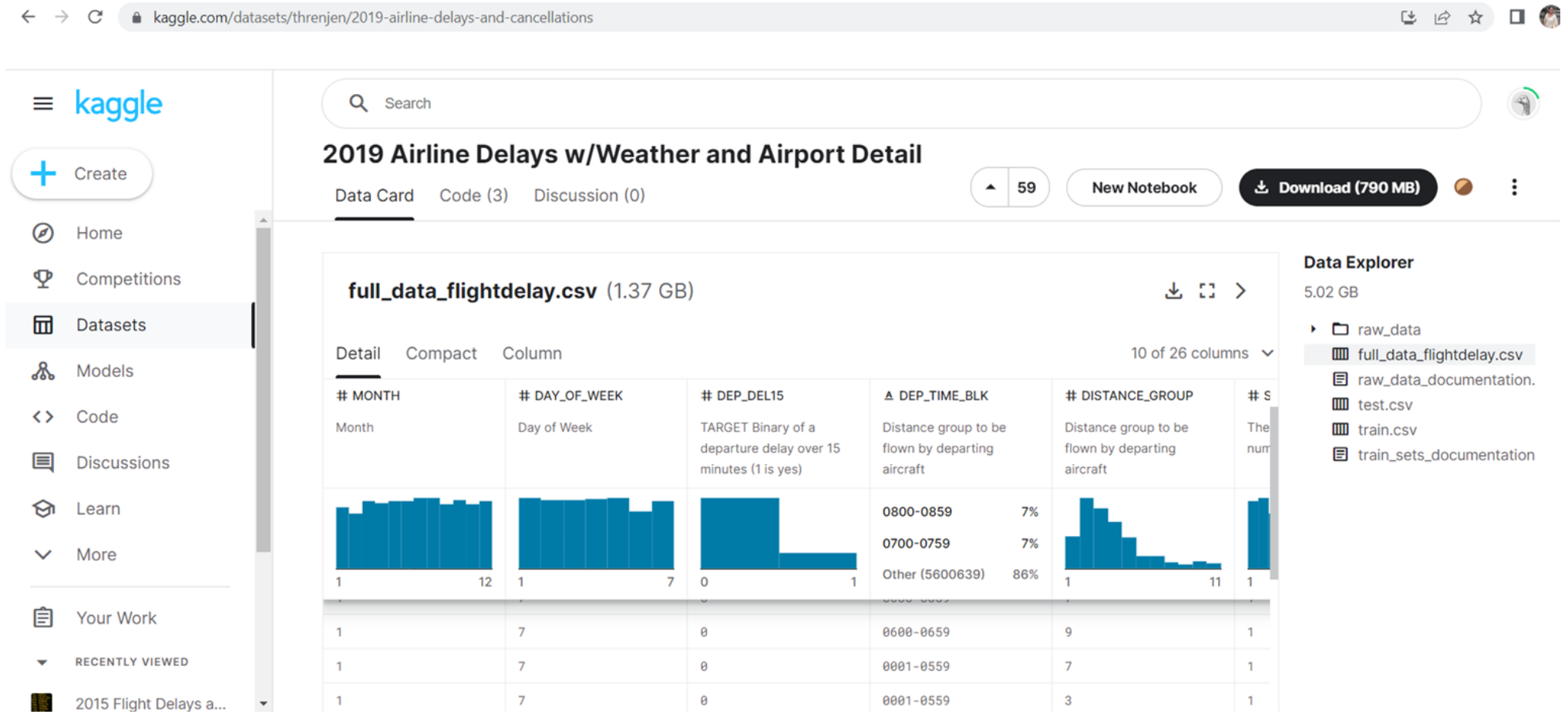
- This step is critical to the success of the project, as the accuracy and completeness of the data will directly impact the performance of the predictive model.

Some of the most important factors are:

- **Historical flight data**: The historical data on flight delays is essential for building a predictive model. This data should include information on the departure times of flights, as well as any delays that occurred.
- **Weather data**: Weather conditions can have a significant impact on flight delays. It is important to include weather data, such as temperature, precipitation, and wind speed, in your analysis.
- **Airport data**: Different airports may have different delay patterns, so it is important to include airport-specific data, such as the number of runways, the amount of air traffic, and the availability of ground staff.
- **Airline data**: Different airlines may have different policies and procedures that can affect flight delays. It is important to include airline-specific data, such as the number of flights operated.
- **Time-related factors**: The time of day, day of the week, and time of year can all impact the likelihood of a flight delay. For example, flights during peak travel times, such as holidays or weekends, may be more likely to experience delays.

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

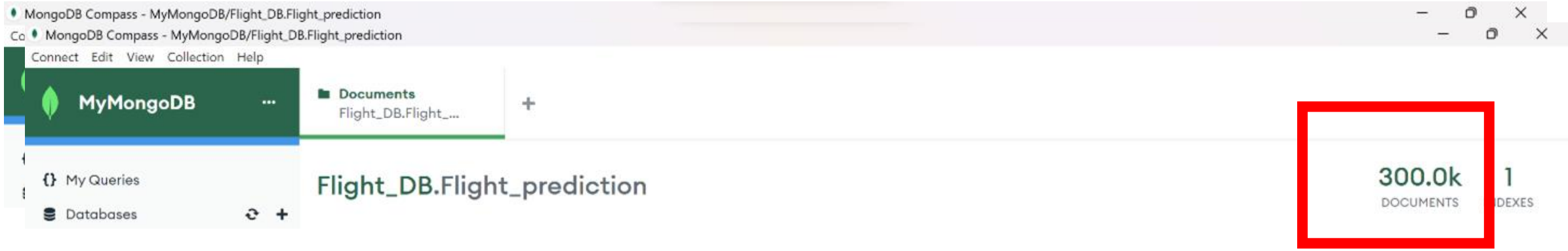
(Kaggle is an online platform for data scientists and machine learning enthusiasts that offers access to public datasets, tools for data analysis and visualization, and a community of over 4 million experts)



- The dataset content is as follows and it consists of 300.0k documents

Content

This is a classification dataset with detailed airline, weather, airport and employment information. If using the included train/test precombined data, the problem is a binary classification evaluating a delayed departure. All raw data files are also included for customization of the dataset, including adding cancellation, specific delay reasons, and/or arrival delays in order to create a multiclass problem. Note: Raw files for weather include only the top 90% of airports for passenger traffic, as all weather data was downloaded manually.



Data Preparation

- The data collected needs to be **transformed and processed** using Python and related tools to ensure that it is in a format suitable for analysis and storage in MongoDB.

- This process involves **identifying missing or erroneous data, removing duplicates if found, and transforming data types and values** as necessary.

- Once the data is cleaned and the schema is designed, the data will be imported into MongoDB using tools such as PyMongo or MongoDB Compass.

- Finally, the data can be further processed and analyzed using MongoDB's aggregation framework, which allows for powerful data transformations and querying capabilities.

