VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

DATA MINING

FINAL PROJECT REPORT

TOPIC:

PREDICTING FLIGHT DELAY

Lecturer: Mrs Cao Thi Nhan

Mr Vu Minh Sang

Class: IS252.O22.HTCL

Group: Team BLOCKS MAGIC

Student performance:

Le Nguyen Nhat Minh	21522338
Hoang Nhat Minh	21522336
Tran Thi Luyen	21521107
Nguyen Thi Thuy Hang	21520822

Ho Chi Minh City, 2024

VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY

UNIVERSITY OF INFORMATION TECHNOLOGY

INFORMATION SYSTEM FACULTY

———□🕮□———

DATA MINING

FINAL PROJECT REPORT

TOPIC:

PREDICTING FLIGHT DELAY

Lecturer: Mrs Cao Thi Nhan

Mr Vu Minh Sang

Class: IS252.O22.HTCL

Group: Team BLOCKS MAGIC

Student performance:

Le Nguyen Nhat Minh	21522338
Hoang Nhat Minh	21522336
Tran Thi Luyen	21521107
Nguyen Thi Thuy Hang	21520822

Ho Chi Minh City, 2024

TEACHER'S COMMENTS

ACKNOWLEDGEMENT

TABLE OF CONTENTS

Chapter 1 INTRODUCTION

1.1: Problems

Air travel has become an essential part of modern life, enabling millions of people to travel around the world every day. However, one of the major challenges faced by passengers, airlines, and airports is flight delays. Delays are an aviation concept that no customer wants to encounter.

Flight delays can cause significant inconvenience, economic losses, and logistical issues. For passengers, delays mean spending additional time at the airport or waiting at the point of departure, which can be uncomfortable and time-consuming. If there are connecting flights, one may have to change their schedule and possibly incur additional costs. Delays can alter planned travel itineraries and cause frustration, disrupting the execution of tourists' activities. For airlines and airports, delays can lead to increased operational costs, decreased customer satisfaction, and challenges in managing complex schedules.

Since flight delays are inevitable, our team aims to examine historical flight data to understand how weather affects the likelihood of flight delays (with a flight delay defined as a departure delay of more than 15 minutes from the scheduled time).

We aim to predict when delays might occur using trained machine learning models, focusing solely on delays without considering flight cancellations.

1.2: Project goal

- Building a natural language data system, utilizing machine learning to train machines to provide highly reliable predictions and information for humans.
- Predicting flight delay probabilities enables airlines to schedule their flights or implement preventive measures beforehand.

1.3: Developer tools & Technology

- Data sources: 2019 Airline Delays w/Weather and Airport Detail | kanggle

In the process of implementation, the group used some software for researching and developing the topic:

- ❖ Information collection and analysis using the Python library and programming language.

Chapter 2 DATA PREPROCESSING

2.1: Description of original data

2.1.1: Data Sources

Author: JEN WADKINS

2.1.2: Data file

Total data rows: 6.489.062

2.1.3: Attribute number and value

Total columns: 26

Dataset characteristics: Multivariable

Attribute number characteristics:

Lost value: None

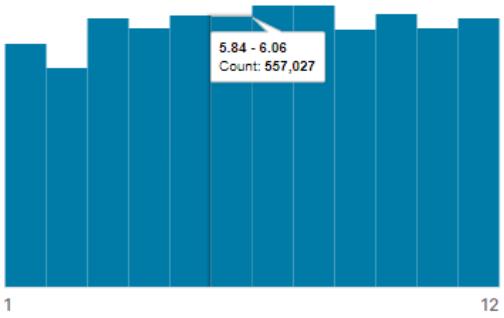
2.1.4: Statistics of attribute values

Symbol: # -number, A -character

Sources: 2019 Airline Delays w/Weather and Airport Detail | kanggle

MONTH

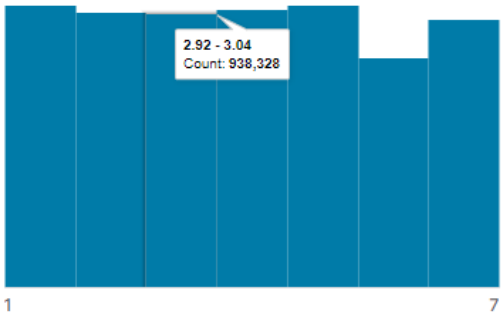
Month



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	6.61	
Std. Deviation	3.4	
Quantiles	1	Min
	12	Max

DAY_OF_WEEK

Day of Week



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	3.94	
Std. Deviation	2	
Quantiles	1	Min
	7	Max

DEP_DEL15

TARGET Binary of a departure delay over 15 minutes (1 is yes)

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.19	
Std. Deviation	0.39	
Quantiles	0	Min
	1	Max

A DEP_TIME_BLK

Distance group to be flown by departing aircraft

0800-0859	7%
0700-0759	7%
Other (5600639)	86%

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Unique	19	
Most Common	0800-0859	7%

A DEP_TIME_BLK

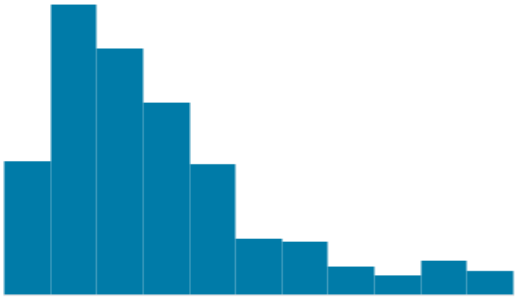
Distance group to be flown by departing aircraft

0800-0859	7%
0700-0759	7%
Other (5600639)	86%

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Unique	19	
Most Common	0800-0859	7%

DISTANCE_GROUP

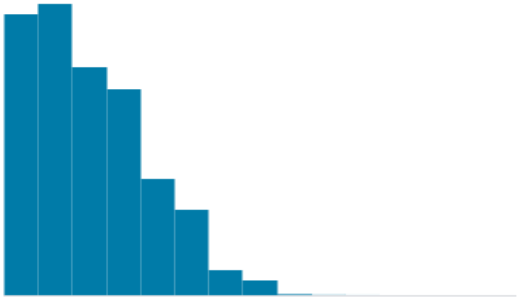
Distance group to be flown by departing aircraft



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	3.82	
Std. Deviation	2.38	
Quantiles	1	Min
	11	Max

SEGMENT_NUMBER

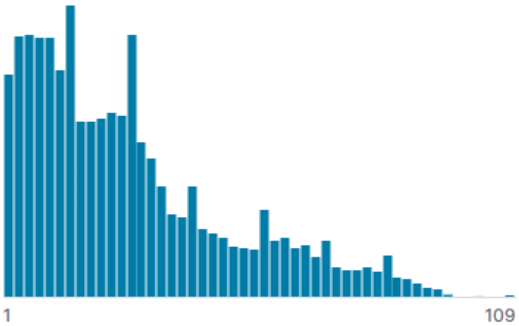
The segment that this tail number is on for the day



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	3.05	
Std. Deviation	1.76	
Quantiles	1	Min
	15	Max

CONCURRENT_FLIGHTS

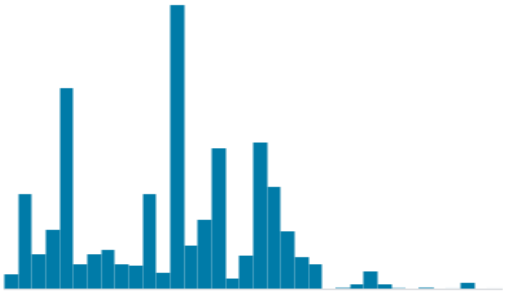
Concurrent flights leaving from the airport in the same departure block



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	27.8	
Std. Deviation	21.5	
Quantiles	1	Min
	109	Max

NUMBER_OF_SEATS

Number of seats on the aircraft



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	134	
Std. Deviation	46.5	
Quantiles	44	Min
	337	Max

A CARRIER_NAME

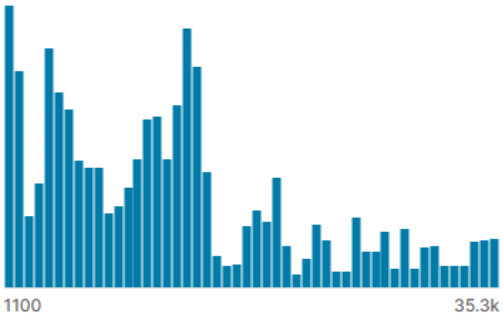
Carrier

Southwest Airlines Co.	20%
Delta Air Lines Inc.	14%
Other (4254387)	66%

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Unique	17	
Most Common	Southwest ...	20%

AIRPORT_FLIGHTS_MONTH

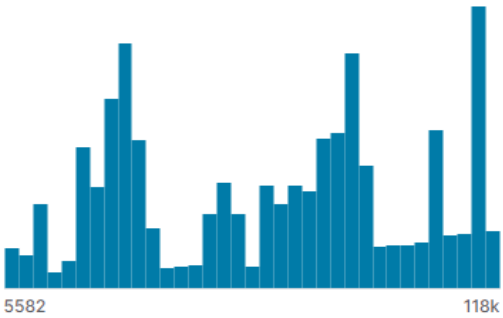
Avg Airport Flights per Month



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	12.7k	
Std. Deviation	8.84k	
Quantiles	1100	Min
	35.3k	Max

AIRLINE_FLIGHTS_MONTH

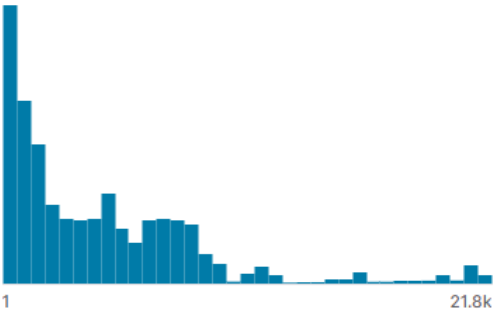
Avg Airline Flights per Month



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	63k	
Std. Deviation	34.4k	
Quantiles	5582	Min
	118k	Max

AIRLINE_AIRPORT_FLIGHTS_MONTH

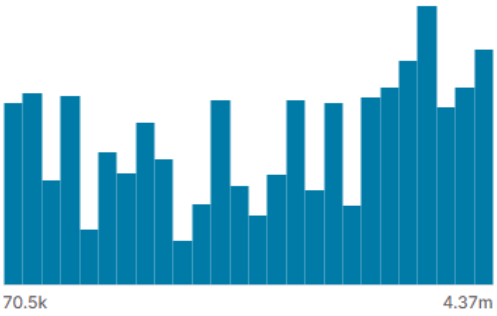
Avg Flights per month for Airline AND Airport



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	3.46k	
Std. Deviation	4.25k	
Quantiles	1	Min
	21.8k	Max

AVG_MONTHLY_PASS_AIRPORT

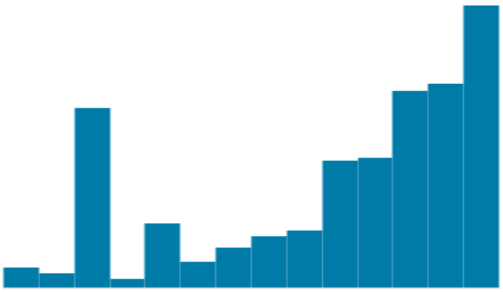
Avg Passengers for the departing airport for the month



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	1.59m	
Std. Deviation	1.12m	
Quantiles	70.5k	Min
	4.37m	Max

AVG_MONTHLY_PASS_AIRLINE

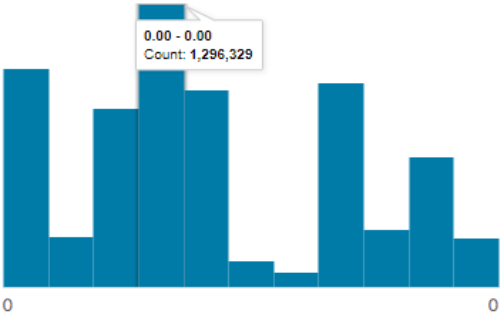
Avg Passengers for airline for month



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	7.81m	
Std. Deviation	5.05m	
Quantiles	474k	Min
	13.4m	Max

FLT_ATTENDANTS_PER_PASS

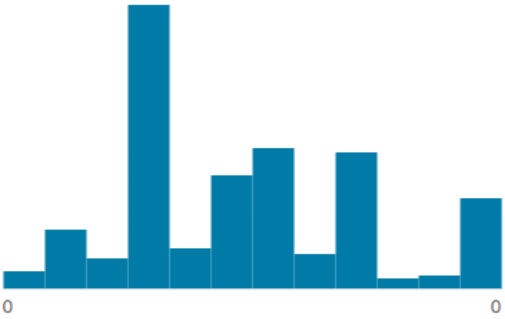
Flight attendants per passenger for airline



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0	
Std. Deviation	0	
Quantiles	0	Min
	0	Max

GROUND_SERV_PER_PASS

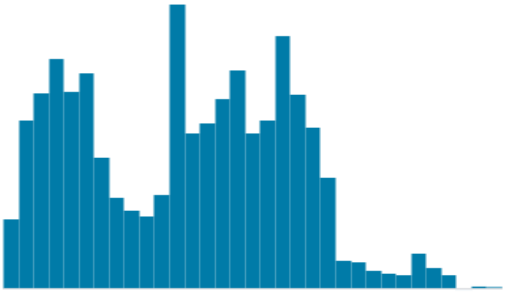
Ground service employees (service desk) per passenger for airline



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0	
Std. Deviation	0	
Quantiles	0	Min
	0	Max

PLANE_AGE

Age of departing aircraft



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	11.5	
Std. Deviation	6.94	
Quantiles	0	Min
	32	Max

A DEPARTING_AIRPORT

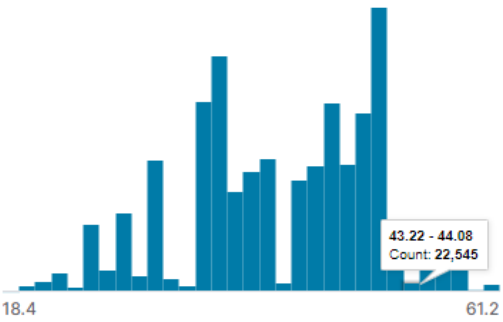
Departing Airport

Atlanta Municipal	6%
Chicago O'Hare International	5%
Other (5767414)	89%

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Unique	96	
Most Common	Atlanta Mun...	6%

A LATITUDE

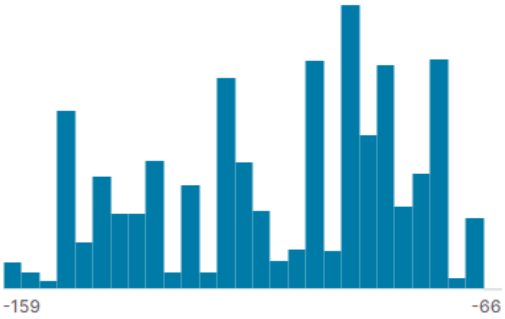
Latitude of departing airport



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	36.7	
Std. Deviation	5.5	
Quantiles	18.4	Min
	61.2	Max

LONGITUDE

Longitude of departing airport



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	-94.3	
Std. Deviation	17.9	
Quantiles	-159	Min
	-66	Max

PREVIOUS_AIRPORT

Previous airport that aircraft departed from

NONE	22%
Atlanta Municipal	3%
Other (4832713)	74%

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Unique	356	
Most Common	NONE	22%

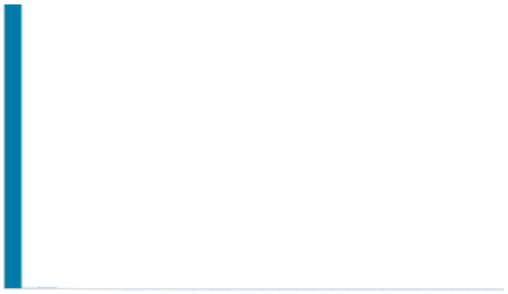
PRCP

Inches of precipitation for day

Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.1	
Std. Deviation	0.34	
Quantiles	0	Min
	11.6	Max

SNOW

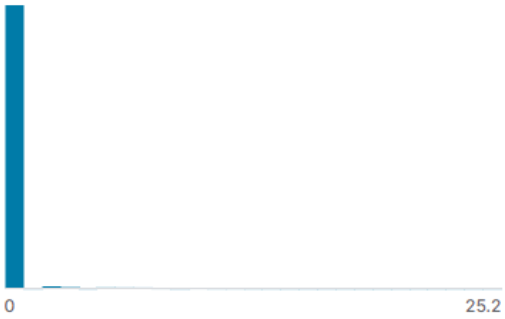
Inches of snowfall for day



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.03	
Std. Deviation	0.32	
Quantiles	0	Min
	17.2	Max

SNWD

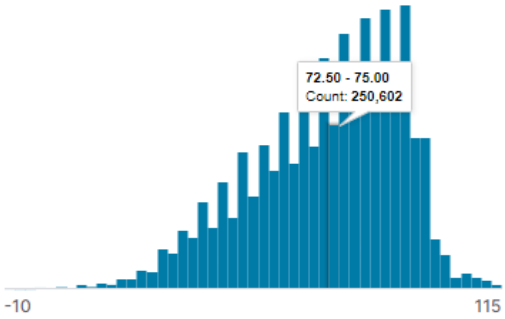
Inches of snow on ground for day



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	0.09	
Std. Deviation	0.73	
Quantiles	0	Min
	25.2	Max

TMAX

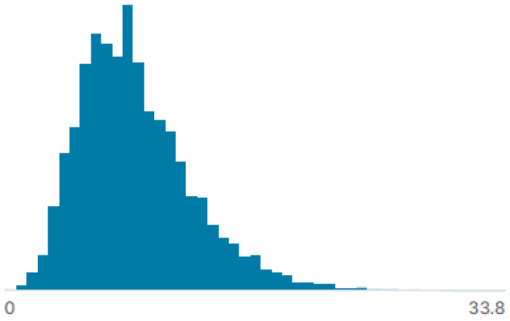
Max temperature for day



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	71.5	
Std. Deviation	18.4	
Quantiles	-10	Min
	115	Max