- These are the steps we followed to prepare the data

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")
```

2. Process data- Convert Object ID to String

3. 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
0	Southwest Airlines Co.	...	13
1	American Eagle Airlines Inc.	...	14
2	Delta Air Lines Inc.	...	2
3	United Air Lines Inc.	...	21
4	United Air Lines Inc.	...	29
...	...	...	...
299995	NaN	...	NaN
299996	NaN	...	NaN
299997	NaN	...	NaN
299998	NaN	...	NaN
299999	NaN	...	NaN

4. Create a dictionary with old and new column names

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

6. Fill all missing values with Zero

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

7. 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
```

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

9. 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
```

```
# 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])
```

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

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

	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

11. 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

12. 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

...ing_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 ...ternational	0	...	13.0	1.45	Stapleton International	5	0.0	0.0	60.0	11.41	10	15
Memphis ...ternational	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 ...ternational	1	...	21.0	0.81	Los Angeles International	2	0.0	0.0	62.0	9.17	5	17
...n Francisco ...ternational	0	...	29.0	0.00	Logan International	2	0.0	0.0	73.0	13.65	6	17

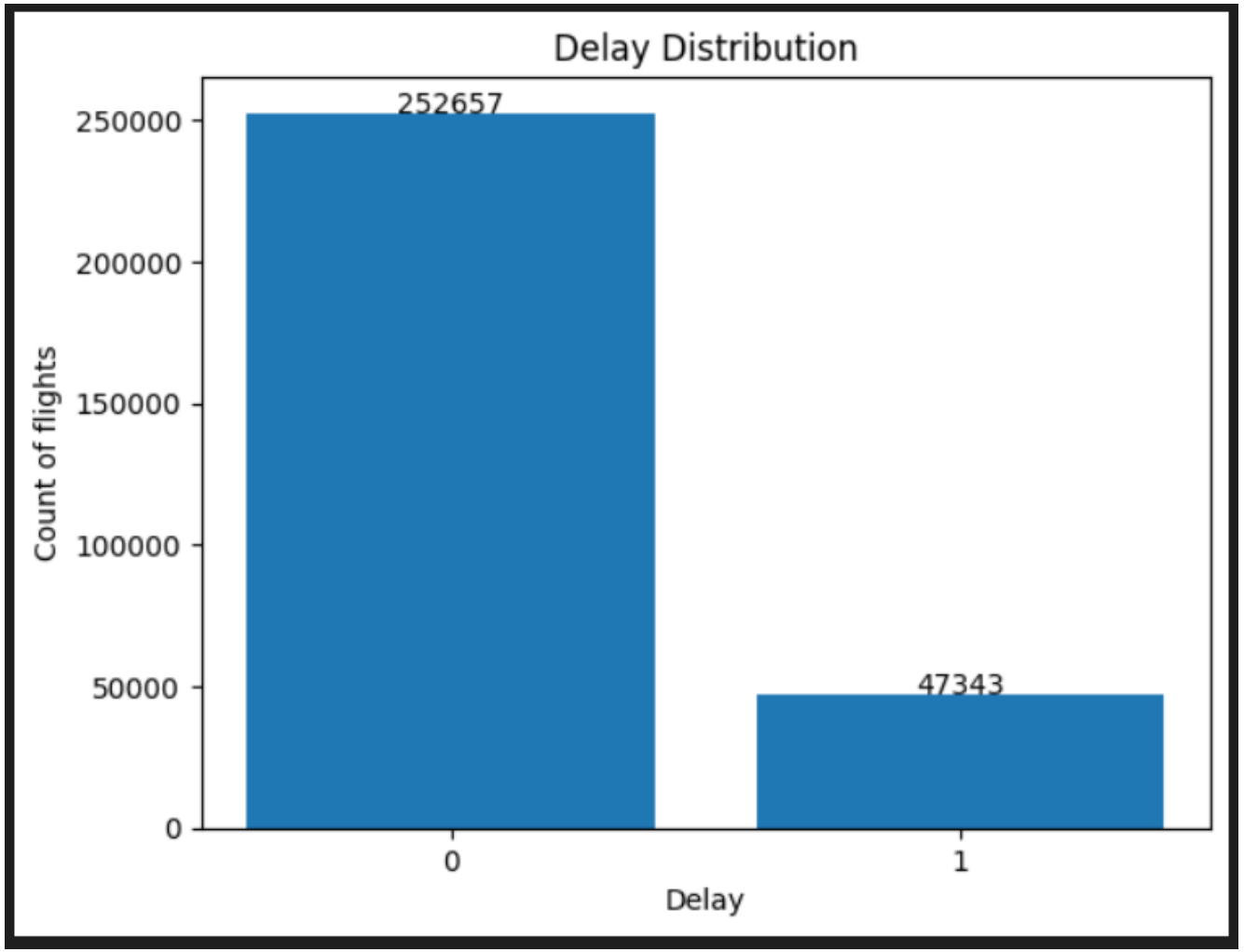
12. Create a new column called "departing_airport_int_label", which maps the categorical variable "departing_airport" to unique numerical values.

```
# 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

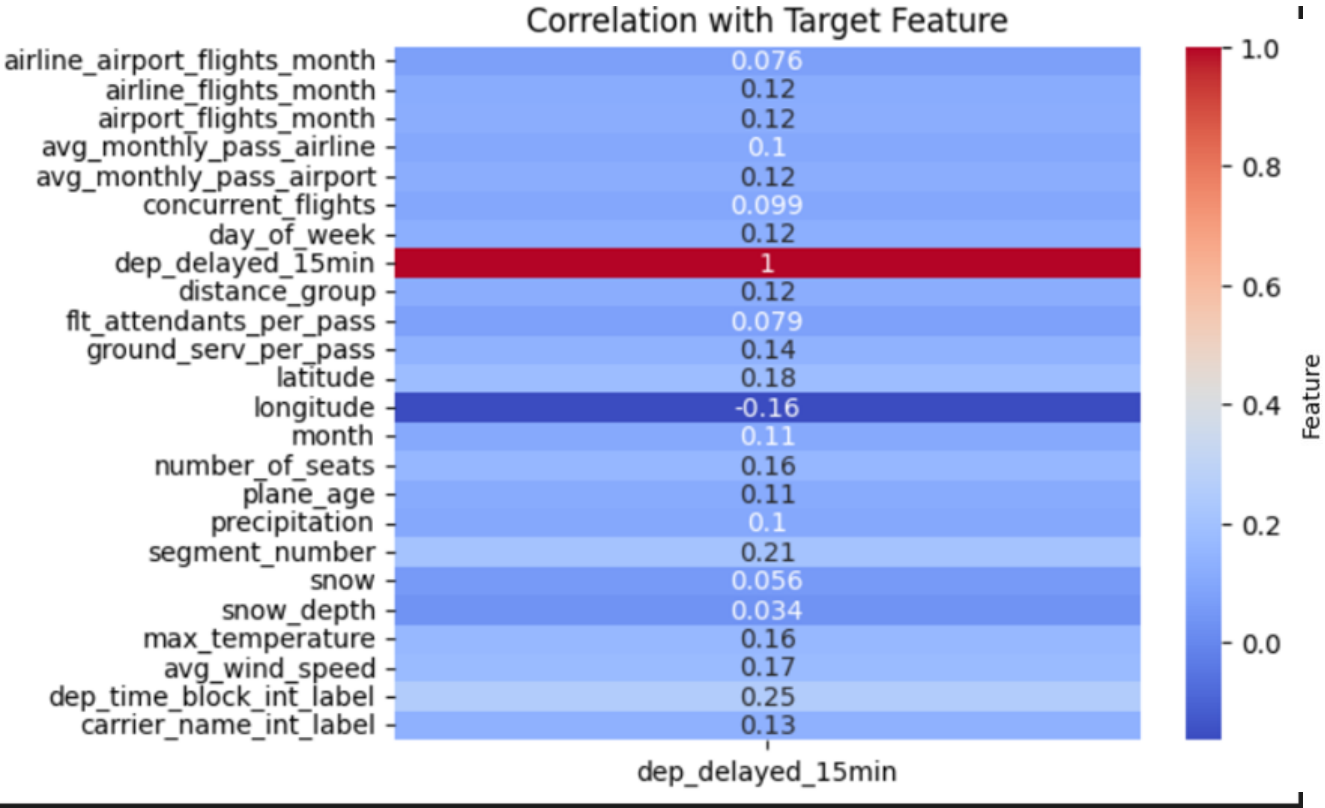
13. Create a plot shows the count of flights for each category ("delayed" or "not delayed") on the x-axis, and the number of flights on the y-axis.