AWND

Max wind speed for day



Valid	6.49m	100%
Mismatched	0	0%
Missing	0	0%
Mean	8.34	
Std. Deviation	3.61	
Quantiles	0	Min
	33.8	Max

Attribute statistics table:

STT	Attribute	Attribute meaning	Attribute type	Value of property	Average value	Median value	Mode
1	**MONTH**	Month	Categorical	[0 -12]		6.61	8

2	**DAY_OF_WEEK**	Day of week	Categorical	[1 – 7]			5
3	**DEP_DEL15**	Departure delay over 15 minutes (1 is yes)	Categorical	[0-1]			0
4	**DEP_TIME_BLK**	Distance group to be flown by departing aircraft	Categorical	19 unique values			0800-0859
5	**DISTANCE_GROUP**		Ordinal	[1 – 11]	3.82	3.0	2
6	**SEGMENT_NUMBER**	The segment that this tail number is on for the day	Ordinal	[1 – 15]	3.04	3.0	2
7	**CONCURRENT_FLIGHTS**	Numeric	Numeric	[1 – 109]	27.83	23.0	10
8	**NUMBER_OF_SEATS**	Number of seats on the aircraft	Ordinal	[44-337]	133.73	134	143
9	**CARRIER_NAME**	Carrier	Categorical	17 unique values			Southwest Airlines Co.
10	**AIRPORT_FLIGHTS_MONTH**	Avg Airport Flights per Month	Numeric	969 unique values			
11	**AIRLINE_FLIGHTS_MONTH**	Avg Airline Flights per Month	Numeric	204 unique values			
12	**AIRLINE_AIRPORT_FLIGHTS_MONTH**	Avg Flights per month for Airline AND Airport	Numeric	2119 unique values			
13	**AVG_MONTHLY_PASS_AIRPORT**	Avg Passengers for the departing airport for the month	Numeric	96 unique values			

14	**AVG_MONTHLY_PASS_AIRLINE**	Avg Passengers for airline for month	Numeric	17 unique values			
15	**FLT_ATTENDANTS_PER_PASS**	Flight attendants per passenger for airline	Ordinal	14 unique values			
16	**GROUND_SERV_PER_PASS**	Ground service employees (service desk) per passenger for airline	Ordinal	17 unique values			
17	**PLANE_AGE**	Age of departing aircraft	Ordinal	[0 – 32]			
18	**DEPARTING_AIRPORT**	Departing Airport	Categorical	96 unique values			
19	**LATITUDE**	Latitude of departing airport	Numeric	96 unique values			
20	**LONGITUDE**	Longitude of departing airport	Numeric	96 unique values			
21	**PREVIOUS_AIRPORT**	Previous airport that aircraft departed from	Categorical	365 unique values			NONE
22	**PRCP**	Inches of precipitation for day	Numeric	[0.0 - 11.63]			
23	**SNOW**	Inches of snowfall for day	Numeric	[0.0 - 17.2]			
24	**SNWD**	Inches of snow on ground for day	Numeric	[0.0 - 25.2]			

25	**TMAX**	Max temperature for day	Numeric	[(-10.0) - 115.0]			
26	**AWND**	Max wind speed for day		[0.0 - 33.78]			

2.2: Data Collection

2.2.1: Import Library

```
import os
import numpy as np # Linear Algebra
import pandas as pd # Data Manipulation
from collections import Counter # Data Manipulation
import seaborn as sns # Data Viz
import matplotlib.pyplot as plt # Data Viz
from sklearn.preprocessing import LabelEncoder

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from imblearn.under_sampling import RandomUnderSampler

from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, classification_report, confusion_matrix
```

✓ 11.2s Python

2.2.2: Import Dataset

```
df = pd.read_csv('/content/drive/MyDrive/Data Mining/full_data_flightdelay.csv', encoding='utf-8')
```

```
df.head()
```

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS	CARRIER_NAME	AIRPORT_FLIGHTS_MONTH	...	PLANE_AGE
0	1	7	0	0800-0859	2	1	25	143	Southwest Airlines Co.	13056	...	8
1	1	7	0	0700-0759	7	1	29	191	Delta Air Lines Inc.	13056	...	3
2	1	7	0	0600-0659	7	1	27	199	Delta Air Lines Inc.	13056	...	18
3	1	7	0	0600-0659	9	1	27	180	Delta Air Lines Inc.	13056	...	2
4	1	7	0	0001-0559	7	1	10	182	Spirit Air Lines	13056	...	1

5 rows × 26 columns

```
df.tail()
```

Python

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS	CARRIER_
6489057	12	7	0	2300-2359	1	11	3	123	Hawaiian A
6489058	12	7	0	1800-1859	1	11	2	123	Hawaiian A
6489059	12	7	0	2000-2059	1	11	2	123	Hawaiian A
6489060	12	7	0	2100-2159	1	12	3	123	Hawaiian A
6489061	12	7	1	2100-2159	1	12	3	123	Hawaiian A

5 rows × 26 columns

2.2.1: Check data type, information

```
df.dtypes
```

✓ 0.0s

```
MONTH                           int64
DAY_OF_WEEK                     int64
DEP_DEL15                       int64
DEP_TIME_BLK                   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                       int64
DEPARTING_AIRPORT              object
LATITUDE                      float64
LONGITUDE                     float64
PREVIOUS_AIRPORT               object
PRCP                          float64
SNOW                          float64
SNWD                          float64
TMAX                          float64
AWND                          float64
dtype: object
```

2.2.2: Overview of the data

The data provides comprehensive insights into various aspects of flight operations and associated factors:

- Temporal Information: the data set stores information about flights over the months, weeks, and departure time frames throughout the year. It includes attributes **MONTH,**

DAY_OF_WEEK, and **DEP_TIME_BLK** allowing analysis based on seasonal and weekly patterns.

- Flight Information: The attributes **DISTANCE_GROUP** and **SEGMENT_NUMBER** provide detailed information about the distance group that the departing aircraft will fly and the number of flight segments in a day, which helps understand the nature and scale of the flights. Additionally, information about carriers (**CARRIER_NAME**), the number of seats on the aircraft (**NUMBER_OF_SEATS**), and the age of the departing aircraft (**PLANE_AGE**) provides insights into the operational aspects of the flights.
- Airport Data: The dataset includes data related to departing airports (**DEPARTING_AIRPORT**), such as latitude and longitude (**LATITUDE**, **LONGITUDE**), which can be utilized for geographical analysis and understanding regional patterns.
- Weather Conditions: precipitation (**PRCP**), snowfall (**SNOW**), snow on the ground (**SNWD**), maximum temperature (**TMAX**), and maximum wind speed (**AWND**) offer valuable information about the weather conditions during flight operations.
- Passenger and Service Information: Attributes provide insights into passenger-related factors and service quality such as **AVG_MONTHLY_PASS_AIRPORT**, **AVG_MONTHLY_PASS_AIRLINE**, **FLT_ATTENDANTS_PER_PASS**, **GROUND_SERV_PER_PASS**.

2.2.3: Description of data

```
#kiểm tra kiểu dữ liệu numberic
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
MONTH	6489062.0	6.607062e+00	3.396853e+00	1.000000	4.000000e+00	7.000000e+00	1.000000e+01	1.200000e+01
DAY_OF_WEEK	6489062.0	3.935598e+00	1.995200e+00	1.000000	2.000000e+00	4.000000e+00	6.000000e+00	7.000000e+00
DEP_DEL15	6489062.0	1.891441e-01	3.916231e-01	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	1.000000e+00
DISTANCE_GROUP	6489062.0	3.821102e+00	2.382233e+00	1.000000	2.000000e+00	3.000000e+00	5.000000e+00	1.100000e+01
SEGMENT_NUMBER	6489062.0	3.046890e+00	1.757864e+00	1.000000	2.000000e+00	3.000000e+00	4.000000e+00	1.500000e+01
CONCURRENT_FLIGHTS	6489062.0	2.783675e+01	2.151060e+01	1.000000	1.100000e+01	2.300000e+01	3.900000e+01	1.090000e+02
NUMBER_OF_SEATS	6489062.0	1.337397e+02	4.645213e+01	44.000000	9.000000e+01	1.430000e+02	1.720000e+02	3.370000e+02
AIRPORT_FLIGHTS_MONTH	6489062.0	1.268458e+04	8.839796e+03	1100.000000	5.345000e+03	1.156200e+04	1.761500e+04	3.525600e+04
AIRLINE_FLIGHTS_MONTH	6489062.0	6.296058e+04	3.438223e+04	5582.000000	2.503400e+04	7.087800e+04	8.631200e+04	1.177280e+05
AIRLINE_AIRPORT_FLIGHTS_MONTH	6489062.0	3.459251e+03	4.251139e+03	1.000000	6.540000e+02	2.251000e+03	4.806000e+03	2.183700e+04
AVG_MONTHLY_PASS_AIRPORT	6489062.0	1.588639e+06	1.123847e+06	70476.000000	6.732210e+05	1.486066e+06	2.006675e+06	4.365661e+06
AVG_MONTHLY_PASS_AIRLINE	6489062.0	7.814970e+06	5.046882e+06	473794.000000	2.688839e+06	8.501631e+06	1.246018e+07	1.338300e+07
FLT_ATTENDANTS_PER_PASS	6489062.0	9.753707e-05	8.644459e-05	0.000000	3.419267e-05	6.178236e-05	1.441659e-04	3.484077e-04
GROUND_SERV_PER_PASS	6489062.0	1.355612e-04	4.649970e-05	0.000007	9.889412e-05	1.246511e-04	1.772872e-04	2.289855e-04
PLANE_AGE	6489062.0	1.153211e+01	6.935706e+00	0.000000	5.000000e+00	1.200000e+01	1.700000e+01	3.200000e+01
LATITUDE	6489062.0	3.670581e+01	5.500804e+00	18.440000	3.343600e+01	3.750500e+01	4.077900e+01	6.116900e+01
LONGITUDE	6489062.0	-9.425515e+01	1.790952e+01	-159.346000	-1.063770e+02	-8.790600e+01	-8.093600e+01	-6.600200e+01
PRCP	6489062.0	1.037063e-01	3.432134e-01	0.000000	0.000000e+00	0.000000e+00	2.000000e-02	1.163000e+01
SNOW	6489062.0	3.159310e-02	3.170163e-01	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	1.720000e+01
SNWD	6489062.0	9.152397e-02	7.281285e-01	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	2.520000e+01
TMAX	6489062.0	7.146846e+01	1.835333e+01	-10.000000	5.900000e+01	7.400000e+01	8.600000e+01	1.150000e+02
AWND	6489062.0	8.341329e+00	3.607604e+00	0.000000	5.820000e+00	7.830000e+00	1.029000e+01	3.378000e+01

The description of the numerical data types in the dataset indicates that the smallest value in the LONGTITUDE field is -159, and the data points are spread out over a very large range. This wide range of values can have a significant impact on the performance of predictive models.

```
#kiểm tra kiểm dữ liệu object
df.describe(include=['O'])
```

	DEP_TIME_BLK	CARRIER_NAME	DEPARTING_AIRPORT	PREVIOUS_AIRPORT
count	6489062	6489062	6489062	6489062
unique	19	17	96	356
top	0800-0859	Southwest Airlines Co.	Atlanta Municipal	NONE
freq	452391	1296329	392603	1449009

We can see that there are 25 features and one target column (DEL_DEL15). Out of the features, 22 are numerical, and 4 are categorical.

2.2.4: Dealing with duplicate data

Checking for duplicate data. Detected 28,473 duplicate lines. Performing deletion of duplicate lines.

```
df.duplicated().sum()
```

```
28473
```

Performing deletion of duplicate data.

```
df.drop_duplicates()
```

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS	CARRIER_NAME	AIRPORT_FLIGHTS_MONTH	...	PLANE_AGE	DEPARTING_AIRPORT	LATITUDE	LONGITUDE
0	1	7	0	0800-0859	2	1	25	143	Southwest Airlines Co.	13056	...	8	McCarran International	36.080	-115.152
1	1	7	0	0700-0759	7	1	29	191	Delta Air Lines Inc.	13056	...	3	McCarran International	36.080	-115.152
2	1	7	0	0600-0659	7	1	27	199	Delta Air Lines Inc.	13056	...	18	McCarran International	36.080	-115.152
3	1	7	0	0600-0659	9	1	27	180	Delta Air Lines Inc.	13056	...	2	McCarran International	36.080	-115.152
4	1	7	0	0001-0559	7	1	10	182	Spirit Air Lines	13056	...	1	McCarran International	36.080	-115.152
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
6489057	12	7	0	2300-2359	1	11	3	123	Hawaiian Airlines Inc.	1318	...	18	Lihue Airport	21.979	-159.346
6489058	12	7	0	1800-1859	1	11	2	123	Hawaiian Airlines Inc.	1318	...	16	Lihue Airport	21.979	-159.346
6489059	12	7	0	2000-2059	1	11	2	123	Hawaiian Airlines Inc.	1318	...	18	Lihue Airport	21.979	-159.346
6489060	12	7	0	2100-2159	1	12	3	123	Hawaiian Airlines Inc.	1318	...	18	Lihue Airport	21.979	-159.346
6489061	12	7	1	2100-2159	1	12	3	123	Hawaiian Airlines Inc.	1318	...	15	Lihue Airport	21.979	-159.346

6460589 rows × 26 columns

2.2.5: Handle the null data

Check the null data.

```
[32]  1 #kiểm tra giá trị null
      2 df_droplatlng[df_droplatlng.isnull().any(axis=1)]
```

MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS	CARRIER_NAME	AIRPORT_FLIGHTS

0 rows × 24 columns

There are no null values.

2.2.6: Data Reduction

Checking the memory usage of the dataset, we found that it occupies 2.9 GB, which is indeed very large and can negatively impact the prediction and data processing phases. To address this issue, we need to take steps to reduce the memory usage of the DataFrame.

```
df.info(memory_usage="deep")
```

✓ 6.7s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6489062 entries, 0 to 6489061
Data columns (total 26 columns):
 #   Column                         Dtype
---  ------                         -----
 0   MONTH                          int64
 1   DAY_OF_WEEK                    int64
 2   DEP_DEL15                      int64
 3   DEP_TIME_BLK                   object
 4   DISTANCE_GROUP                 int64
 5   SEGMENT_NUMBER                 int64
 6   CONCURRENT_FLIGHTS             int64
 7   NUMBER_OF_SEATS                int64
 8   CARRIER_NAME                   object
 9   AIRPORT_FLIGHTS_MONTH          int64
 10  AIRLINE_FLIGHTS_MONTH          int64
 11  AIRLINE_AIRPORT_FLIGHTS_MONTH  int64
 12  AVG_MONTHLY_PASS_AIRPORT       int64
 13  AVG_MONTHLY_PASS_AIRLINE       int64
 14  FLT_ATTENDANTS_PER_PASS        float64
 15  GROUND_SERV_PER_PASS           float64
 16  PLANE_AGE                      int64
 17  DEPARTING_AIRPORT              object
 18  LATITUDE                       float64
 19  LONGITUDE                      float64
...
 24  TMAX                           float64
 25  AWND                           float64
dtypes: float64(9), int64(13), object(4)
memory usage: 2.9 GB
```

Convert integer and float data types to smaller corresponding integer or float types if possible. For columns with string (object) data types, convert them to categorical (category) data types if the number of distinct values is low. This is particularly useful when there are many repeated values.

```python
def reduce_mem_usage(df):
    """ Iterate through all the columns of a dataframe and modify the data type
        to reduce memory usage.
    """
    start_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))
    for col in df.columns:
        col_type = df[col].dtype
        if col_type == 'category':
            continue
        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max < np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
                elif c_min > np.iinfo(np.int16).min and c_max < np.iinfo(np.int16).max:
                    df[col] = df[col].astype(np.int16)
                elif c_min > np.iinfo(np.int32).min and c_max < np.iinfo(np.int32).max:
                    df[col] = df[col].astype(np.int32)
                elif c_min > np.iinfo(np.int64).min and c_max < np.iinfo(np.int64).max:
                    df[col] = df[col].astype(np.int64)
            else:
                if c_min > np.finfo(np.float16).min and c_max < np.finfo(np.float16).max:
                    df[col] = df[col].astype(np.float16)
                elif c_min > np.finfo(np.float32).min and c_max < np.finfo(np.float32).max:
                    df[col] = df[col].astype(np.float32)
                else:
                    df[col] = df[col].astype(np.float64)
        else:
            # Convert object types to category
            df[col] = df[col].astype('category')
    end_mem = df.memory_usage().sum() / 1024**2
    print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
    print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))
    return df
df = reduce_mem_usage(df)
```

✓ 10.7s

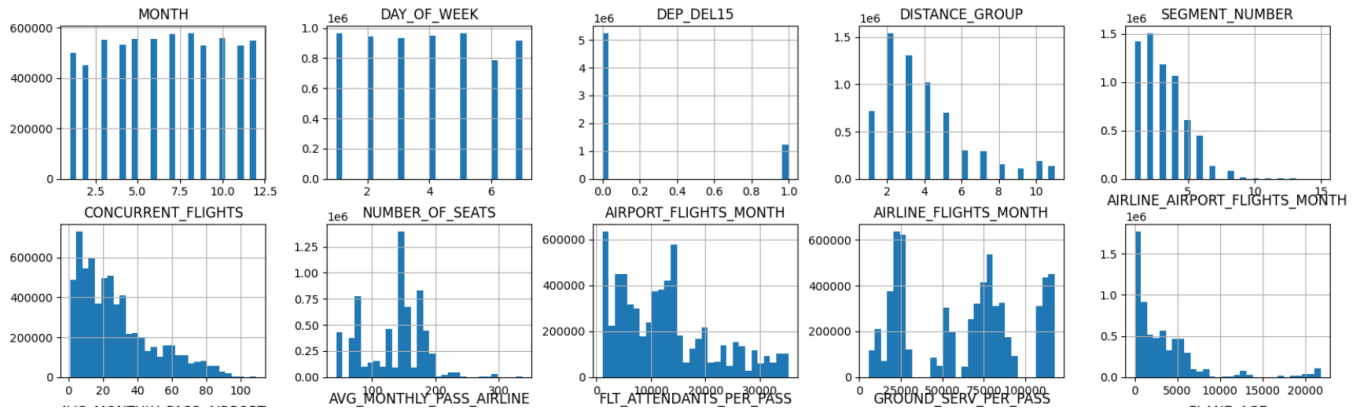
```
Memory usage of dataframe is 1287.20 MB
Memory usage after optimization is: 309.44 MB
Decreased by 76.0%
```

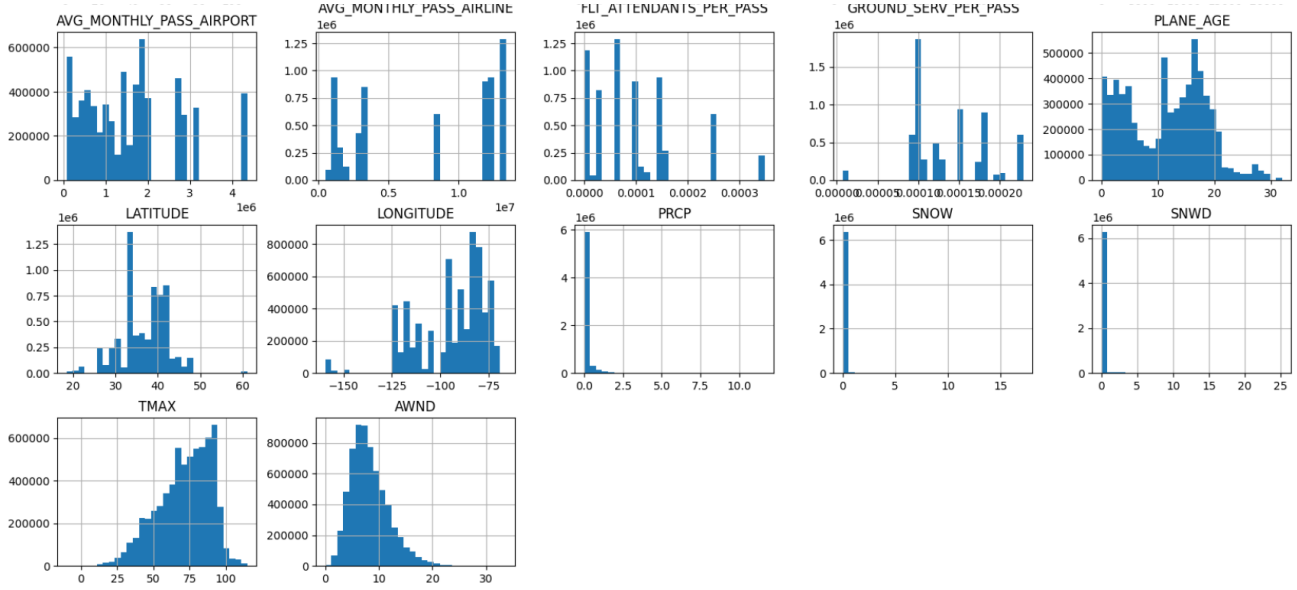
2.3: Data Visualization

2.3.1.1: Variable Distribution

Phân phối biến số

```python
# Phân phối
df.hist(bins=30, figsize=(20, 15))
plt.show()
```





2.3.1.2: MONTH

Check the values of the 'MONTH' column and we see that the values in the 'MONTH' column are valid, with no values less than 1 or greater than 12.

```
#Month
# Số lượng giá trị duy nhất của cột MONTH
unique_months = df['MONTH'].nunique()
unique_month_values = df['MONTH'].unique()
print("Các giá trị duy nhất của cột MONTH:")
print(unique_month_values)
```

✓ 0.1s

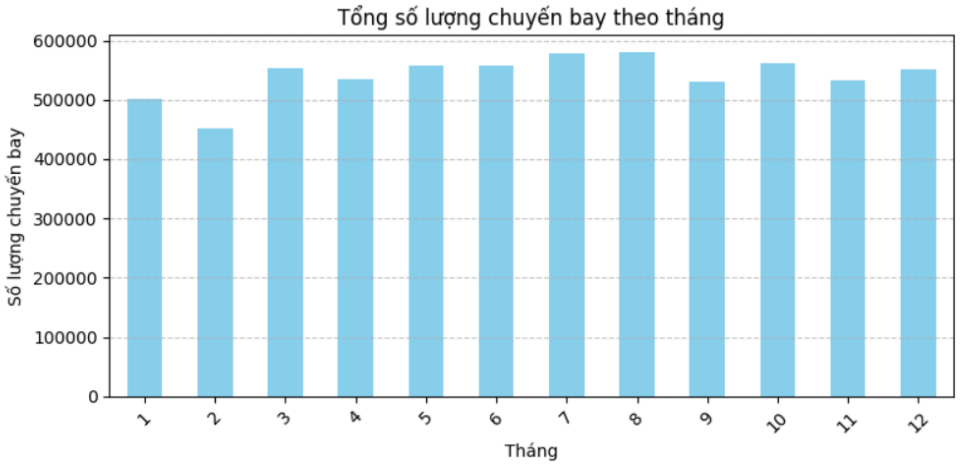
```
Các giá trị duy nhất của cột MONTH:
[ 1  2  3  4  5  6  7  8  9 10 11 12]
```

The graph showing the number of flights across each month in 2019. There is a difference between the number of flights in the months, with around 580,000 flights in June/July compared to around 450,000 flights in February.

The difference can be attributed to seasonality, where during the summer months, especially June and July, the demand for air travel is higher as more people travel for leisure, and the favorable weather conditions allow airlines to increase the number of flights. In contrast, during February, the demand for air travel is lower due to the winter season and the potential for disruptions caused by snow and ice, which can lead to flight cancellations or delays.

```python
# Tính tổng số lượng chuyến bay của từng tháng và vẽ biểu đồ
plt.figure(figsize=(8, 4))
df.groupby('MONTH').size().plot(kind='bar', color='skyblue')
plt.title('Tổng số lượng chuyến bay theo tháng')
plt.xlabel('Tháng')
plt.ylabel('Số lượng chuyến bay')
plt.xticks(rotation=45)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.tight_layout()
plt.show()
```

✓ 0.4s

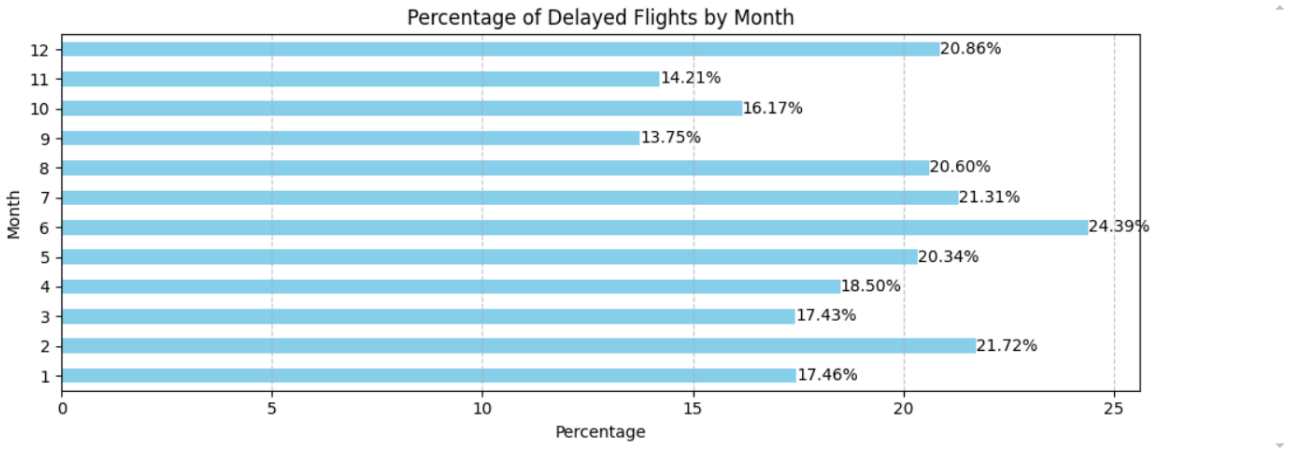


Flight Delay Rates: June had the highest flight delay rate of 24.39%, possibly due to increased travel demand during the summer season leading to congestion or unfavorable weather conditions. September had the lowest flight delay rate of 13.75%, which may be attributed to more stable weather conditions during the fall season.

Monthly Variations: There are significant variations between months in both the number of flights and flight delay rates. This may reflect the influence of factors such as seasonality, weather conditions, and fluctuations in travel and work schedules.

```
plt.figure(figsize=(10, 4))
bars = delayed_percentage_per_month.plot(kind='barh', color='skyblue')
plt.title('Percentage of Delayed Flights by Month')
plt.xlabel('Percentage')
plt.ylabel('Month')
plt.grid(axis='x', linestyle='--', alpha=0.7)
plt.tight_layout()
for bar in bars.patches:
    plt.text(bar.get_width(), bar.get_y() + bar.get_height() / 2, f'{bar.get_width():.2f}%',
             va='center', ha='left', color='black', fontsize=10)

plt.show()
```

✓ 0.2s Python



2.3.1.3: DAY_OF_WEEK

Check the values of the “DAY_OF_WEEK" attribute are valid, representing the days of the week through numbers from [1-7] corresponding to Monday through Sunday.

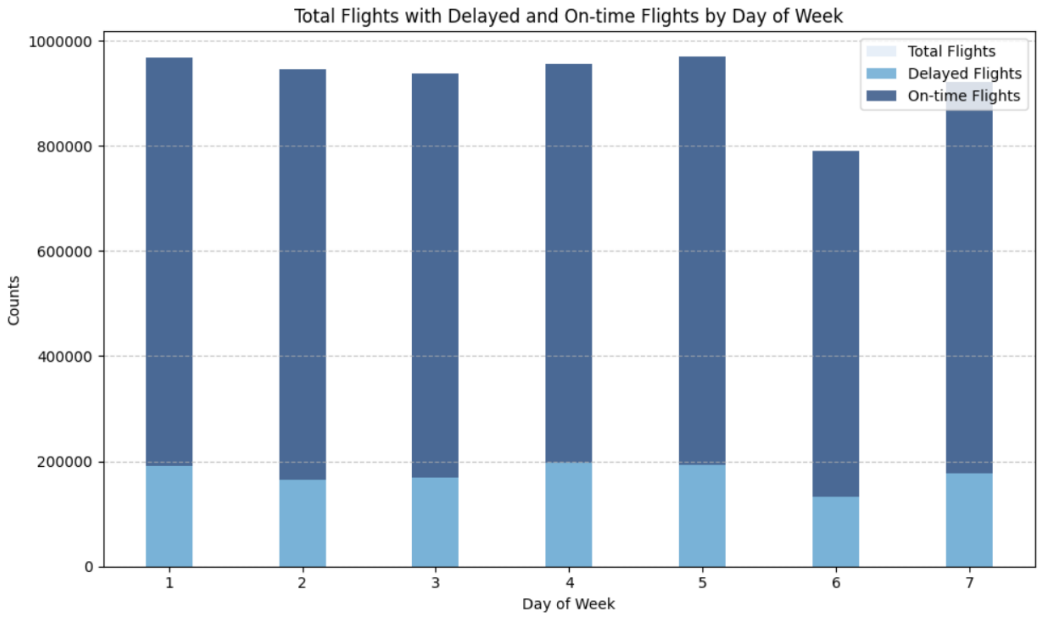
```
#DAY_OF_WEEK
# QUANTYTI UNIQUE DAY OF WEEK VALUE
print(df['DAY_OF_WEEK'].unique())
```

✓ 0.1s

```
[7 5 3 4 2 1 6]
```

```
import matplotlib.ticker as ticker
plt.figure(figsize=(10, 6))
colors = plt.cm.Blues(np.linspace(0.2, 1, 3))
bar_width = 0.35
index = np.arange(len(delayed_percentage_per_day))

#Cột tổng số lượng chuyến bay
plt.bar(index, total_flights_per_day, bar_width, color=colors[0], alpha=0.5, label='Total Flights')
#Cột số chuyến bay trễ
plt.bar(index, delayed_flights_per_day, bar_width, color=colors[1], alpha=0.7, label='Delayed Flights')
#Cột số chuyến bay đúng giờ
ontime_flights_per_day = total_flights_per_day - delayed_flights_per_day

plt.bar(index, ontime_flights_per_day, bar_width, bottom=delayed_flights_per_day, color=colors[2], alpha=0.7, label='On-time Flights')
plt.xlabel('Day of Week')
plt.ylabel('Counts')
plt.title('Total Flights with Delayed and On-time Flights by Day of Week')

plt.xticks(index, delayed_percentage_per_day.index)
plt.ticklabel_format(style='plain', axis='y')
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight_layout()
plt.show()
```

✓ 0.1s



In 2019, on Fridays there were fewer flights compared to the other days of the week. However, in general, the days with more flights also had a higher rate of flight delays, suggesting that the day of the week does not significantly affect the likelihood of flight delays.

2.3.1.4: DEP_TIME_BLK

To check the values of the departure time blocks and the data type of the column. This column represents departure time blocks within a day, so the data belongs to the categorical type, with each value representing a time block. Proceed with converting the column data type to 'category'.

```
#DEP_TIME_BLK
# QUANTYTI UNIQUE DEPARTURE TIME BLOCK VALUE
print(df['DEP_TIME_BLK'].unique())
print(df['DEP_TIME_BLK'].dtype)
```

✓ 0.1s

```
['0800-0859' '0700-0759' '0600-0659' '0001-0559' '2300-2359' '1200-1259'
 '0900-0959' '1000-1059' '2200-2259' '1500-1559' '1100-1159' '2000-2059'
 '1400-1459' '1300-1359' '1800-1859' '1900-1959' '1600-1659' '1700-1759'
 '2100-2159']
object
```

Converting the string data type to category is indeed appropriate for this dataset. It helps avoid confusion and reduces memory usage for the data.

```
# Chuyển đổi cột sang dạng dữ liệu category
df['DEP_TIME_BLK'] = df['DEP_TIME_BLK'].astype('category')
print(df['DEP_TIME_BLK'].dtype)
```

✓ 0.0s

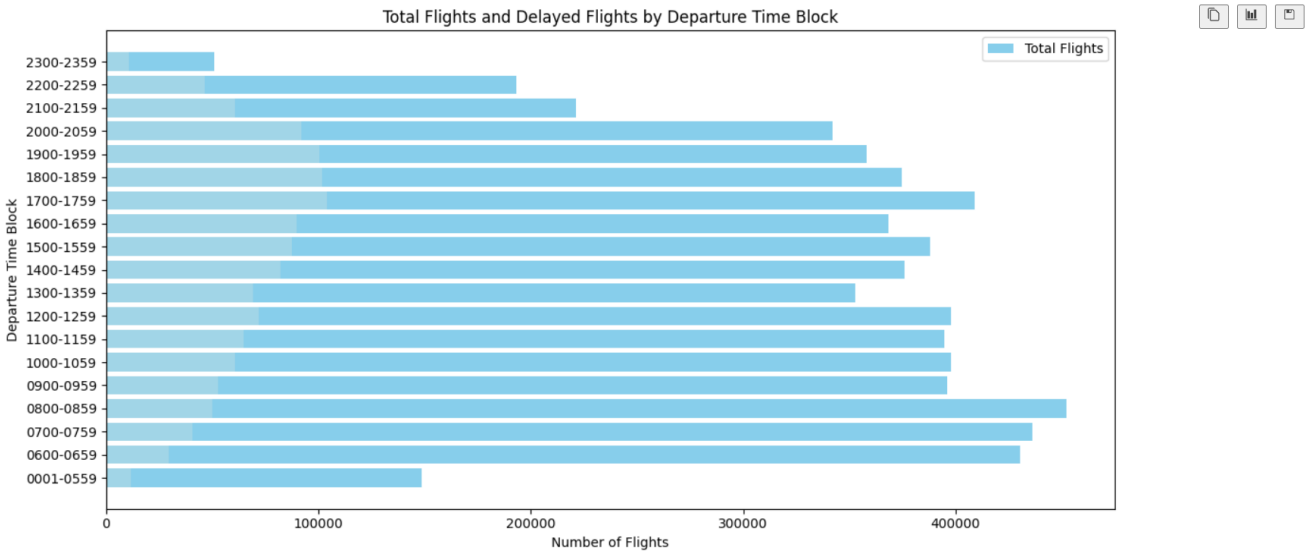
```
category
```

To create a chart representing the total number of flights within each time block and the total number of delayed flights within each time block.

```
flights_per_block = df.groupby('DEP_TIME_BLK').size()
delayed_flights_per_block = df[df['DEP_DEL15'] == 1].groupby('DEP_TIME_BLK').size()

plt.figure(figsize=(12, 6))
plt.barh(flights_per_block.index, flights_per_block, color='skyblue', label='Total Flights')
for index, value in enumerate(delayed_flights_per_block):
    plt.barh(delayed_flights_per_block.index[index], value, color='lightblue', alpha=0.7)

plt.ylabel('Departure Time Block')
plt.xlabel('Number of Flights')
plt.title('Total Flights and Delayed Flights by Departure Time Block')
plt.legend()
plt.tight_layout()
plt.show()
```

✓ 3.6s

Based on the chart, we see that there are very few flights late at night or early in the morning. The morning has more flights compared to other times of the day. The chart shows that morning flights are generally less affected and increase gradually over time. This could be due to delay propagation since they are the first flights of the day, and flights later in the day have a higher likelihood of delays.