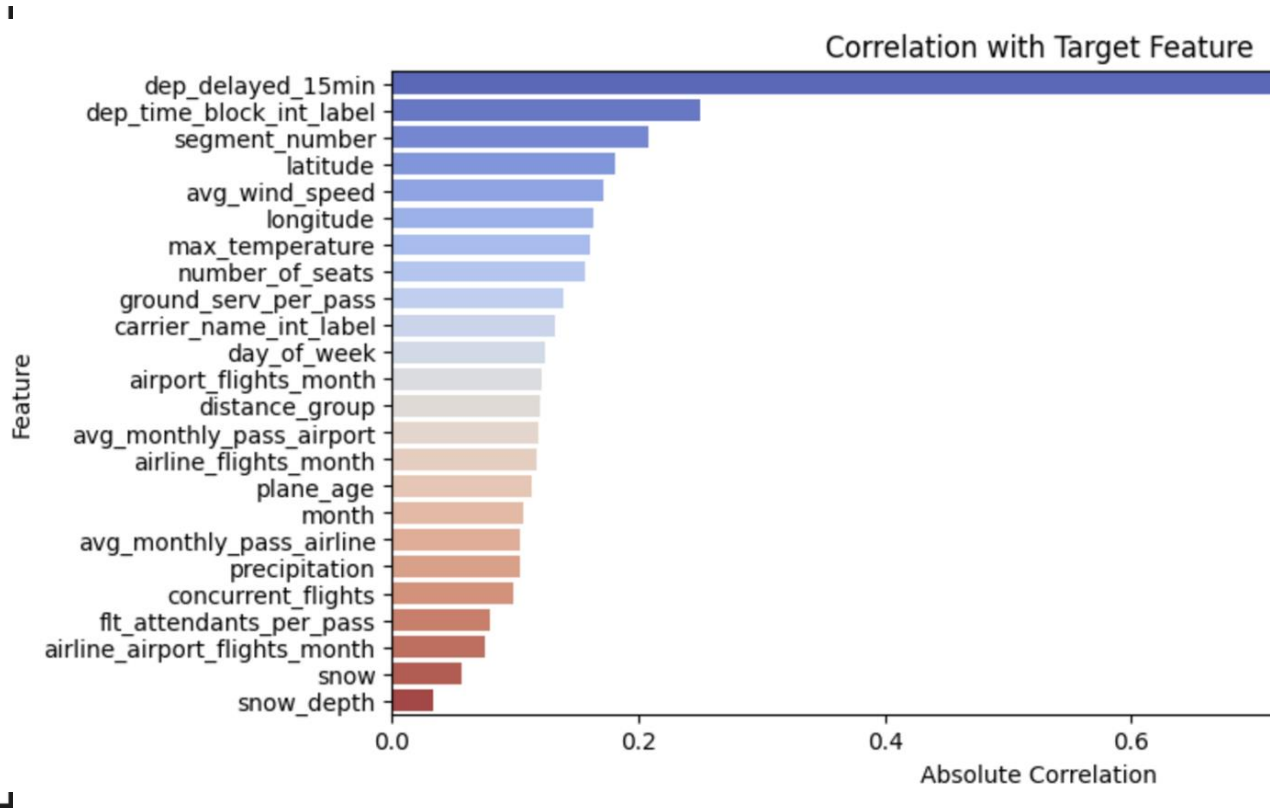
```python
import matplotlib.pyplot as plt

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()
```