To better examine this, let's print out the flight delay percentages in descending order. This will allow us to clearly see that the flights within the 5pm-7pm time frame have the highest percentage of delays, as explained earlier. The possibility of later flights being delayed is higher due to the cascading effect from previous delayed flights. This is a key variable that needs to be considered when predicting whether a flight will be delayed or not.

```
delays due to 'DEP_TIME_BLK'?
time_blk = flight[['DEP_TIME_BLK','DEP_DEL15']].groupby('DEP_TIME_BLK').sum().sort_values(by='DEP_DEL15',ascending=False)
time_blk['PERCENTUAL'] = time_blk['DEP_DEL15']/(time_blk['DEP_DEL15'].sum())*100
time_blk
```

✓ 1.5s

DEP_TIME_BLK	DEP_DEL15	PERCENTUAL
1700-1759	104046	8.477164
1800-1859	101993	8.309896
1900-1959	100564	8.193468
2000-2059	91811	7.480316
1600-1659	89612	7.301152
1500-1559	87336	7.115714
1400-1459	82138	6.692206
1200-1259	71869	5.855538
1300-1359	69102	5.630096
1100-1159	64848	5.283501
2100-2159	60873	4.959637
1000-1059	60643	4.940898
0900-0959	52868	4.307429
0800-0859	50125	4.083942
2200-2259	46556	3.793157
0700-0759	40685	3.314817
0600-0659	29746	2.423560
0001-0559	11749	0.957252
2300-2359	10804	0.880258

2.3.1.5: DISTANCE GROUP

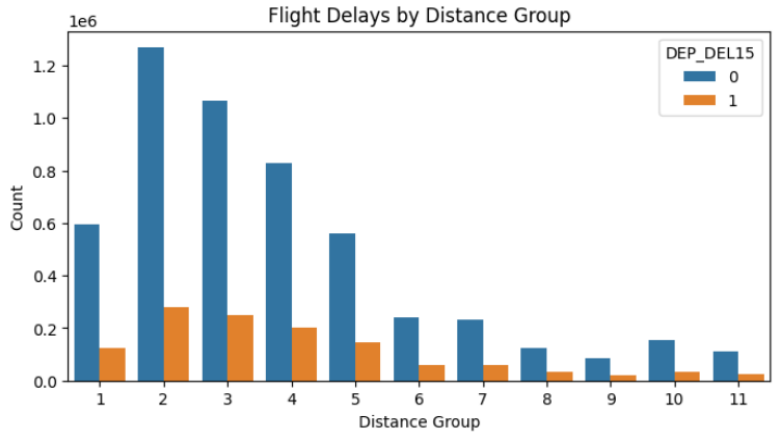
DISTANCE_GROUP is the Distance group to be flown by departing aircraft.

```
#DISTANCE_GROUP
print(df['DISTANCE_GROUP'].value_counts())
```

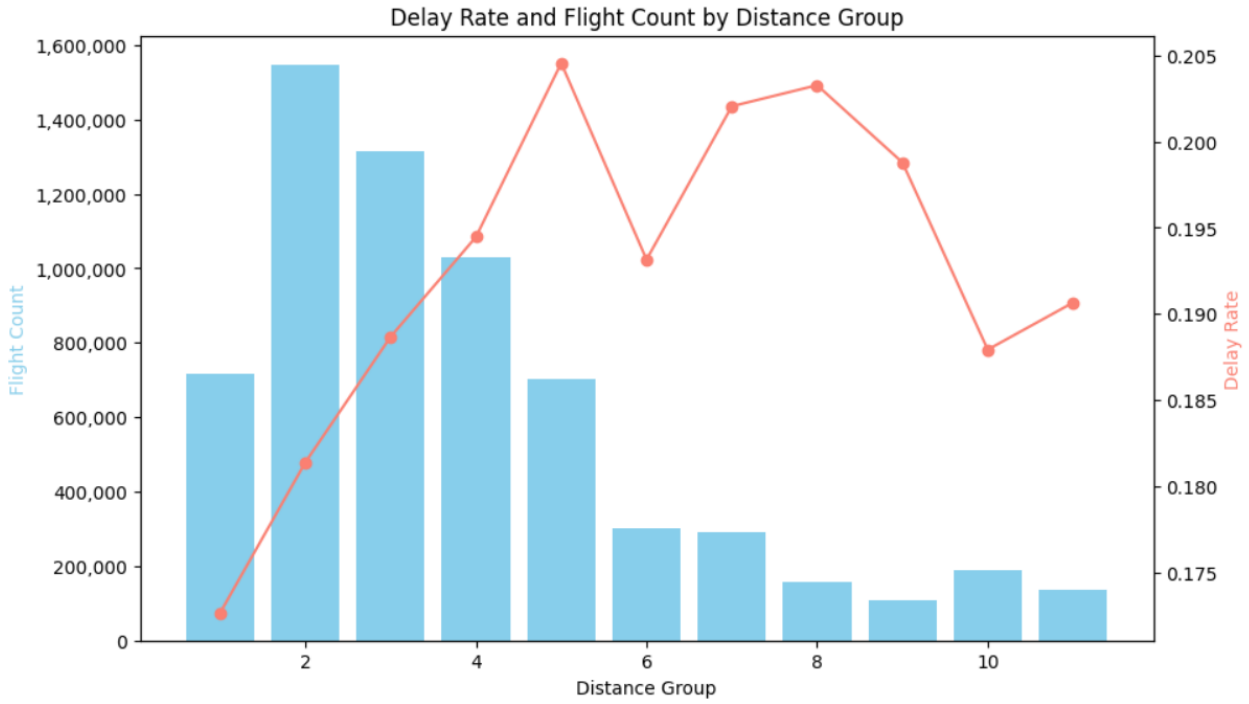
✓ 0.4s

```
DISTANCE_GROUP
2     1547851
3     1315004
4     1029580
1      717985
5      703113
6      300148
7      289456
10     188059
8      155872
11     135096
9      106898
Name: count, dtype: int64
```

```
#DISTANCE_GROUP
plt.figure(figsize=(8, 4))
sns.countplot(data=df, x='DISTANCE_GROUP', hue='DEP_DEL15')
plt.title('Flight Delays by Distance Group')
plt.xlabel('Distance Group')
plt.ylabel('Count')

plt.show()
```



To illustrate the relationship between flight delays and the size of distance groups, we can use a combined line and bar chart.



From the chart, we can see that the larger distance groups have fewer flights, but the rate of flight delays for long-distance flights is higher. This can be explained by the fact that long-

distance flights typically have more complex schedules, with multiple intermediate stops. This increases the risk of delays at any point.

Flights are mainly concentrated in the [1 - 5] distance group: this is due to higher demand for travel on shorter routes compared to longer routes, and airlines often prioritize shorter flights as they are easier to manage, have lower risk, and have less propagation impact.

In summary, factors such as complexity, ripple effects, manageability, and market demand mean that long-distance flights are usually more prone to delays and have fewer flights overall…

2.3.1.6: SEGMENT_NUMBER

Check the value of the variable **SEGMENT_NUMBER** and the quantity of each variable.

```
# SEGMENT_NUMBER
df['SEGMENT_NUMBER'].value_counts()
```

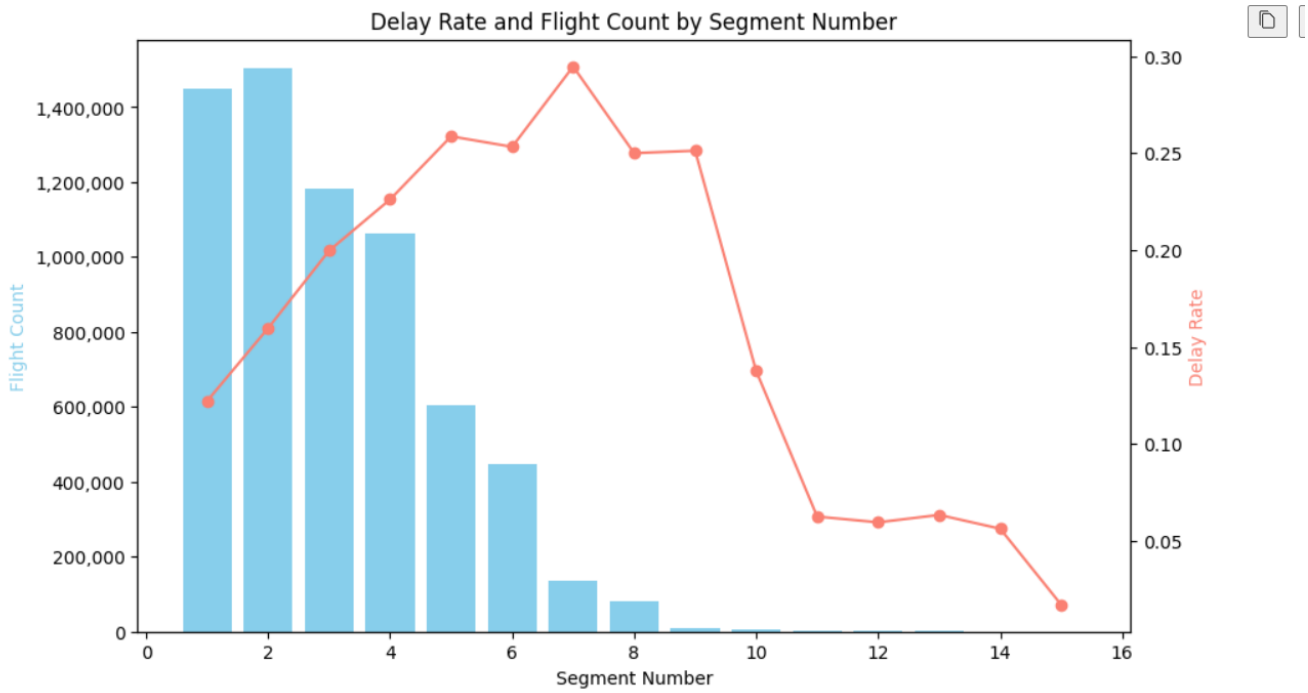
✓ 0.0s

```
SEGMENT_NUMBER
2     1505083
1     1449009
3     1182583
4     1062789
5      605609
6      446805
7      136254
8       79636
9       10151
10       4957
11       3090
12       1917
13        853
14        267
15         59
Name: count, dtype: int64
```

The number of flights decreases as the number of segments increases: flights with [1-4] segments have the highest numbers, at 1.449.009; 1.505.083, and 1.182.583 flights respectively. As the number of segments increases, the number of flights declines, with only over 59 flights having 15 segments.

The distribution is predominantly within the range of 2-6 segments: the majority of flights (around 85%) have between 2 and 6 segments. ➔ This suggests that direct flights or those with few intermediate stops are the most common.

There are few flights with an excessively high number of segments: flights with a very high number of segments (8 or more) are quite rare, numbering only in the tens of thousands.



Examining the distribution of delayed flights by segment number, it appears the data is primarily concentrated in the range of [1-7] segments.

2.3.1.7: CARRIER_NAME

```
# CARRIER_NAME
df['CARRIER_NAME'].unique()
```

✓ 0.7s

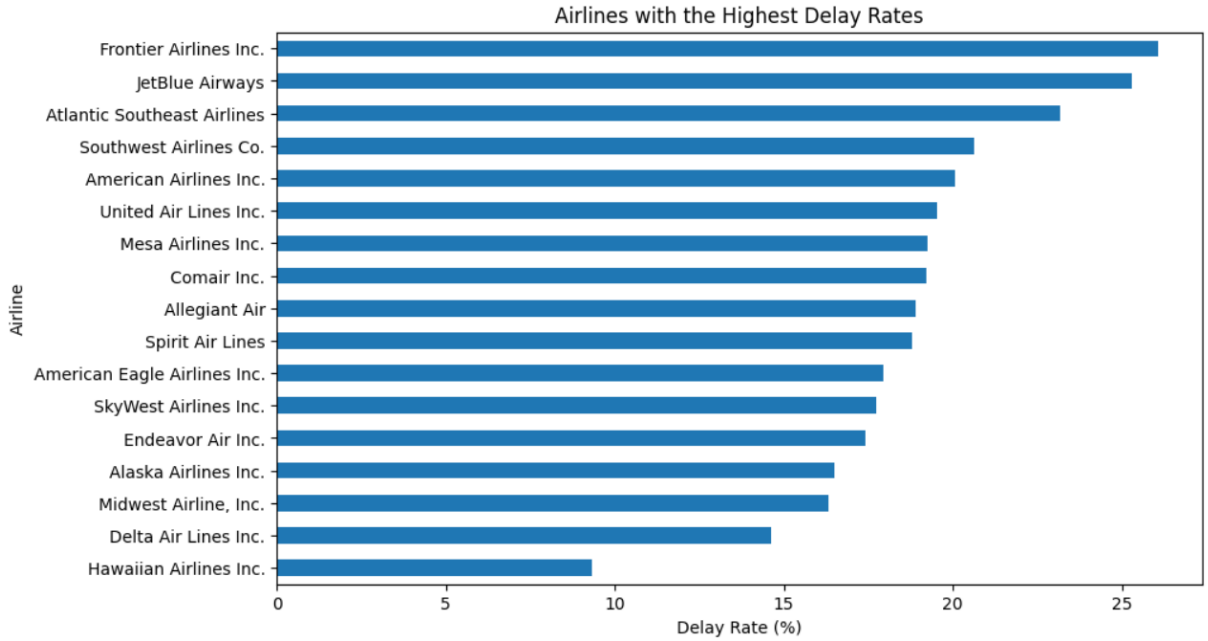
```
array(['Southwest Airlines Co.', 'Delta Air Lines Inc.',
       'Spirit Air Lines', 'Frontier Airlines Inc.',
       'Alaska Airlines Inc.', 'Hawaiian Airlines Inc.',
       'American Airlines Inc.', 'United Air Lines Inc.',
       'JetBlue Airways', 'Allegiant Air', 'SkyWest Airlines Inc.',
       'Mesa Airlines Inc.', 'American Eagle Airlines Inc.',
       'Midwest Airline, Inc.', 'Comair Inc.', 'Endeavor Air Inc.',
       'Atlantic Southeast Airlines'], dtype=object)
```

There are 17 commercial airline companies operating in the United States.

Draw a chart to show the proportion of airlines that lack punctuality and reliability.

```
# Calculate the delay percentage for each airline
delay_percentage = df.groupby('CARRIER_NAME')['DEP_DEL15'].mean() * 100
delayed_carriers = delay_percentage.sort_values(ascending=False)

# Plot the horizontal bar chart
plt.figure(figsize=(10, 6))
delayed_carriers.plot(kind='barh')
plt.xlabel('Delay Rate (%)')
plt.ylabel('Airline')
plt.title('Airlines with the Highest Delay Rates')
plt.gca().invert_yaxis()  # Invert the y-axis to display the airline with the highest delay rate first
plt.show()
```

✓ 0.6s

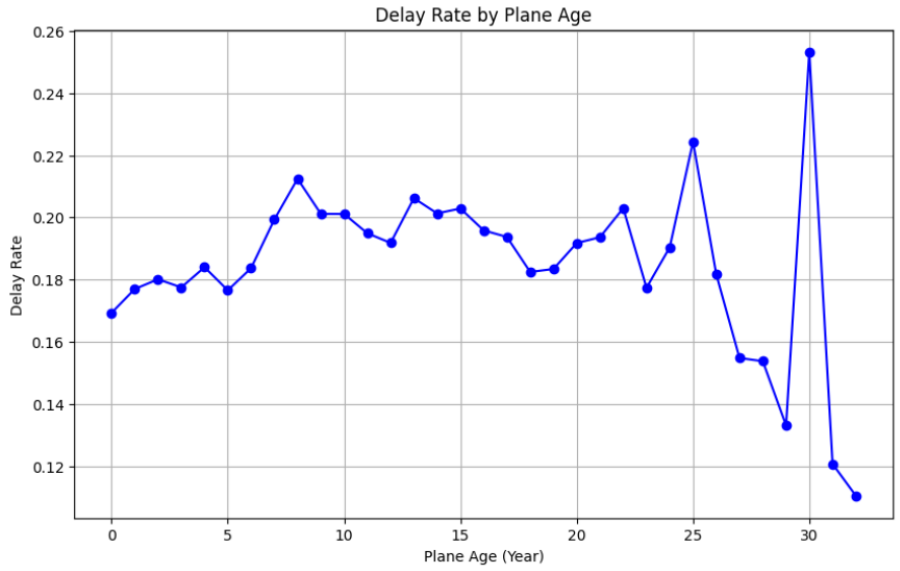


Frontier Airlines Inc. and **JetBlue Airways** are the airlines with the least reliable on-time performance. **Hawaiian Airlines Inc.** is the best choice for on-time flight operations.

The airline name cannot naturally affect whether flights are delayed, but rather it is the operations, number of flights, or other issues that influence on-time performance. Therefore, this column is not very useful for prediction, so it should be removed.

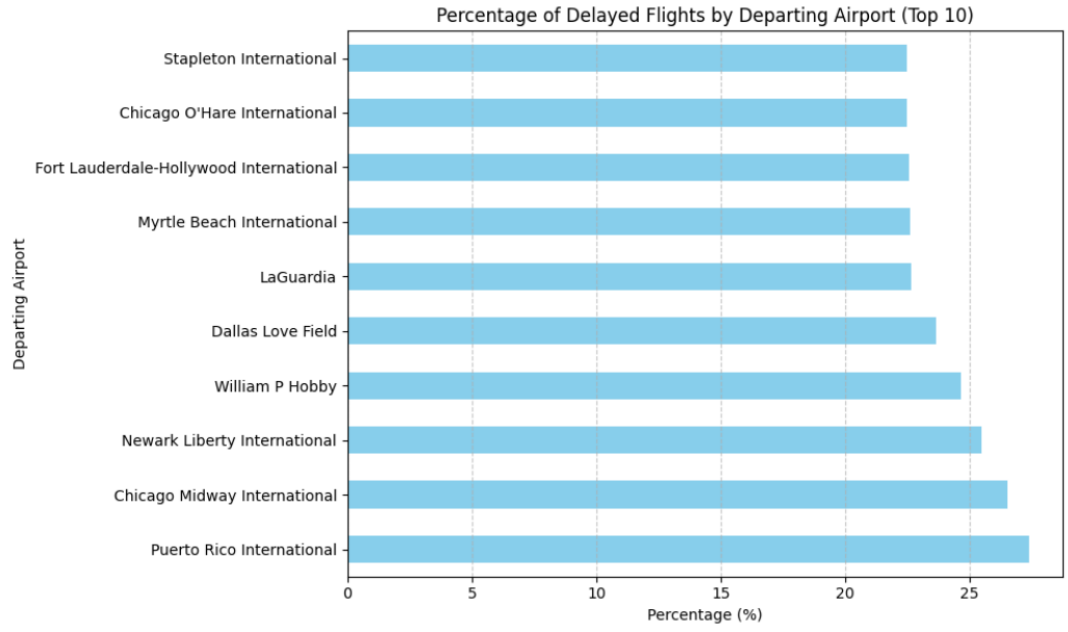
2.3.1.8: Plane age

Draw a distribution chart of flight delays according to the age of the aircraft.



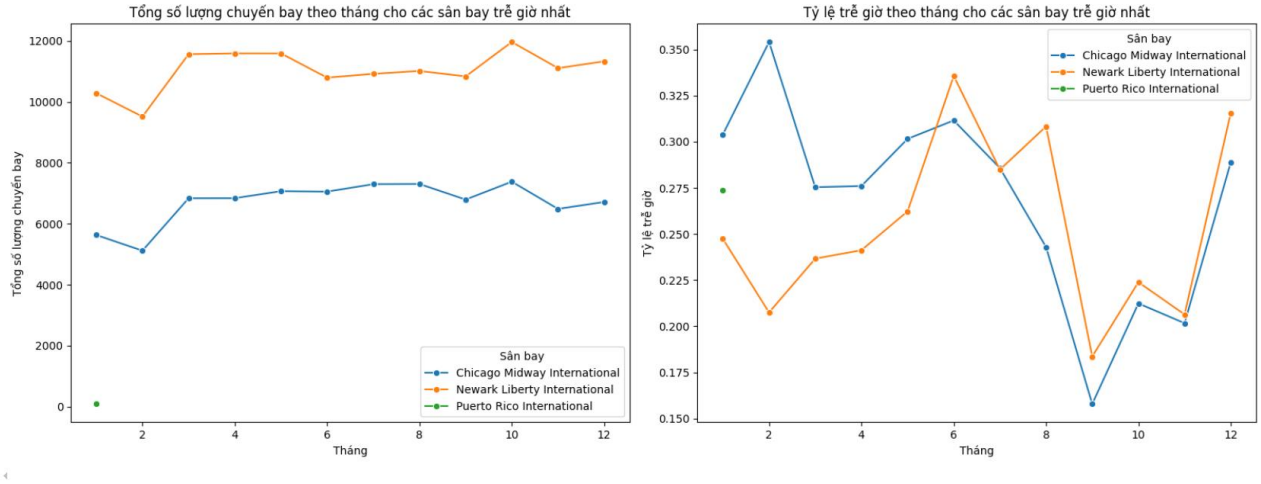
2.3.1.9: Airport

A chart showing the top 10 airports with the highest delay rates.



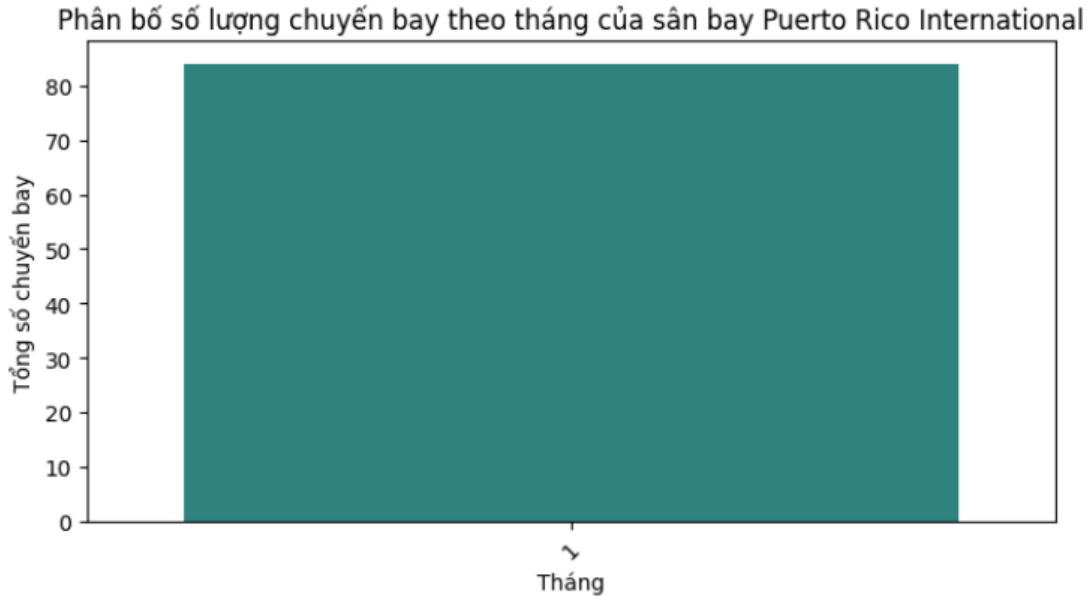
Through the chart, it shows that three airports (Puerto Rico International, Chicago Midway International, Newark Liberty International) have the highest delay rates above 25%. Select these three airports for analyzing the reasons affecting delays at these airports.

Use subplots to draw charts displaying the total number of flights and the delay rate of the top 3 airports with the highest delays per month.



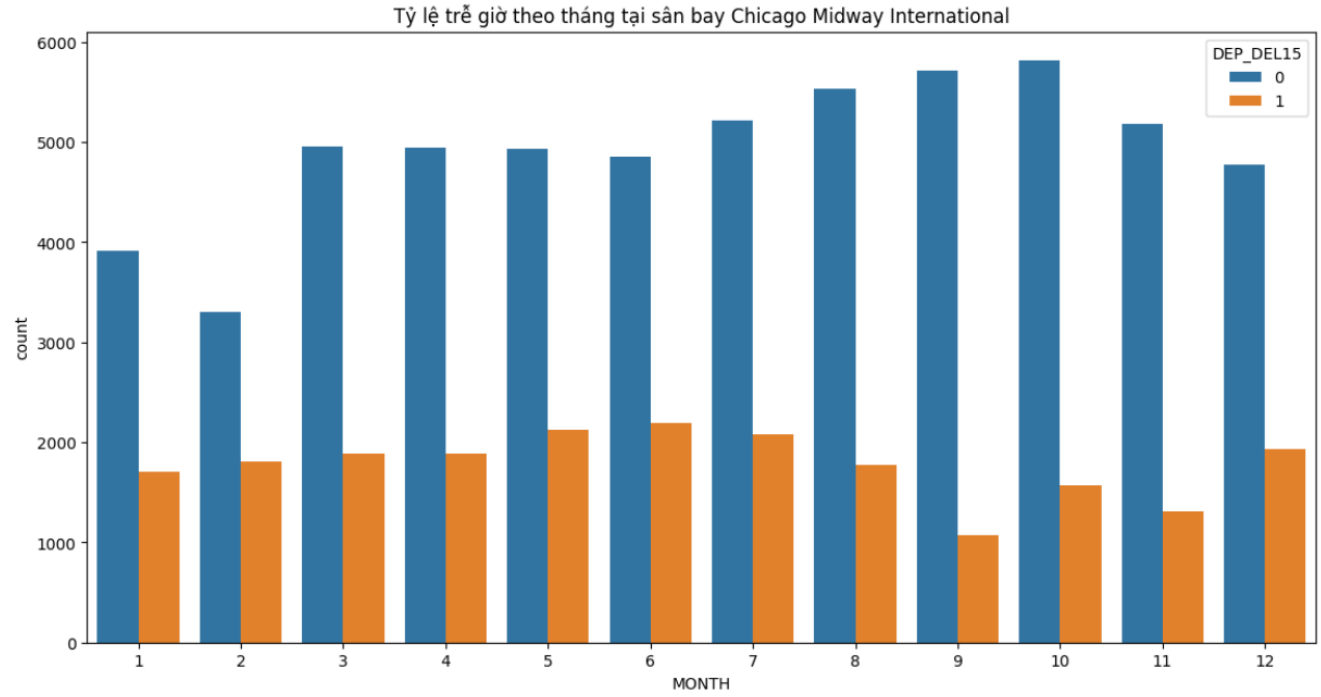
It can be seen that the total number of flights decreases in February, November, and December. The delay rate is high in February, June, July, and December. This may be due to weather conditions during December to February, which is the snowy season, affecting delays at the airports.

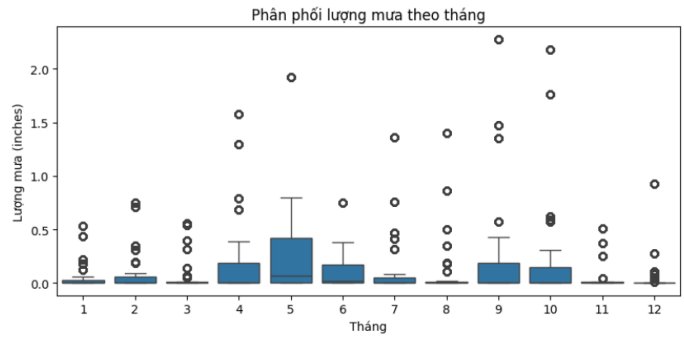
An anomaly is detected at Puerto Rico International Airport with only one data point in January. Conduct further analysis and provide conclusions.

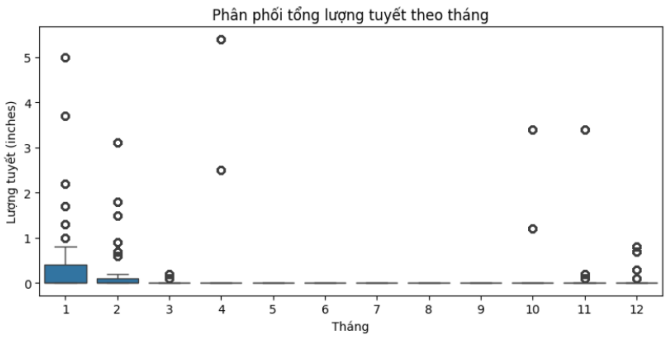


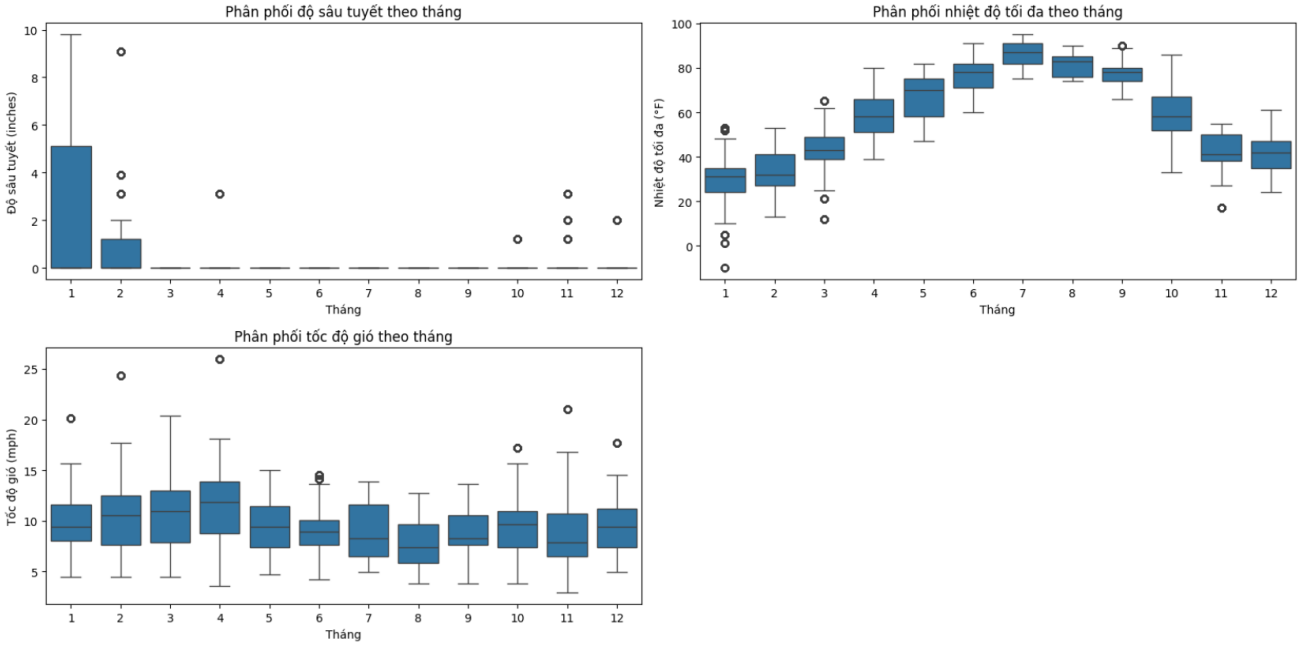
Data is recorded for Puerto Rico International Airport only in January. Upon investigation, Puerto Rico International Airport is the busiest airport in Puerto Rico, with 9.4 million passengers recorded in 2019. This indicates that the collected dataset is insufficient for Puerto Rico International Airport.

Analyze Chicago Midway International to examine the on-time and delay rates at the airport each month.







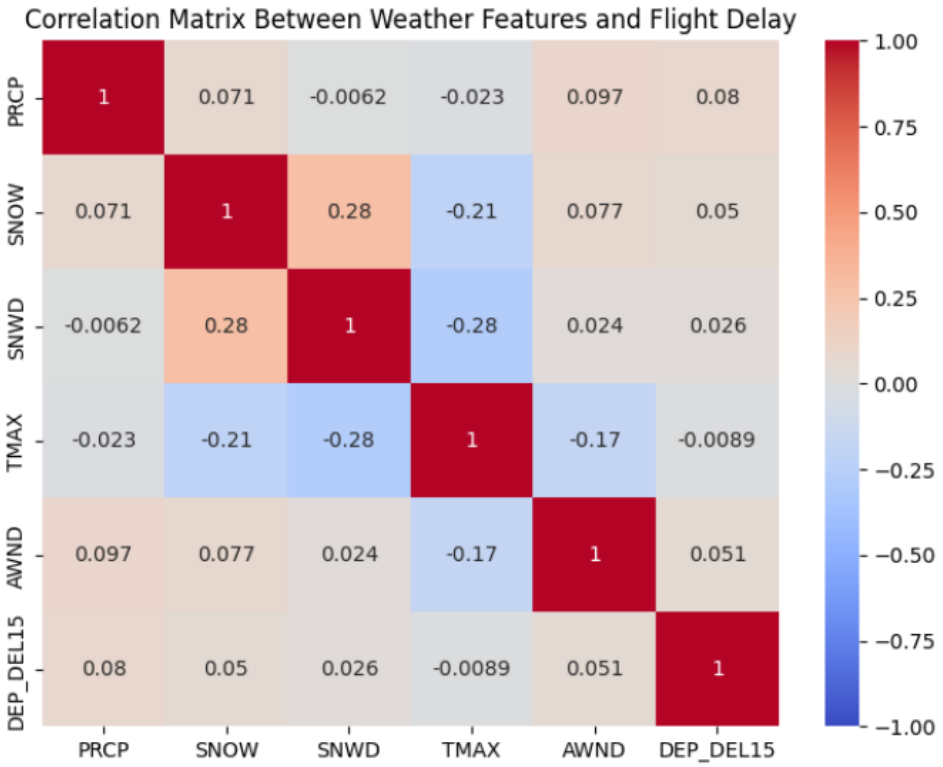


Through the chart, it is observed that months with high rainfall, low temperatures, or heavy snowfall have higher delay rates, indicating that weather is a factor affecting flight delays.

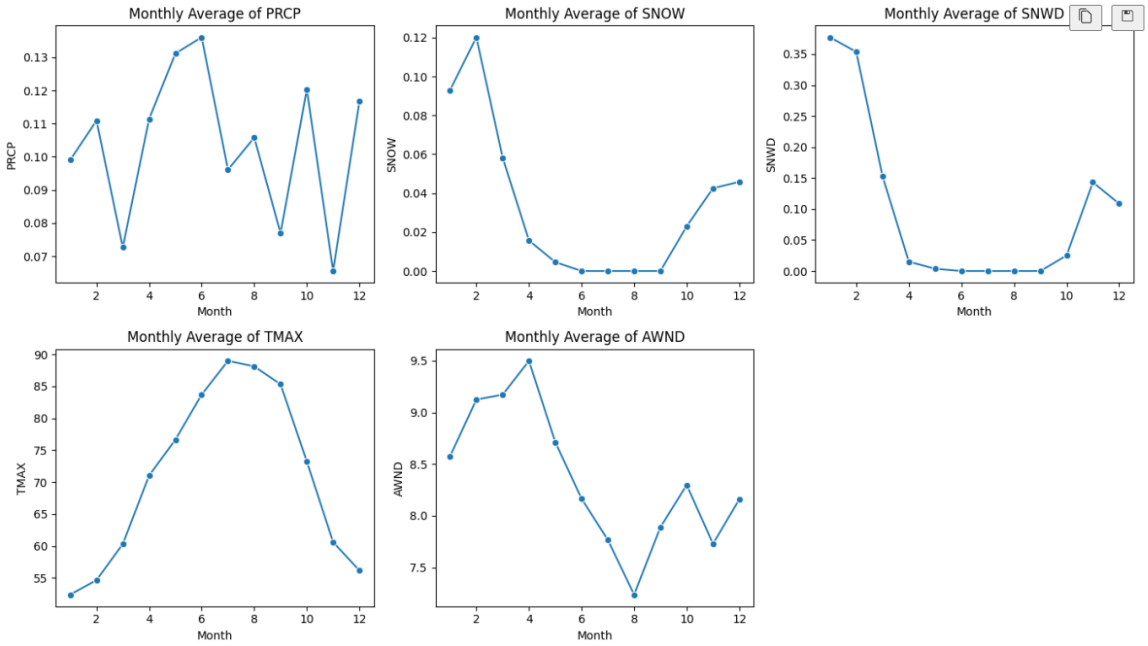
2.3.1.10: Weather futures

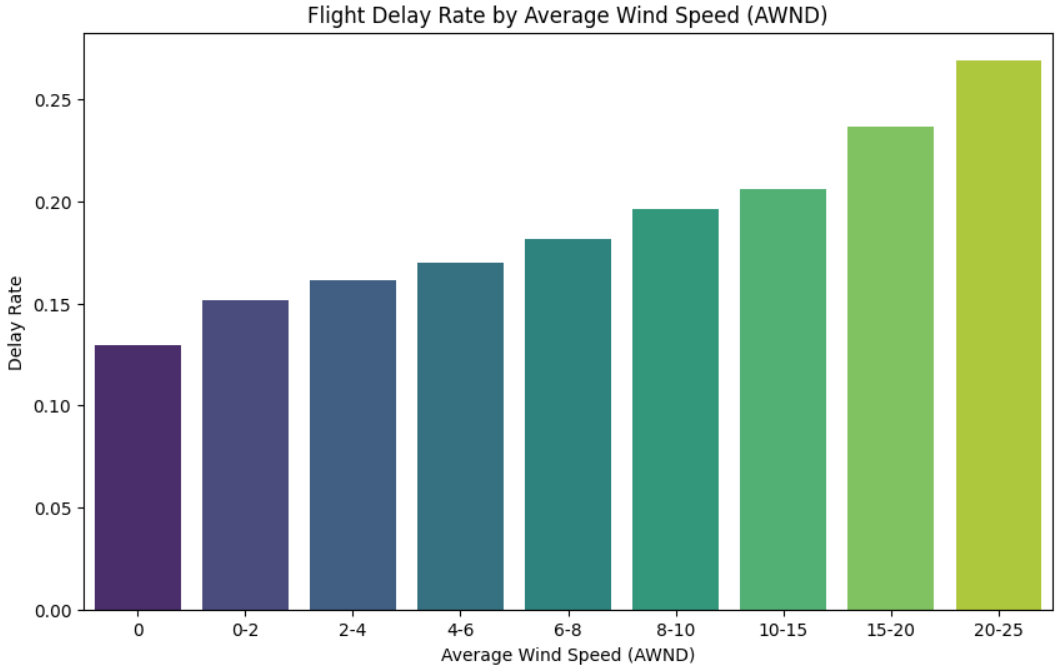
Use a heatmap to represent the relationship between weather attributes. From the graph, it can be observed that there is a positive correlation between the amount of snow on the ground and the daily snowfall. Temperature also affects these two factors, but in a negative correlation - the lower the temperature, the more likely it is to have more snow.