16. 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()
```





Modeling

- Analyze data stored in MongoDB using aggregation pipeline.
- Identify patterns and trends in flight delays using grouping, filtering, and sorting. Build a predictive model using Python and related tools.
- Model uses historical flight data and weather information.
- Predicts which flights are likely to be delayed.
- We used two models here.

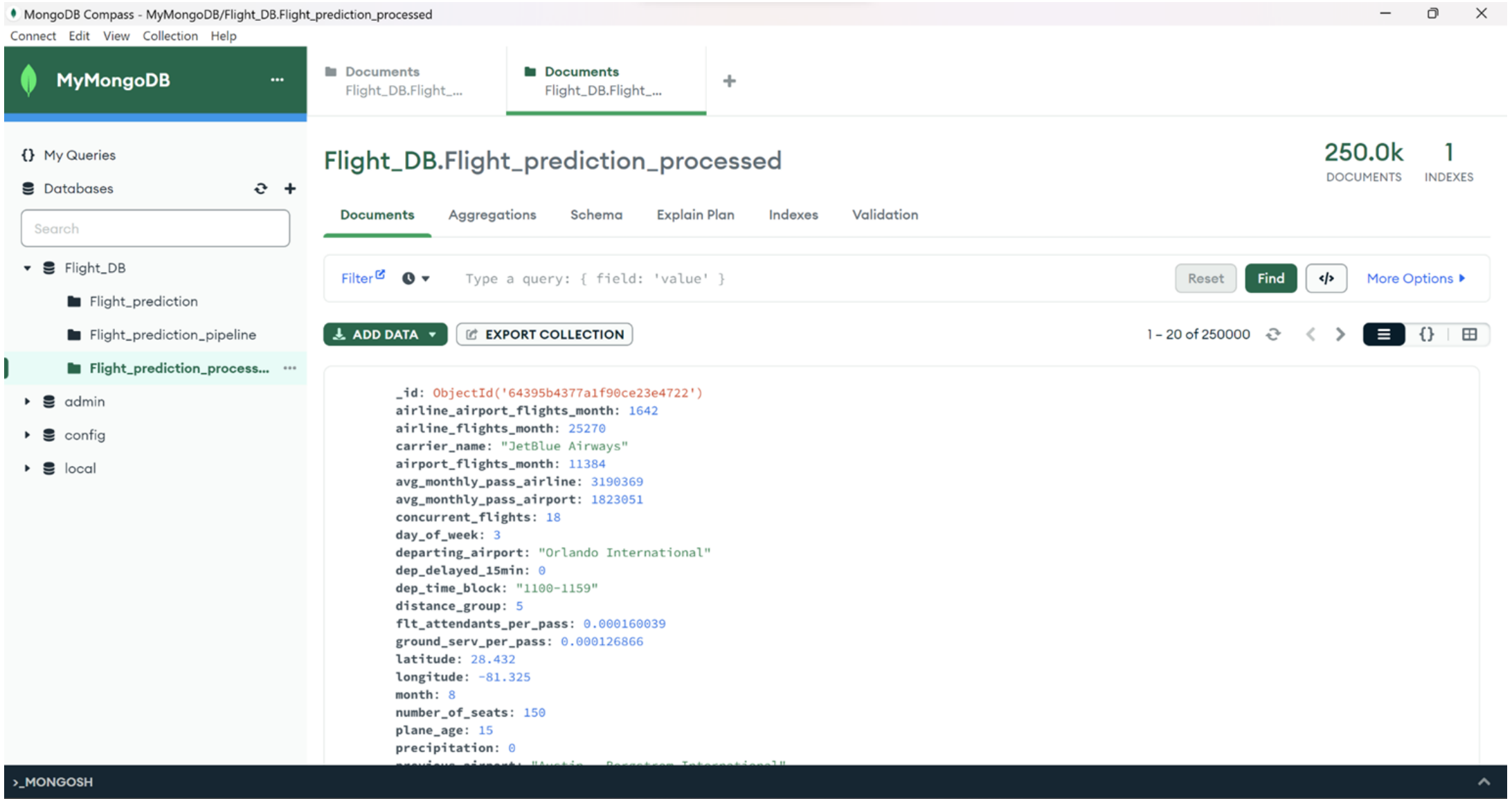
1. **Logistic Regression**
2. **Random forest classifier**

Import processed data to MongoDB

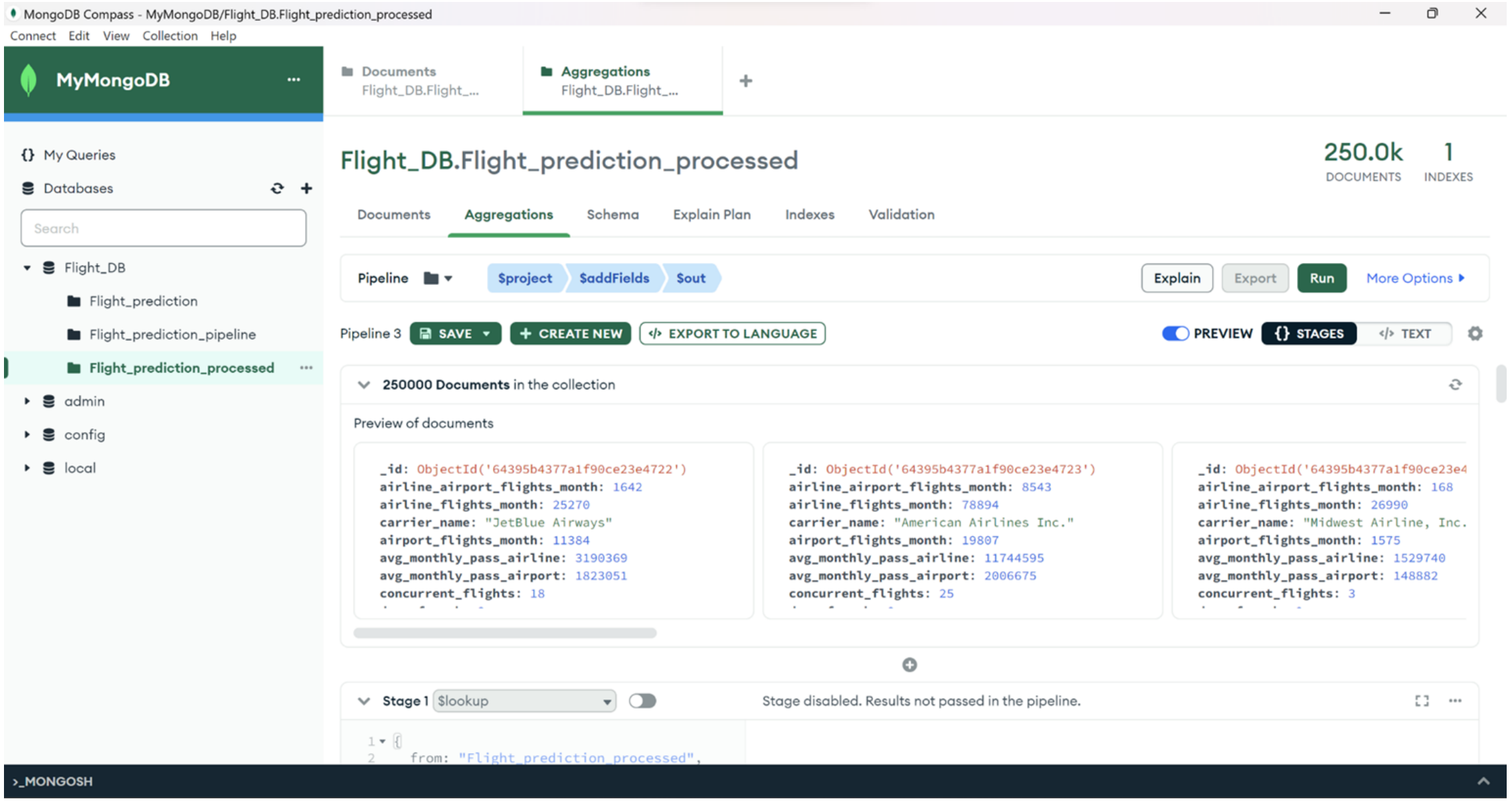
```
# Set up the MongoDB client
client = MongoClient("mongodb+srv://enoch:admin@mycluster.ixjzbgw.mongodb.net/test")