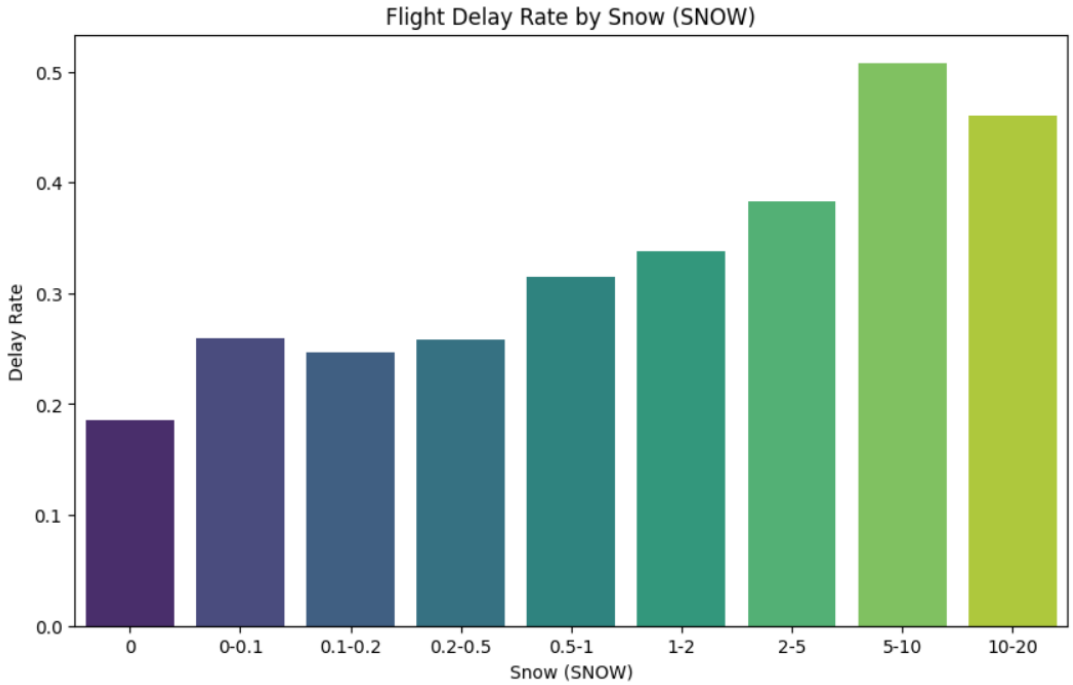
The temperature variables also have a partial influence on flight delays, but "TMAX" has the weakest impact and can be considered the most negligible.



Create a graph to represent the variation of weather conditions throughout the months in a year.







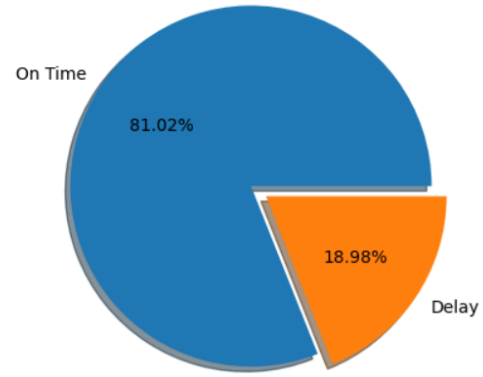
The graphs should depict the distribution of weather conditions and their impact on flight delays. The rate of delays increases as the amount of snowfall increases, and strong winds also cause more flight delays.

2.4: Data processing data

2.4.1: Check the balance data of the predictor attribute

```
#check the balance data
sizes = [df.DEP_DEL15[df["DEP_DEL15"]==0].count(), df.DEP_DEL15[df["DEP_DEL15"]== 1].count()]
labels = ["On Time", "Delay"]
plt.pie(sizes, labels= labels, explode= (0, 0.1), shadow= True, autopct= "%.2f%%")
plt.show()
```

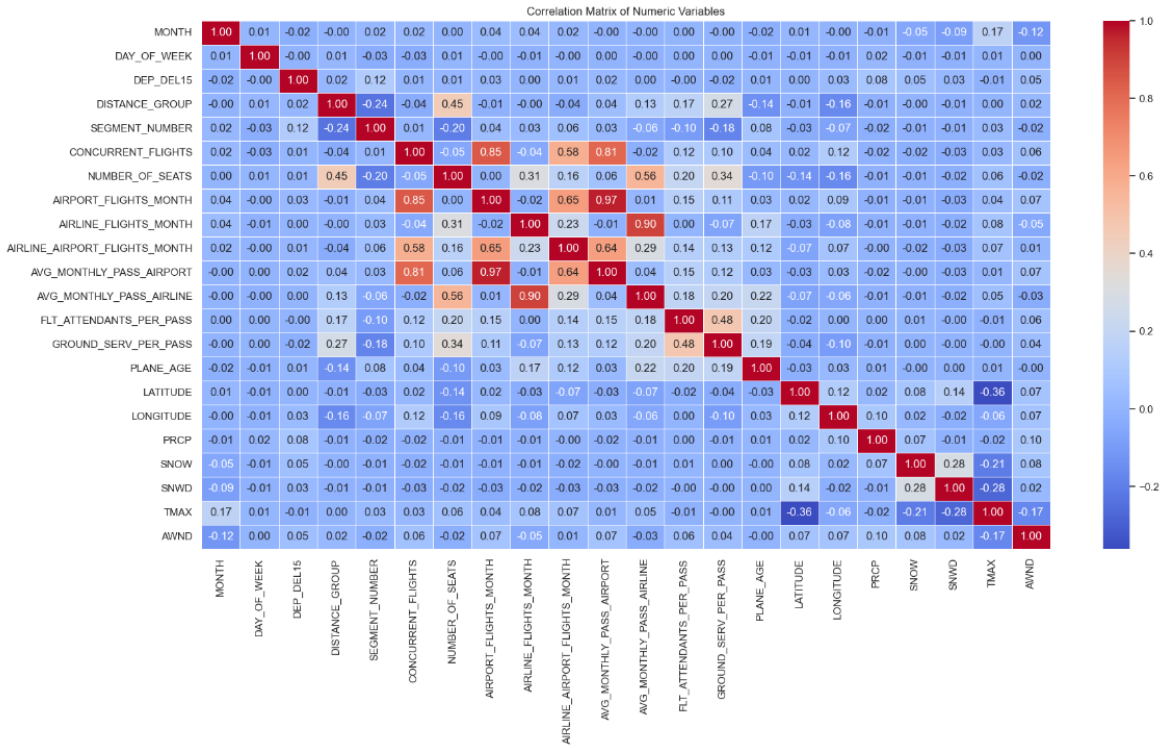
✓ 1.4s



DEP_DEL15 TARGET Binary of a departure delay over 15 minutes (1 is yes).

When checking the data file, we see that out of 6.489.061 flights, 82 percent are on time, and 18,91 percent are delayed. Here, the class “On time” is called the **majority class**, and the much smaller in size “Delay” class is called the **minority class.** Measures to handle data imbalance must be used before proceeding with model training.

2.4.2: Correlation Coefficient



Using the correlation matrix to calculate the correlation coefficients between variables:

- **DAY_OF_WEEK, AIRLINE_FLIGHTS_MONTH, AVG_MONTHLY_PASS_AIRLINE, FLT_ATTENDANTS_PER_PASS,** and **LATITUDE** variables do not correlate (Corr = 0) with the target variable, it can be removed from the dataset as it does not provide necessary information for the model.
- **CONCURRENT_FLIGHTS, AIRPORT_FLIGHTS_MONTH,** and **AVG_MONTHLY_PASS_AIRPORT** have strong correlations with each other. This could affect the model prediction process. Remove some columns such as **CONCURRENT_FLIGHTS** and **AIRPORT_FLIGHTS_MONTH** and keep the column **AVG_MONTHLY_PASS_AIRPORT.**
- **AVG_MONTHLY_PASS_AIRLINE** and **AIRLINE_FLIGHTS_MONTH** have a very high correlation coefficient of around **0.9**. A high correlation coefficient between two variables can affect the model prediction results.

Perform the removal of attribute fields and review the dataset again.

```
df = df.drop(['DAY_OF_WEEK', 'AIRLINE_FLIGHTS_MONTH','AVG_MONTHLY_PASS_AIRLINE','FLT_ATTENDANTS_PER_PASS','LATITUDE',
'AVG_MONTHLY_PASS_AIRPORT'], axis=1)
df.head()
# AVG_MONTHLY_PASS_AIRPORT do hệ số tương quan với AIRPORT_FLIGHTS_MONTH cao, trên 90%
```

Python

	MONTH	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	CARRIER_NAME	AIRPORT_FLIGHTS_MONTH	AIRLIN
0	1	0	0800-0859	2	1	25	Southwest Airlines Co.	13056	
1	1	0	0700-0759	7	1	29	Delta Air Lines Inc.	13056	
2	1	0	0600-0659	7	1	27	Delta Air Lines Inc.	13056	
3	1	0	0600-0659	9	1	27	Delta Air Lines Inc.	13056	
4	1	0	0001-0559	7	1	10	Spirit Air Lines	13056	

2.4.3: DEPARTING AIRPORT

```
df['DEPARTING_AIRPORT'].value_counts()
```

✓ 0.3s

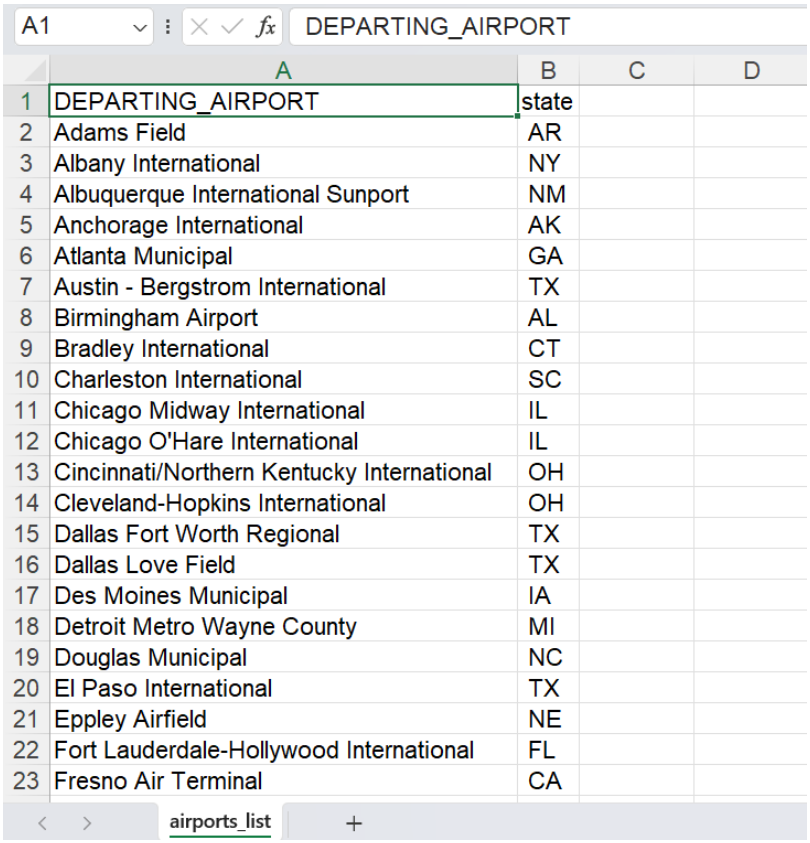
```
DEPARTING_AIRPORT
Atlanta Municipal                392603
Chicago O'Hare International     329045
Dallas Fort Worth Regional       296449
Stapleton International          247175
Douglas Municipal                231855
                                  ...
Portland International Jetport     3609
Pensacola Regional                 3578
Spokane International              3421
Sanford NAS                        2293
Puerto Rico International            84
Name: count, Length: 96, dtype: int64
```

The departing airport is the airport from which the flight will depart, and the dataset also contains information about the longitude and latitude of the airport. The departing airport can affect the flight delay through the information about the location, which may be influenced by weather conditions that can impact the flight, causing delays or on-time arrivals, depending on the number of passengers arriving at the airport and whether there is overcrowding, the number of staff to handle the passengers, and the efficiency in addressing airport-related issues.

The departing airport attribute consists of 96 different airports in the United States, with a string data type.

There are many categorical variables, which can be processed by dividing the airports based on their location, using the "airport_list.csv" dataset.

Here are a few sample lines from the "airport_list.csv" file:



A1 DEPARTING_AIRPORT

	A	B	C	D
1	DEPARTING_AIRPORT	state		
2	Adams Field	AR		
3	Albany International	NY		
4	Albuquerque International Sunport	NM		
5	Anchorage International	AK		
6	Atlanta Municipal	GA		
7	Austin - Bergstrom International	TX		
8	Birmingham Airport	AL		
9	Bradley International	CT		
10	Charleston International	SC		
11	Chicago Midway International	IL		
12	Chicago O'Hare International	IL		
13	Cincinnati/Northern Kentucky International	OH		
14	Cleveland-Hopkins International	OH		
15	Dallas Fort Worth Regional	TX		
16	Dallas Love Field	TX		
17	Des Moines Municipal	IA		
18	Detroit Metro Wayne County	MI		
19	Douglas Municipal	NC		
20	El Paso International	TX		
21	Eppley Airfield	NE		
22	Fort Lauderdale-Hollywood International	FL		
23	Fresno Air Terminal	CA		

airports_list +

Perform classification of the departing airports based on the state, using the pandas library to group the dataset and remove the DEPARTING_AIRPORT column from the dataset after creating the "DEPARTING_STATE" column, which will reduce the number of attributes down to 43 unique values.

```python
airports_df = pd.read_csv('airports_list.csv')
# Merge the DataFrames to add the state information to the departing airport
df = df.merge(airports_df[['DEPARTING_AIRPORT', 'state']], left_on='DEPARTING_AIRPORT', right_on='DEPARTING_AIRPORT', how='left')
# Rename the state column to DEPARTING_STATE for clarity
df.rename(columns={'state': 'DEPARTING_STATE'}, inplace=True)
```

Python

```python
df = df.drop(['DEPARTING_AIRPORT'], axis=1)
```

Python

```python
df.head(5)
```

Python

IRPORT_FLIGHTS_MONTH	...	PLANE_AGE	DEPARTING_AIRPORT	LONGITUDE	PREVIOUS_AIRPORT	PRCP	SNOW	SNWD	TMAX	AWND	DEPARTING_STATE
5873	...	8	McCarran International	-115.152	NONE	0.0	0.0	0.0	65.0	2.91	NV
1174	...	3	McCarran International	-115.152	NONE	0.0	0.0	0.0	65.0	2.91	NV
1174	...	18	McCarran International	-115.152	NONE	0.0	0.0	0.0	65.0	2.91	NV
1174	...	2	McCarran International	-115.152	NONE	0.0	0.0	0.0	65.0	2.91	NV
1257	...	1	McCarran International	-115.152	NONE	0.0	0.0	0.0	65.0	2.91	NV

2.4.4: SEGMENT_NUMBER

SEGMENT_NUMBER represents the number of seats on an aircraft, with various integer variables. Based on the actual classification of aircraft, it is known that:

- ❖ Small aircraft have 1-7 seats
  - Medium aircraft have 8-100 seats
  - Large aircraft have 100-500 seats

Process the aircraft classification based on the number of seats as mentioned above:

```
bins = [0, 7, 100, 500]
labels = ['0', '1', '2']
# small (0): 0 <- 7
# medium(1): 7 <- 100
# large(2): 100 <- 500
df['Capacity'] = pd.cut(df['NUMBER_OF_SEATS'], bins=bins, labels=labels)
df.head()
```

Python

	MONTH	DAY_OF_WEEK	DEP_DEL15	DEP_TIME_BLK	DISTANCE_GROUP	SEGMENT_NUMBER	CONCURRENT_FLIGHTS	NUMBER_OF_SEATS	CARRIER_NAME
0	1	7	0	0800-0859	2	1	25	143	Southwest Airlines Co.
1	1	7	0	0700-0759	7	1	29	191	Delta Air Lines Inc.
2	1	7	0	0600-0659	7	1	27	199	Delta Air Lines Inc.
3	1	7	0	0600-0659	9	1	27	180	Delta Air Lines Inc.
4	1	7	0	0001-0559	7	1	10	182	Spirit Air Lines

5 rows × 27 columns

Use bins to divide the value ranges for each classification in a new column called "Capacity" to represent the aircraft's seating capacity.

```
df['Capacity'] = df['Capacity'].astype(int)
df.drop(columns=['NUMBER_OF_SEATS'], inplace=True)
df.head()
```

Python

MONTH	AIRLINE_FLIGHTS_MONTH	...	DEPARTING_AIRPORT	LATITUDE	LONGITUDE	PREVIOUS_AIRPORT	PRCP	SNOW	SNWD	TMAX	AWND	Capacity
13056	107363	...	McCarran International	36.09375	-115.125	NONE	0.0	0.0	0.0	65.0	2.910156	2
13056	73508	...	McCarran International	36.09375	-115.125	NONE	0.0	0.0	0.0	65.0	2.910156	2
13056	73508	...	McCarran International	36.09375	-115.125	NONE	0.0	0.0	0.0	65.0	2.910156	2
13056	73508	...	McCarran International	36.09375	-115.125	NONE	0.0	0.0	0.0	65.0	2.910156	2
13056	15023	...	McCarran International	36.09375	-115.125	NONE	0.0	0.0	0.0	65.0	2.910156	2

Drop the SEGMENT_NUMBER column after creating the new Capacity column.

2.4.5: Encoding categorical variables

The result of one-hot encoding will be a new DataFrame with the columns encoded using one-hot encoding. The new DataFrame will have the same number of rows as the original DataFrame, but the number of columns will increase. Each initial categorical feature will be replaced by a set of new columns, with each column representing the presence of a specific categorical value.

```
#One-hot encoding
variables = ['DEP_TIME_BLK','CARRIER_NAME','DEPARTING_STATE']
df = pd.get_dummies(df, columns=variables, drop_first = True)
print(df.head())
```

```
   MONTH  DEP_DEL15  DISTANCE_GROUP  SEGMENT_NUMBER  CONCURRENT_FLIGHTS  \
0      1          0               2               1                  25
1      1          0               7               1                  29
2      1          0               7               1                  27
3      1          0               9               1                  27
4      1          0               7               1                  10

   AIRPORT_FLIGHTS_MONTH  AIRLINE_AIRPORT_FLIGHTS_MONTH  GROUND_SERV_PER_PASS  \
0                  13056                           5873              0.000099
1                  13056                           1174              0.000149
2                  13056                           1174              0.000149
3                  13056                           1174              0.000149
4                  13056                           1257              0.000125

   PLANE_AGE  LONGITUDE  ... DEPARTING_STATE_ PA  DEPARTING_STATE_ PR  \
0          8    -115.125  ...               False                False
1          3    -115.125  ...               False                False
2         18    -115.125  ...               False                False
3          2    -115.125  ...               False                False
4          1    -115.125  ...               False                False

   DEPARTING_STATE_ RI  DEPARTING_STATE_ SC  DEPARTING_STATE_ TN  \
0                False                False                False
1                False                False                False
2                False                False                False
...
3                False                False
4                False                False

[5 rows x 92 columns]
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

After performing one-hot encoding on the 3 attributes [DEP_TIME_BLK, CARRIER_NAME, DEPARTING_STATE], the dataset has increased to 92 columns.

To handle the conversion of the `PREVIOUS_AIRPORT` column using label encoding.

```
#Label encoding
le = LabelEncoder()
df['PREVIOUS_AIRPORT'] = le.fit_transform(df['PREVIOUS_AIRPORT'])
```

Python

```
df.head()
```

Python

PARTING_STATE_ RI	DEPARTING_STATE_ SC	DEPARTING_STATE_ TN	DEPARTING_STATE_ TX	DEPARTING_STATE_ UT	DEPARTING_STATE_ VA	DEPARTING_STATE_ WA	DEPARTING_STATE_ WI
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False
False	False	False	False	False	False	False	False

Chapter 3 ALGORITHMS AND EXPERIMENTS

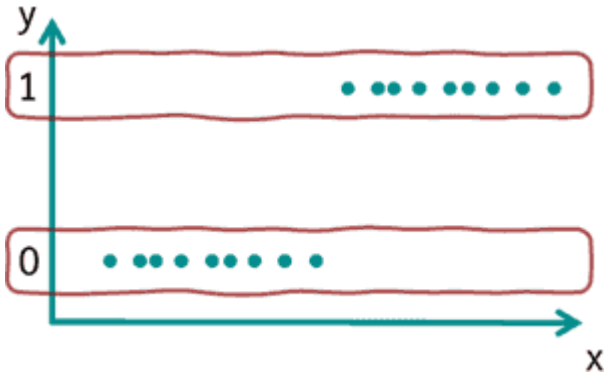
3.1: Definition Algorithms

3.1.1: Linear Regression

Logistic regression, also known as a logit model, is a statistical analysis method to predict a binary outcome, such as yes or no, based on prior observations of a data set.

A logistic regression model predicts a dependent data variable by analyzing the relationship between one or more existing independent variables. For example, logistic regression could be used to predict whether a political candidate will win or lose an election or whether a high school student will be admitted to a particular college. These binary outcomes enable straightforward decisions between two alternatives.

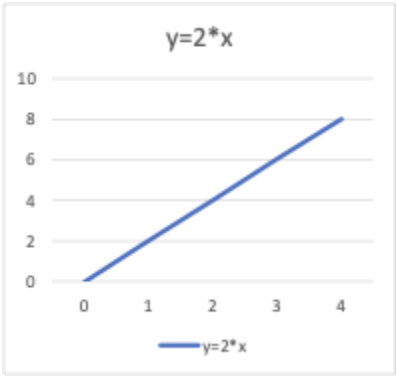
A logistic regression model can take into consideration multiple input criteria. In the case of college acceptance, the logistic function could consider factors such as the student's grade point average, SAT score and number of extracurricular activities. Based on historical data about earlier outcomes involving the same input criteria, it then scores new cases on their probability of falling into one of two outcome categories.



3.1.1.1: How does the logistic regression model work?

- Equations

In mathematics, equations give the relationship between two variables: x and y. You can use these equations, or functions, to plot a graph along the x-axis and y-axis by putting in different values of x and y. For instance, if you plot the graph for the function y = 2*x, you will get a straight line as shown below. Hence this function is also called a linear function.



❖ Variables

In statistics, variables are the data factors or attributes whose values vary. For any analysis, certain variables are independent or explanatory variables. These attributes are the cause of an outcome. Other variables are dependent or response variables; their values depend on the independent variables. In general, logistic regression explores how independent variables affect one dependent variable by looking at historical data values of both variables.

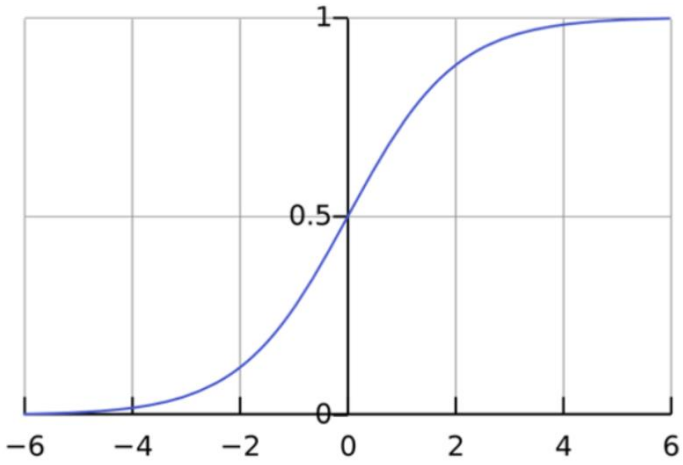
In our example above, x is called the independent variable, predictor variable, or explanatory variable because it has a known value. Y is called the dependent variable, outcome variable, or response variable because its value is unknown.

❖ Logistic regression function

Logistic regression is a statistical model that uses the logistic function, or logit function, in mathematics as the equation between x and y. The logit function maps y as a sigmoid function of x.

$$f(x) = \frac{1}{1 + e^{-x}}$$

If you plot this logistic regression equation, you will get an S-curve as shown below. A graph of the logistic function on the t-interval (−6,6).



As you can see, the logit function returns only values between 0 and 1 for the dependent variable, irrespective of the values of the independent variable. This is how logistic regression estimates the value of the dependent variable. Logistic regression methods also model equations between multiple independent variables and one dependent variable.

3.1.2: Random Forest

Random forests create decision trees on randomly selected data samples, make predictions from each tree, and choose the best solution by voting. It also provides a fairly good indicator of feature importance. Random forests have many applications, such as recommendation tools, image classification, and feature selection.

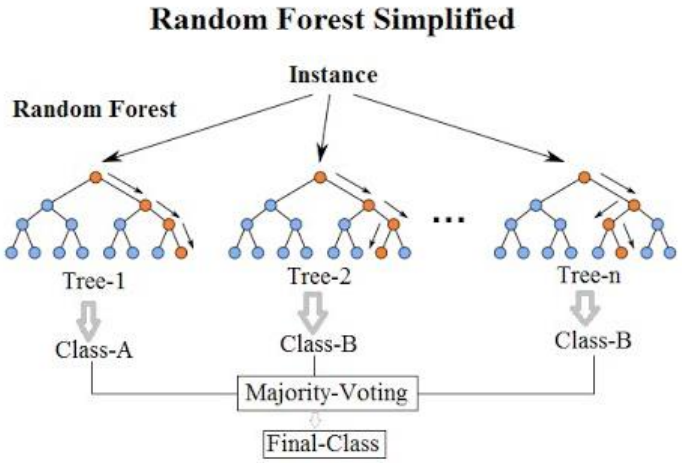
Random Forest is an ensemble model. The Random Forest model is very effective for classification tasks because it mobilizes hundreds of smaller models inside with different rules to make the final decision. Each sub-model may be stronger or weaker, but according to the "wisdom of the crowd" principle, we will have a better chance of classifying accurately than using any single model.

As the name implies, Random Forest (RF) is based on:

- **Random** = Randomness;
- **Forest** = Many decision trees.

The unit of RF is the decision tree algorithm, with hundreds of them. Each decision tree is created randomly by: Resampling (bootstrap, random sampling) and using only a small subset of random features from the entire set of variables in the data.

https://rpubs.com/lengockhanhi/343200



Random Forest operates in four steps:

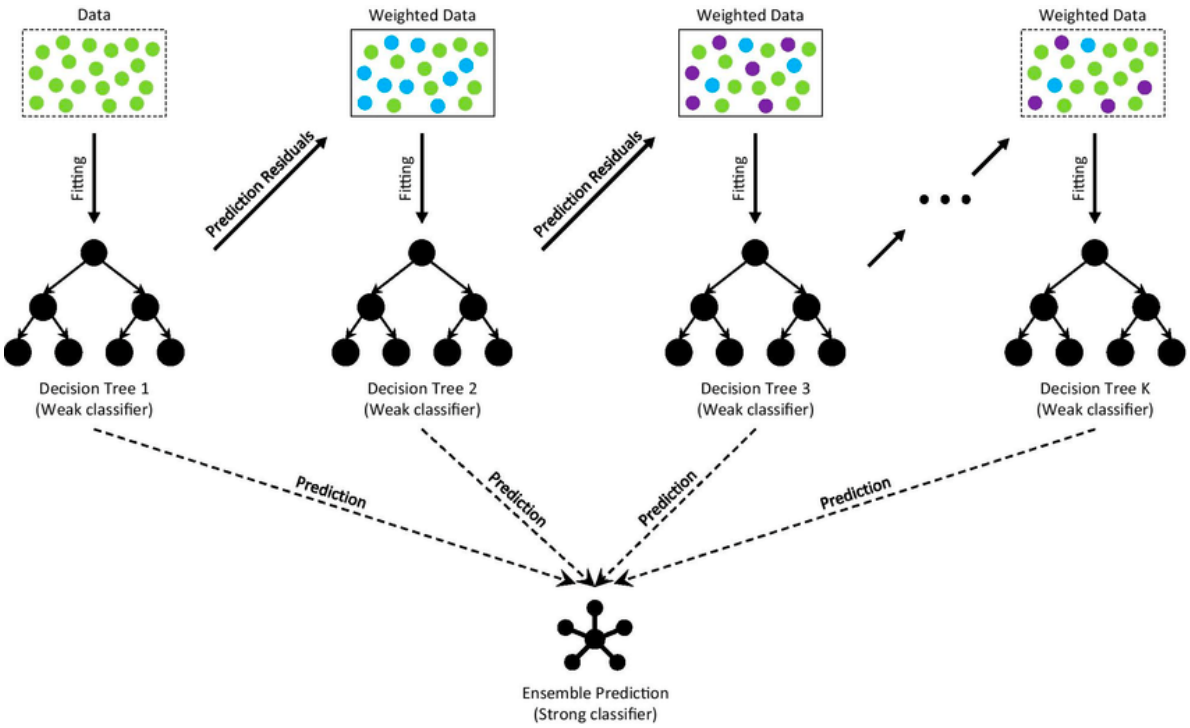
- Select random samples from the given dataset.
- Set up a decision tree for each sample and receive the predicted results from each decision tree.
- Vote for each predicted result.
- Choose the most predicted result as the final prediction.

3.1.3: Gradient Boosting Trees (GBT)

Gradient-boosted trees (GBT) are a type of machine learning algorithm used for both regression and classification problems. They form an integral part of the Gradient Boosting method, which is an ensemble technique.

In Gradient Boosting, trees are built one at a time. Each new tree is created to address the errors or shortcomings of the existing ensemble of trees. The trees used in GBT are typically decision trees, which are drawn by splitting the input space (features) in a way that minimizes a predefined loss function.

After each tree is added to the model, the algorithm calculates the loss function, which quantifies how far off the current ensemble's predictions are from the actual values. Residuals, which are the differences between the predicted and actual values, are then used to build the next tree. The next tree aims to predict these residuals.



Each tree in GBT is a decision tree that splits the data based on feature thresholds to minimize the loss function. Essentially, it chooses splits to best correct the previous trees' errors. The learning rate is crucial as it scales each new tree's contribution. A smaller learning rate means each tree's impact is more conservative, requiring more trees but often resulting in a more robust model. In the following section, we will explore the gradient-boosting algorithm in detail and use gradient-boosting trees to enhance our understanding.

Why are boosting algorithms important?

Boosting algorithms enhance prediction accuracy and model performance by converting weak learners into a strong learner. Machine learning models can be either weak or strong learners:

- Weak learners have low prediction accuracy, often comparable to random guessing. These models are prone to overfitting, unable to classify data significantly different

from the training set. For instance, a model trained to identify cats by sharp ears might fail to recognize cats with folded ears.

- **Strong learners** have higher prediction accuracy. Boosting transforms a system of weak learners into a strong learner. For example, in identifying cat images, one weak learner predicts sharp ears while another predicts cat eyes. After detecting sharp ears, the system reanalyzes for cat eyes, thus improving overall accuracy.

3.1.4: XGBoots (eXtreme Gradient Boosting)

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible, and portable. It implements machine learning algorithms under the Gradient Boosting framework.

Some of its key features include:

- Speed and Performance: XGBoost is fast and known for its performance. It uses a more regularized model formalization to control overfitting, which gives it better performance.
- Scalability and Flexibility: XGBoost scales to billions of examples and works well on a range of computing environments, including distributed systems.
- Customizable: It supports custom objective functions and evaluation criteria, adding to its flexibility.

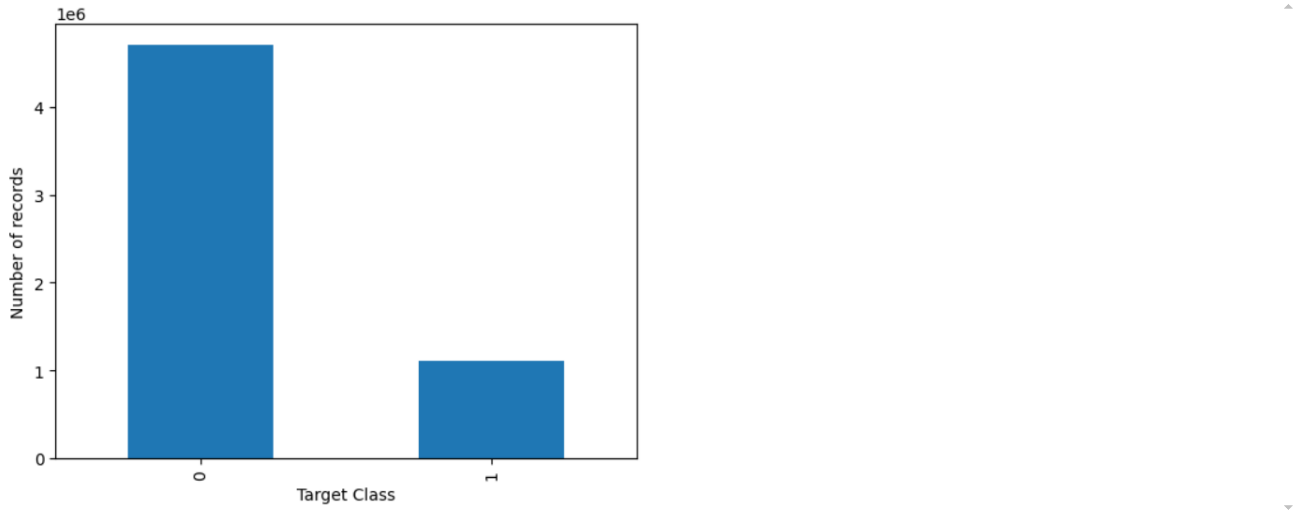
3.2: Experiments on Jupyter Notebook

3.2.1: Separating train and test data

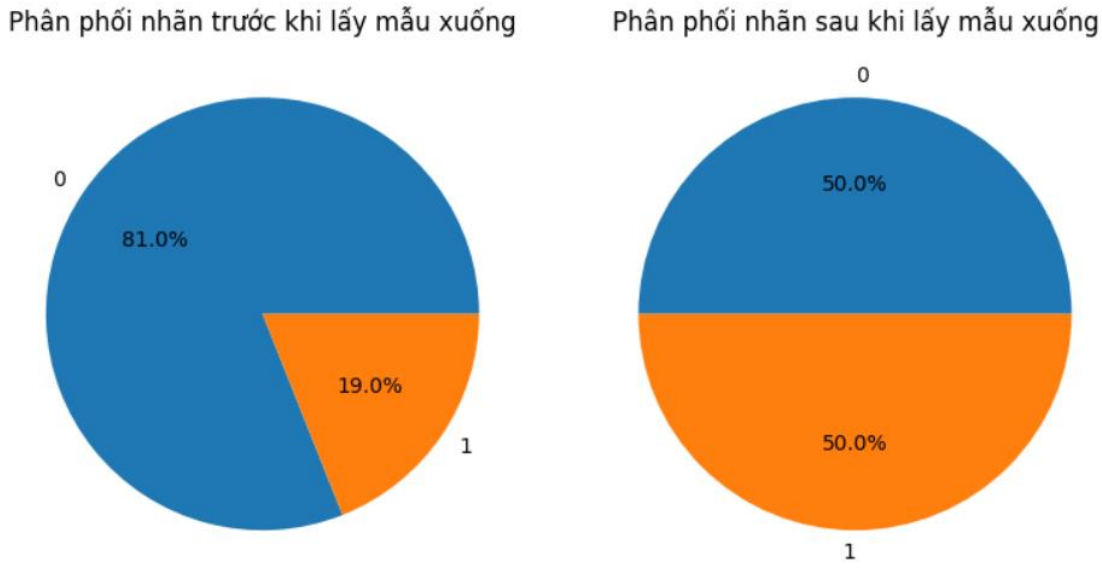
Split the dataset into training and testing sets with a 90:10 ratio

```
X_train, X_test, y_train, y_test = train_test_split(df.drop('DEP_DEL15', axis=1), df['DEP_DEL15'], test_size=0.1, random_state=42)
count = y_train.value_counts()
count.plot.bar()
plt.ylabel('Number of records')
plt.xlabel('Target Class')
plt.show()
```

✓ 18.0s Python



Because the dataset is imbalanced between the target variables, after splitting the data into training and testing sets, handle the data imbalance by applying under sampling to the training set. Here is the revised version of the code with under sampling:



After performing under sampling, the training data is sampled down to have an equal ratio of delay and on-time instances at 50:50.