# 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()
```

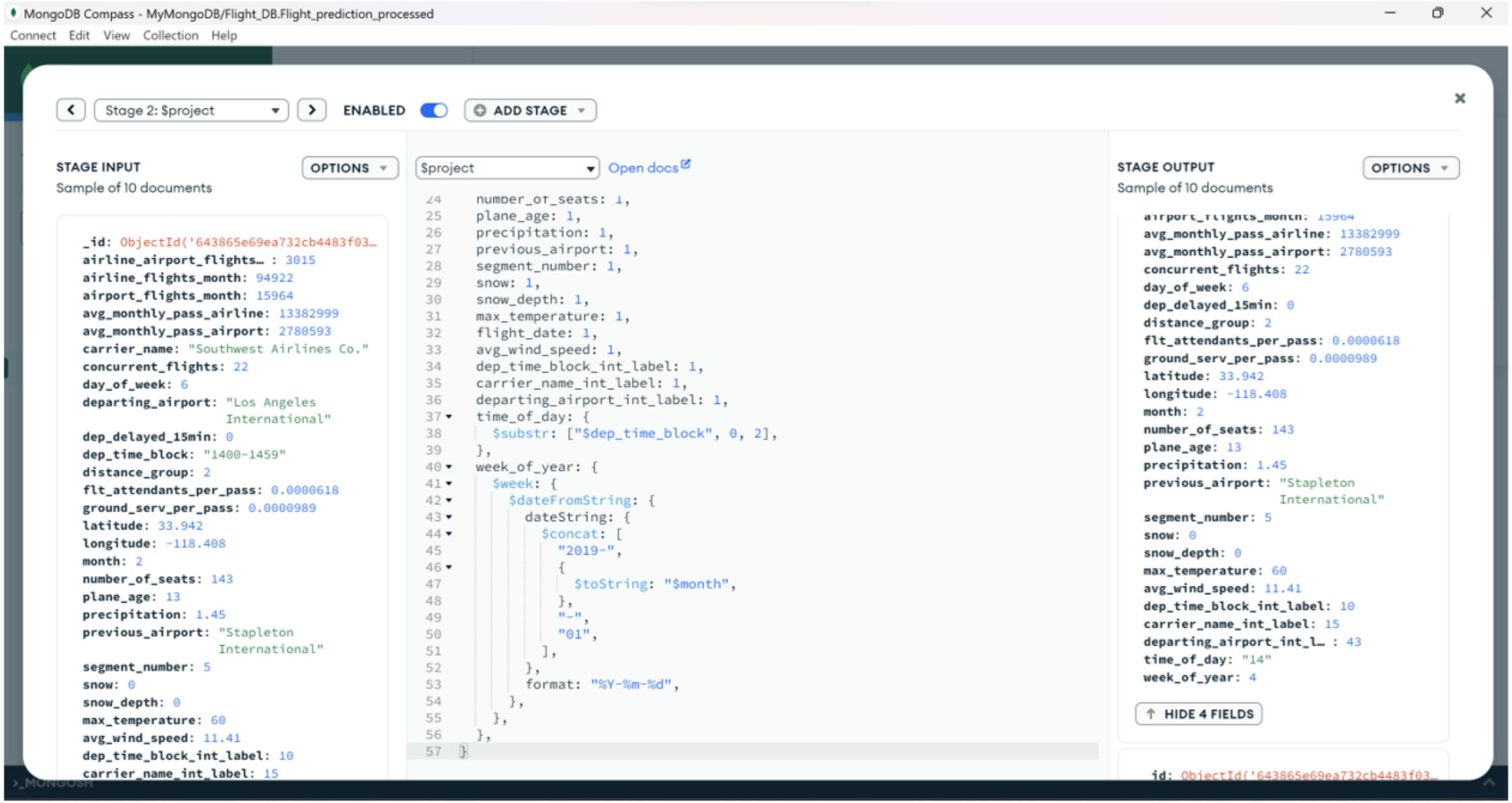
View of Flight_predcition_processed Collection, after Data has been processed



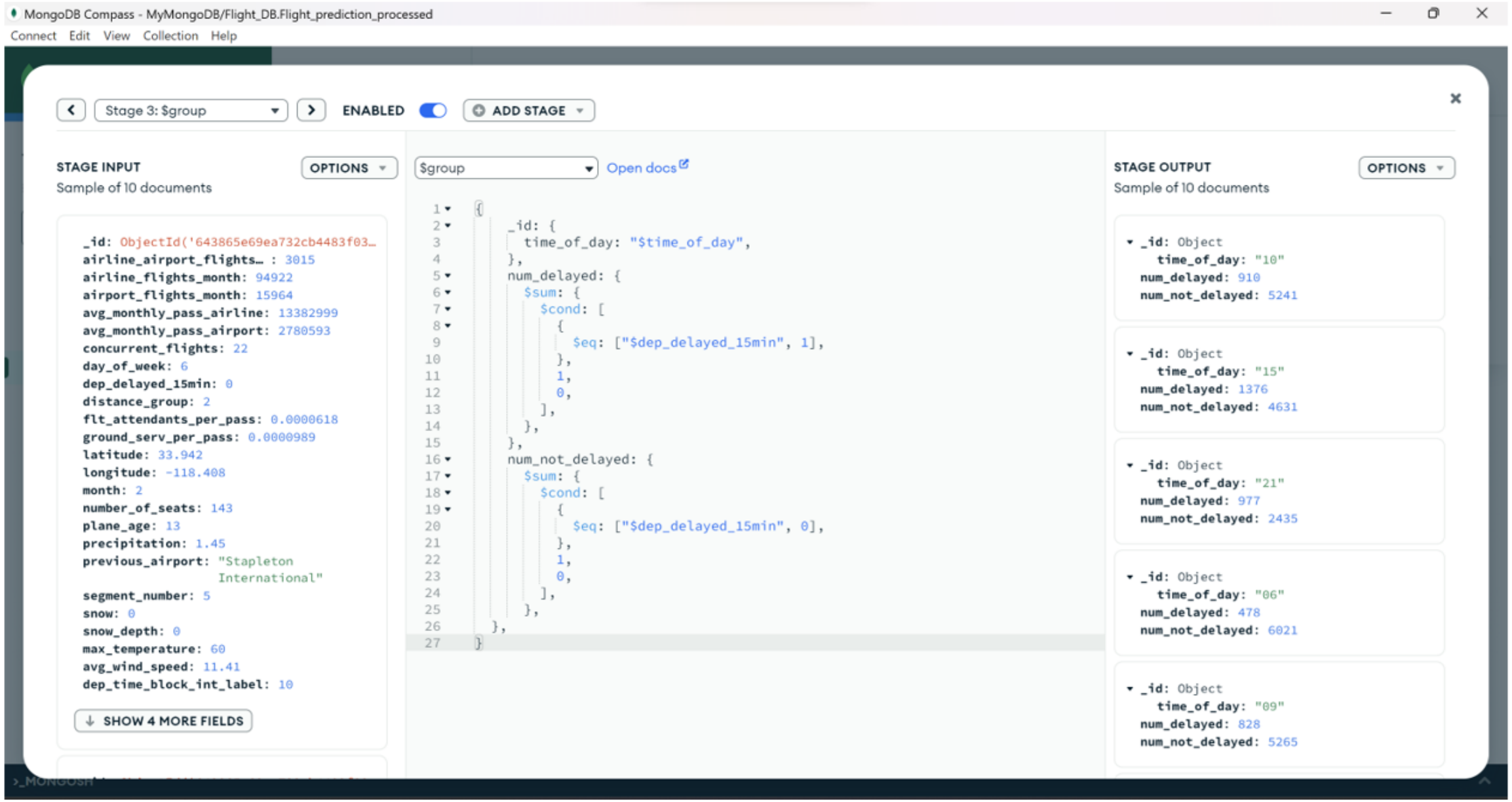
Using MongoDB aggregation pipeline



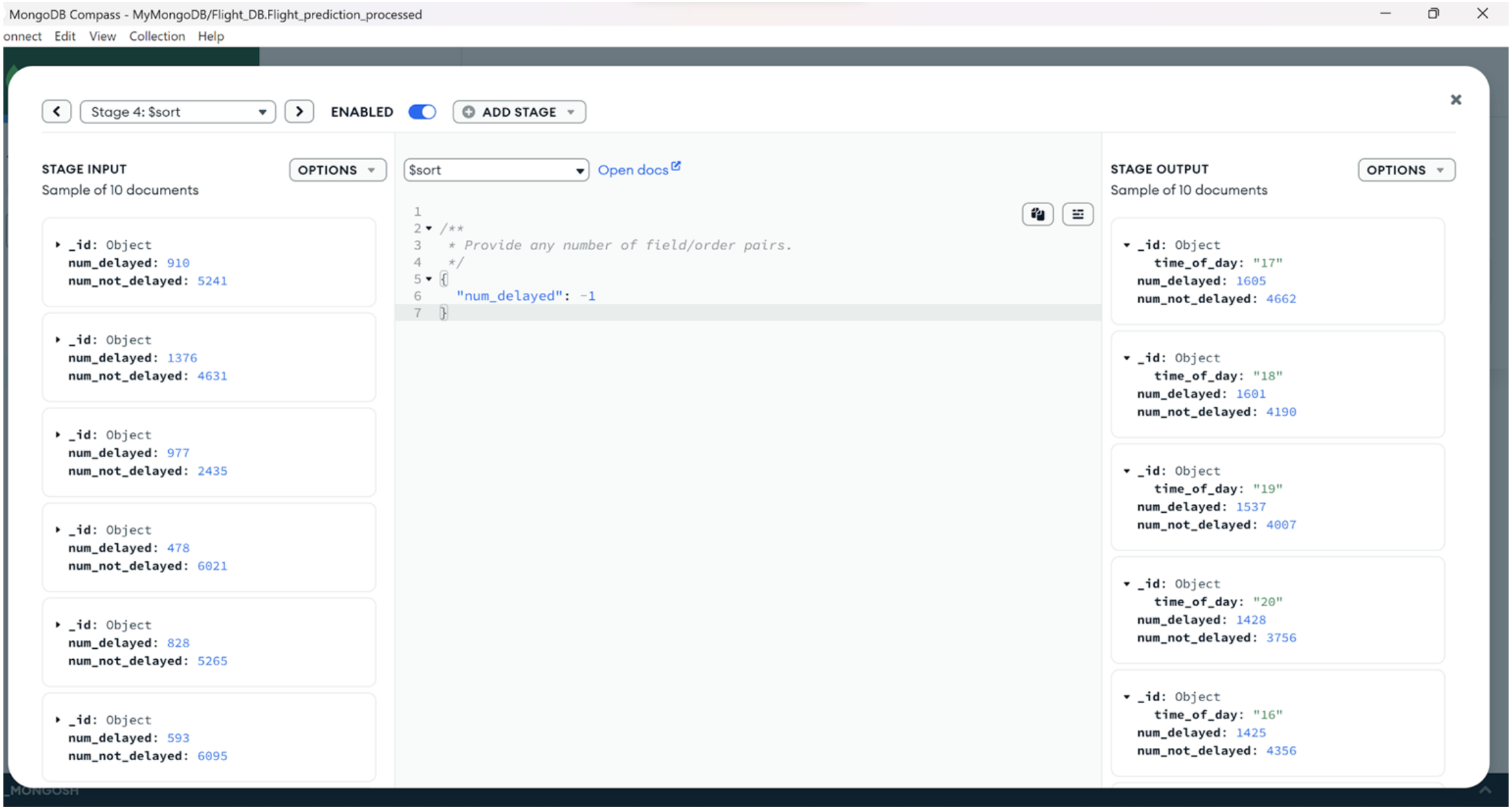
Stage 1 of the pipeline



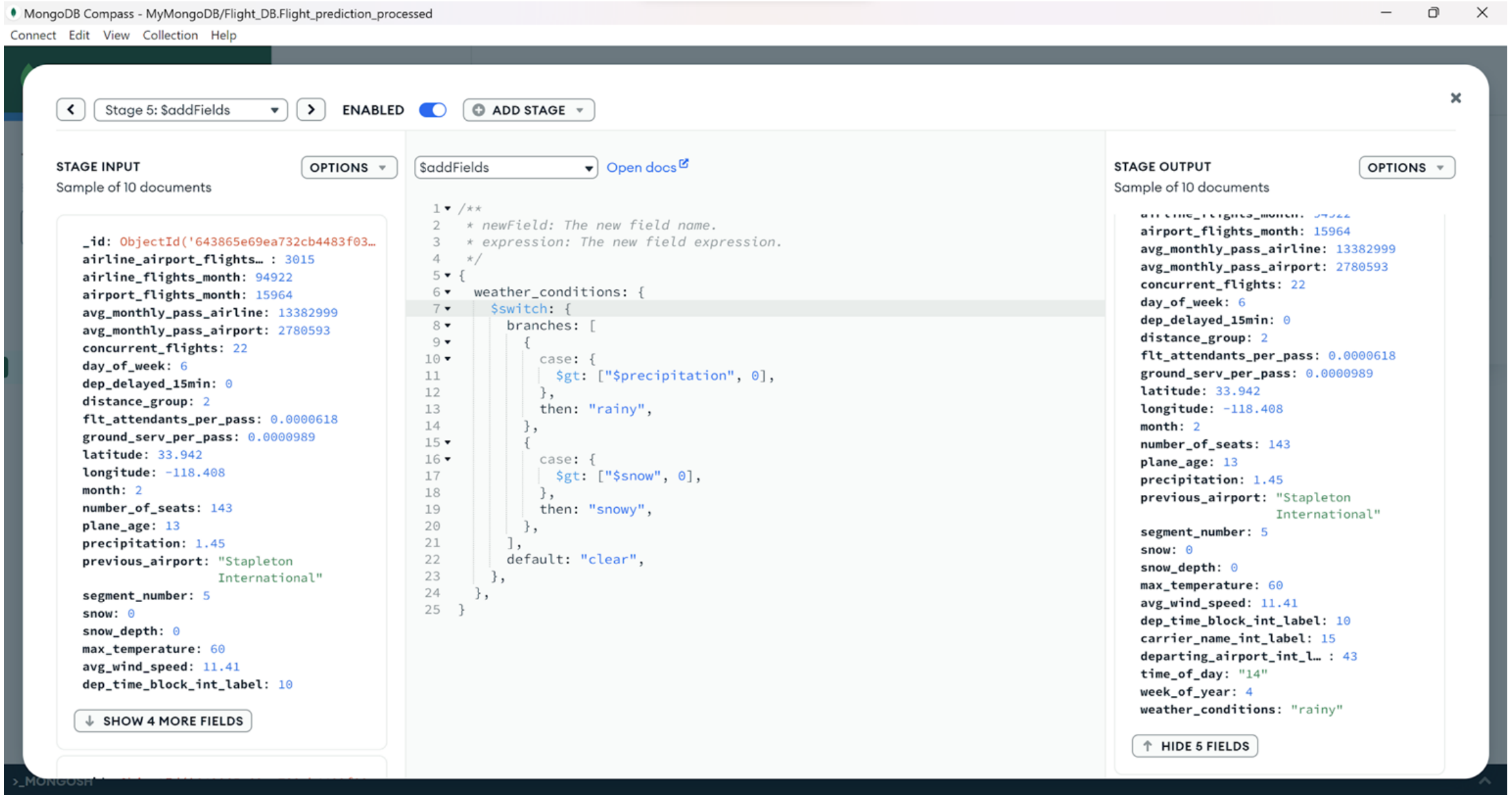
Stage 2 of the pipeline



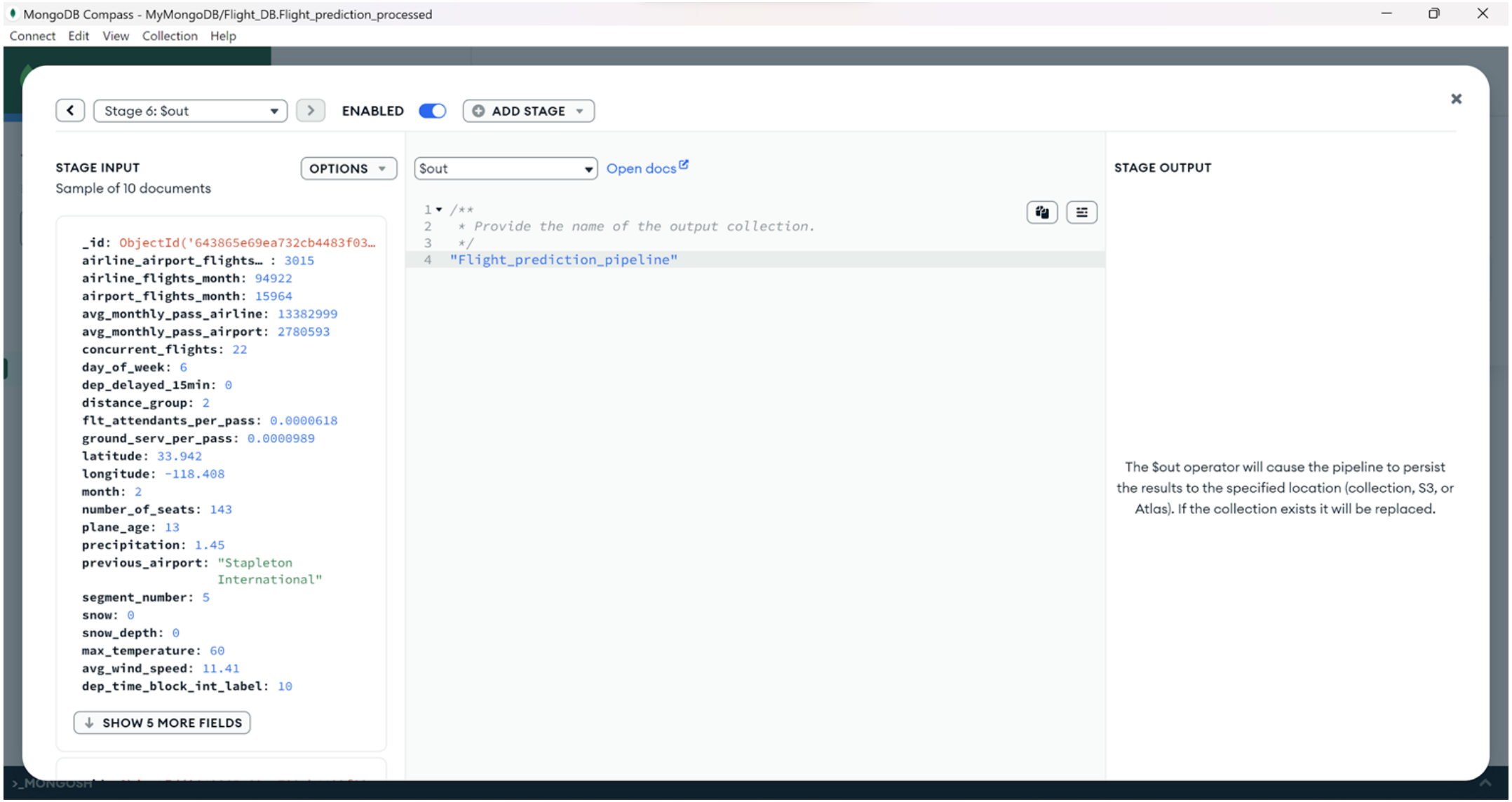
Stage 3 of the pipeline



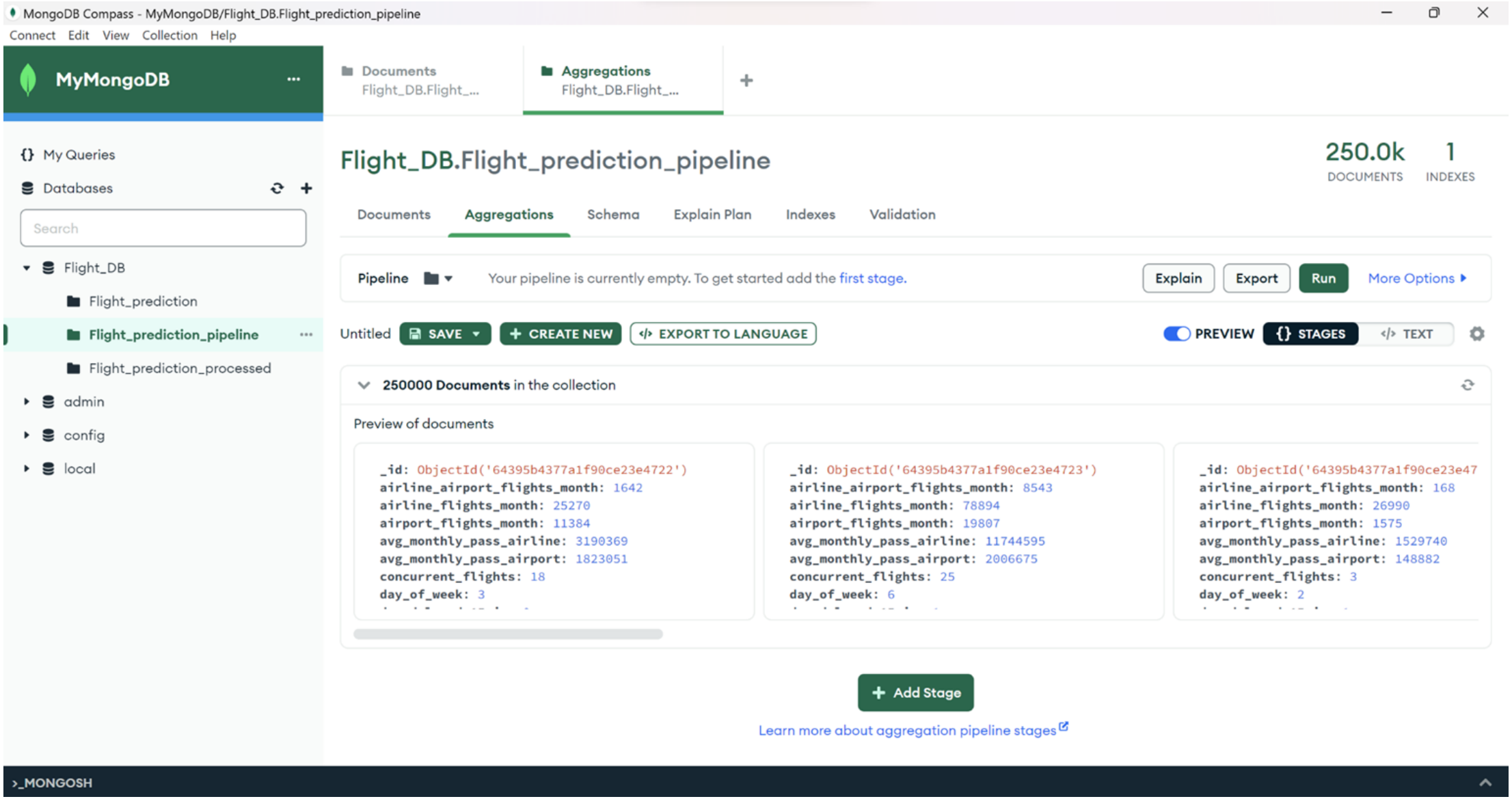
Stage 4 of the pipeline



Stage 5 of the pipeline



View of Flight_prediction_pipeline collection



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

```python
# 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	

1. Logistic Regression

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.

```python
# Adding the
#Assigning X and Y and reshaping to be bale to be used in the model.

x = df_pipeline[["concurrent_flights","day_of_week","distance_group","flt_attendants_per_pass","ground_serv_per_pass","month","number_of_seats","precipitation","snow","snow_depth","
y = df_pipeline["dep_delayed_15min"]
```

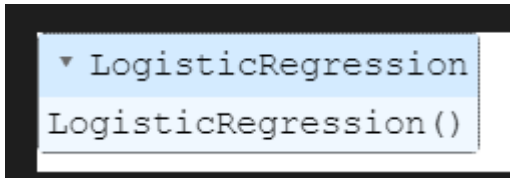
✓ 0.0s Python

```python
# .20 = 20%  80 Train 20% test
train_data, test_data, train_target, test_target = train_test_split(x, y, test_size=0.2,random_state=21)
```

✓ 0.0s Python

```python
model = LogisticRegression()
model.fit(train_data, train_target)
```

✓ 0.3s Python



- This code generates predictions for the target variable based on the test data using the logistic regression model trained in previous code.
- The predictions are stored in a pandas DataFrame along with the actual values of the target variable from the test data.
- 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

2. Random Forest Classifier

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)
```

Evaluation

- Evaluation is important in CRISP to determine accuracy and effectiveness of model
- Compares predicted outcomes with actual outcomes to see model's performance
- Techniques used include confusion matrices, ROC AUC, precision, recall and F1-score
- Evaluation ensures model meets required standards and can predict flight delays accurately
- Helps identify areas for improvement/refinement to achieve better results

Evaluation Metrics for Logistic Regression Model

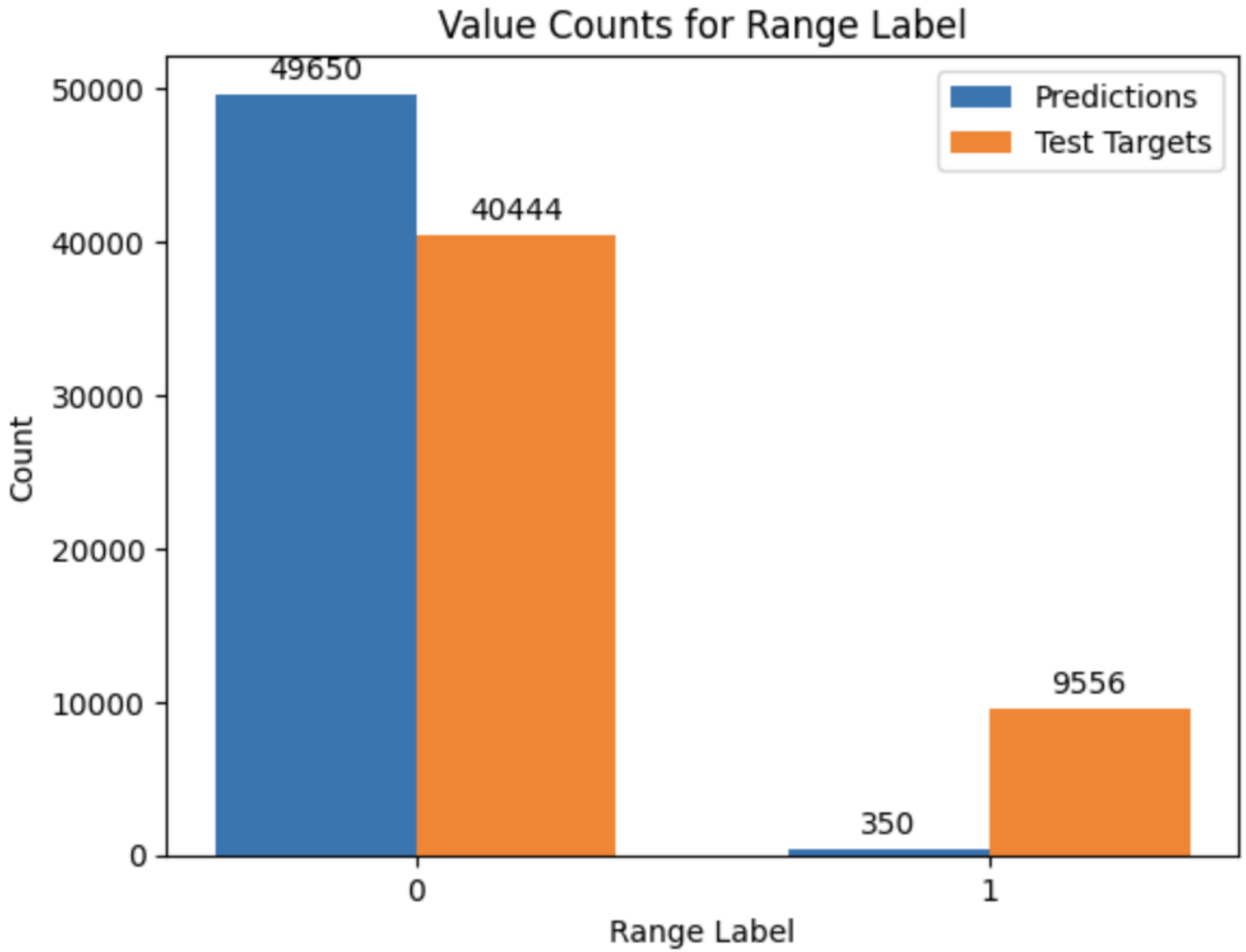
- Accuracy: **80.88%** - How accurate is the model?
- Precision: **49.43%** - How many did we predict true and were right ?
- Recall: **1.81%** - How many did we guess out of the actual positive?
- F1 Score: **3.49%**
- ROC AUC Score: **50.69%**
- Confusion Matrix:

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

```python
# -----> 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

Logistic Regression prediction Bar Chart



Evaluation Metrics for Random Forest Classifier

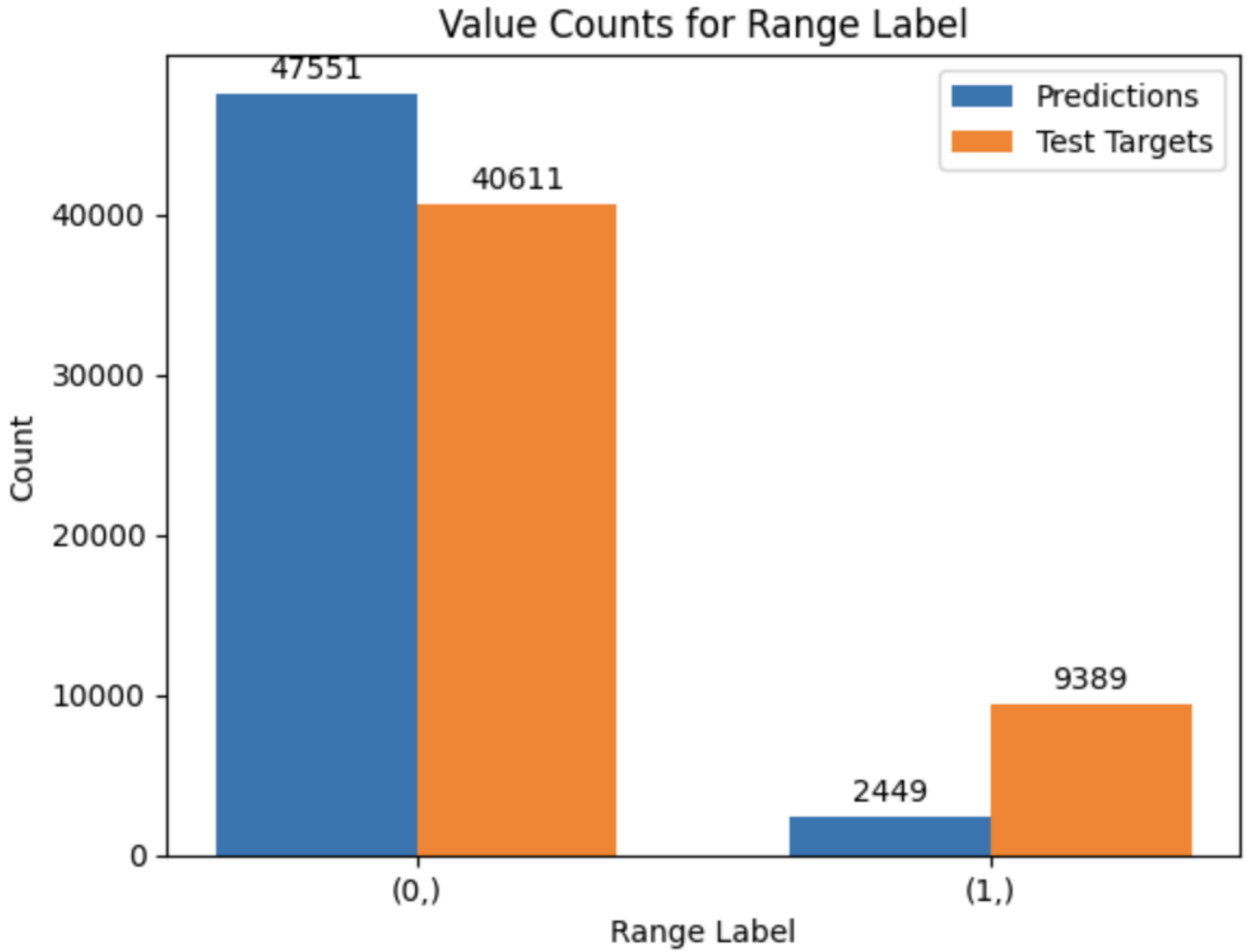
- Accuracy: **81.19%**
- Precision: **49.65%**
- Recall: **12.95%**
- F1 Score: **20.54%**
- ROC AUC Score: **54.96%**
- Confusion Matrix:

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

```python
# 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')
```

```python
# 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()
```

Random Forest Classifier Bar Chart



- **Increasing dataset size**: Collecting more data to train the model
- **Feature engineering**: Identifying new features
- **Incorporating external data sources**: Utilizing external data sources.
- **Deploying the model**: Implementing the model into a production environment

Future enhancements

References

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

THANK YOU