```
num_samples_0_before = sum(y_train == 0)
num_samples_1_before = sum(y_train == 1)
num_samples_0_after = sum(y_train_under == 0)
num_samples_1_after = sum(y_train_under == 1)

print("Số lượng mẫu nhãn 0 trước khi lấy mẫu xuống:", num_samples_0_before)
print("Số lượng mẫu nhãn 1 trước khi lấy mẫu xuống:", num_samples_1_before)
print("Số lượng mẫu nhãn 0 sau khi lấy mẫu xuống:", num_samples_0_after)
print("Số lượng mẫu nhãn 1 sau khi lấy mẫu xuống:", num_samples_1_after)
```

✓ 1.4s

```
Số lượng mẫu nhãn 0 trước khi lấy mẫu xuống: 4710961
Số lượng mẫu nhãn 1 trước khi lấy mẫu xuống: 1103569
Số lượng mẫu nhãn 0 sau khi lấy mẫu xuống: 1103569
Số lượng mẫu nhãn 1 sau khi lấy mẫu xuống: 1103569
```

3.2.2: Data scaler

MinMaxScaler: A data scaler that transforms features to a specified range (commonly [0, 1].

```
scaler = MinMaxScaler()
X_train_under = scaler.fit_transform(X_train_under)
X_test = scaler.transform(X_test)
```

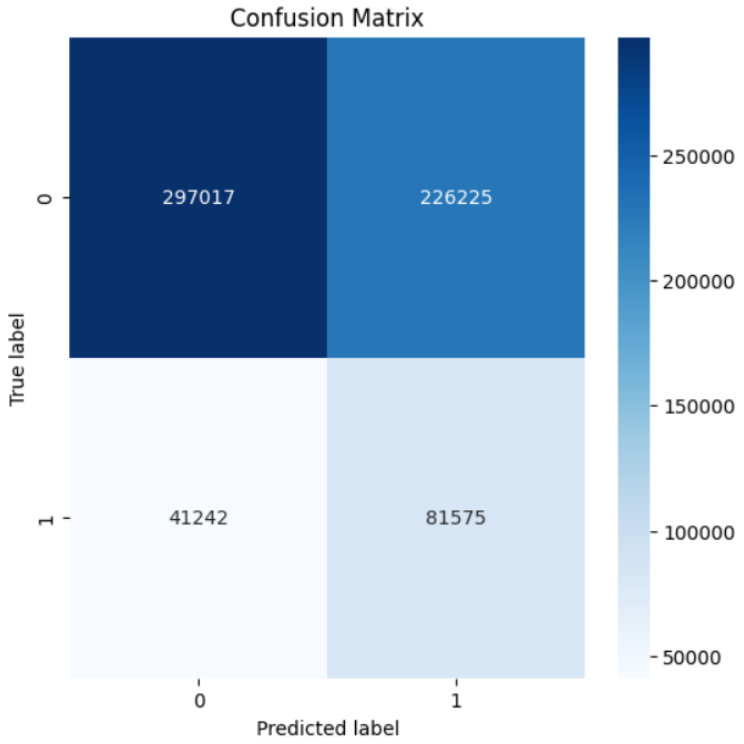
✓ 50.4s

3.2.3: Logistic Regression Algorithm

```
lr = LogisticRegression()
lr.fit(X_train_under, y_train_under)

y_pred_lr = lr.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_lr)
precision = precision_score(y_test, y_pred_lr)
recall = recall_score(y_test, y_pred_lr)
f1 = f1_score(y_test, y_pred_lr)

print(f'Linear Regression:')
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-score: {f1:.2f}')
cm = confusion_matrix(y_test, y_pred_lr)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

✓ 1m 24.9s

```
Linear Regression:
Accuracy: 0.59
Precision: 0.27
Recall: 0.66
F1-score: 0.38
```



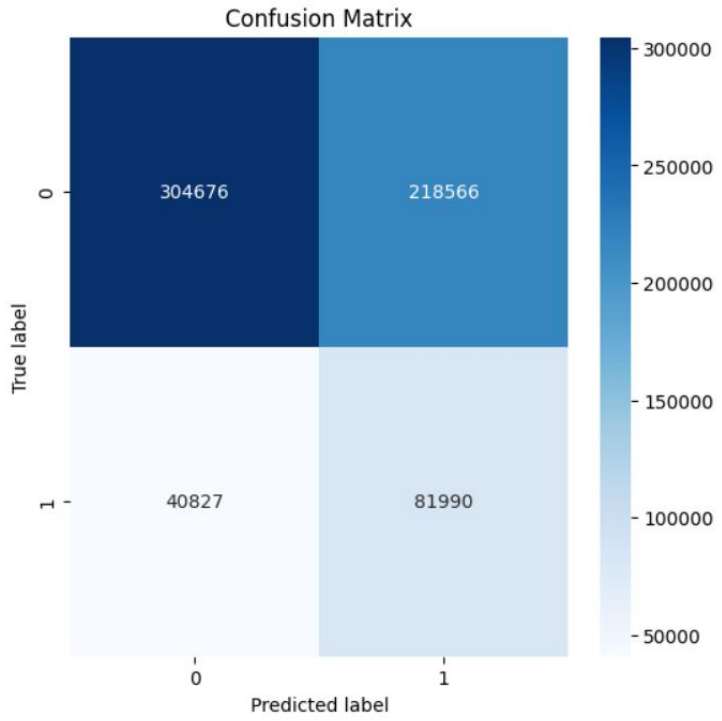
The algorithm runs within 1m24.9s, achieving an accuracy of 59%.

3.2.4: Random Forest Algorithm

```
rf = RandomForestClassifier(n_estimators=200, max_depth=10, min_samples_split=5, min_samples_leaf=2, random_state=42)
rf.fit(X_train_under, y_train_under)

y_pred_rf = rf.predict(X_test)

# Tính toán độ đo
accuracy_rf = accuracy_score(y_test, y_pred_rf)
precision_rf = precision_score(y_test, y_pred_rf)
recall_rf = recall_score(y_test, y_pred_rf)
f1_rf = f1_score(y_test, y_pred_rf)

print(f'Random Forest Classifier:')
print(f'Accuracy: {accuracy_rf:.2f}')
print(f'Precision: {precision_rf:.2f}')
print(f'Recall: {recall_rf:.2f}')
print(f'F1-score: {f1_rf:.2f}')

cm = confusion_matrix(y_test, y_pred_rf)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

✓ 22m 40.9s

```
Random Forest Classifier:
Accuracy: 0.60
Precision: 0.27
Recall: 0.67
F1-score: 0.39
```



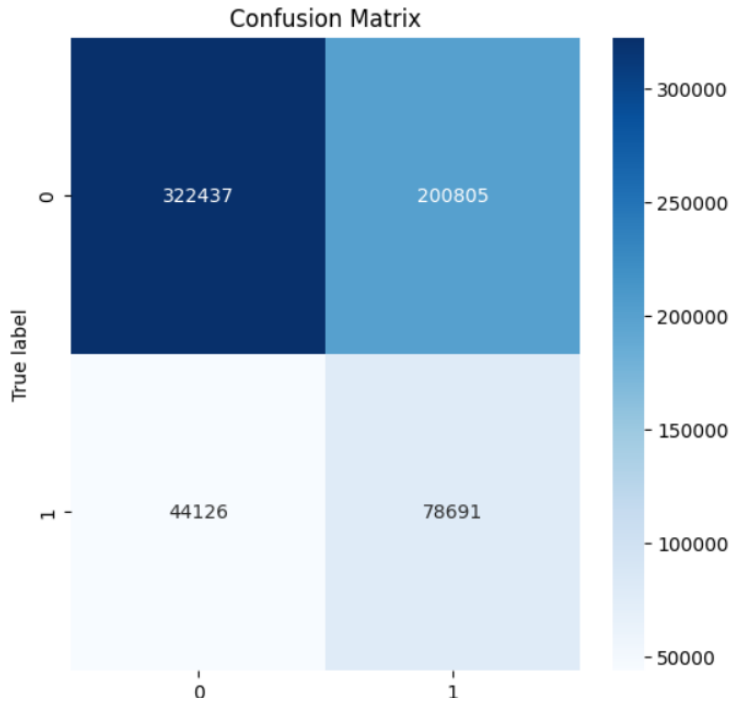
The algorithm runs within 22m40s, achieving an accuracy of 60%.

3.2.5: Gradient Boosting Trees (GBT)

```
#Gradient Boosting Trees
from sklearn.ensemble import GradientBoostingClassifier

gbt = GradientBoostingClassifier()
gbt.fit(X_train_under, y_train_under)
y_pred_gbt = gbt.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_gbt)
precision = precision_score(y_test, y_pred_gbt)
recall = recall_score(y_test, y_pred_gbt)
f1 = f1_score(y_test, y_pred_gbt)

print("\nGradient Boosting Trees Classifier:")
print(f'Accuracy: {accuracy:.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-score: {f1:.2f}')

cm = confusion_matrix(y_test, y_pred_gbt)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

✓ 37m 52.7s

```
Gradient Boosting Trees Classifier:
Accuracy: 0.62
Precision: 0.28
Recall: 0.64
F1-score: 0.39
```



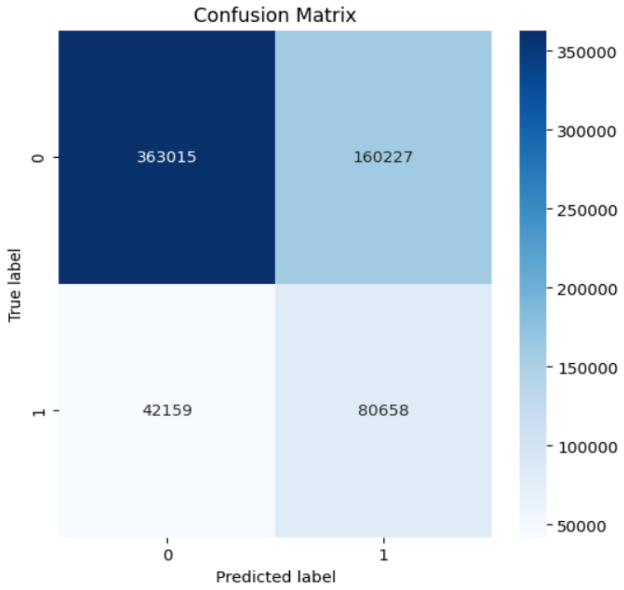
The algorithm runs within 37m52s, achieving an accuracy of 62%.

3.2.6: XGBoost

```
xgb = XGBClassifier(
    n_estimators=200,
    max_depth=10,
    learning_rate=0.3,
    random_state=42
)
xgb.fit(X_train_under, y_train_under)
y_pred_xgb = xgb.predict(X_test)

accuracy = accuracy_score(y_test, y_pred_xgb)
precision = precision_score(y_test, y_pred_xgb)
recall = recall_score(y_test, y_pred_xgb)
f1 = f1_score(y_test, y_pred_xgb)

print(f'XGBoost:')
print(f'Accuracy: {accuracy :.2f}')
print(f'Precision: {precision:.2f}')
print(f'Recall: {recall:.2f}')
print(f'F1-score: {f1:.2f}')

cm = confusion_matrix(y_test, y_pred_xgb)

plt.figure(figsize=(6, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.show()
```

✓ 2m 57.2s

```
XGBoost:
Accuracy: 0.69
Precision: 0.33
Recall: 0.66
F1-score: 0.44
```



The algorithm runs within 2m57s, achieving an accuracy of 69%.

3.2.7: Overview

The algorithm	Accuracy (%)
Logistic Regression Algorithm	59
Random Forest Algorithm	60
Gradient Boosting Trees (GBT)	62
XGBoost Algorithm	69

From the statistical table, it is observed that all 4 algorithms achieved an accuracy range of 59-69%. Comparative analysis shows that the **XGBoost algorithm** is the most effective, achieving a **69%** prediction accuracy.

Chapter 4 FLIGHT DELAY PREDICTION WEBSITE

4.1: Algorithms used

Based on the results obtained from the algorithms, XGBoost was found to be the most effective algorithm with relatively fast prediction time. Therefore, XGBoost is the chosen algorithm for building the Flight delay prediction website.

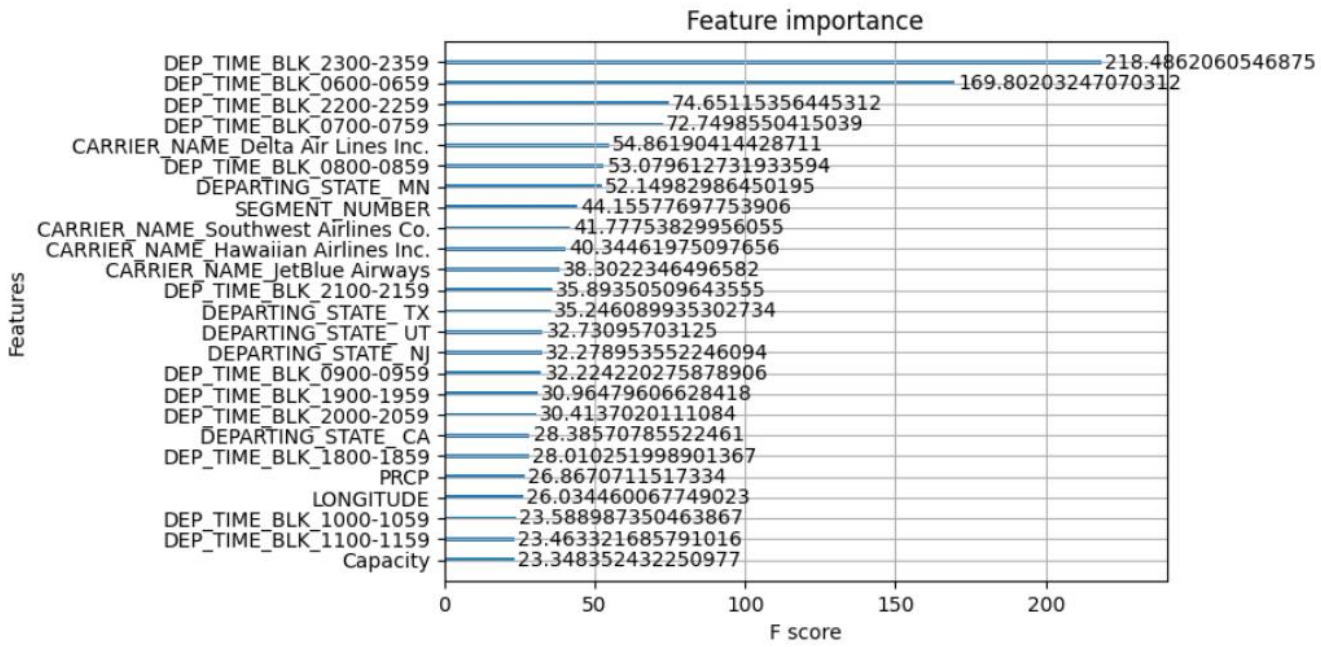
4.2: Properties used to make predictions

Utilizing algorithms to identify the most influential attributes affecting flight delays as input variables.

important feature

```
xgb = XGBClassifier(
    n_estimators=200,
    max_depth=10,
    learning_rate=0.3,
    random_state=42
)
xgb.fit(X_train, y_train)

feature_importances = xgb.feature_importances_
features = X_train.columns

importance_df = pd.DataFrame({
    'Feature': features,
    'Importance': feature_importances
})

importance_df = importance_df.sort_values(by='Importance', ascending=False)
print(importance_df)
# Vẽ biểu đồ
plt.figure(figsize=(16, 6))
plot_importance(xgb, max_num_features=20, importance_type='gain')
plt.show()
```

Results of running the feature importance algorithm

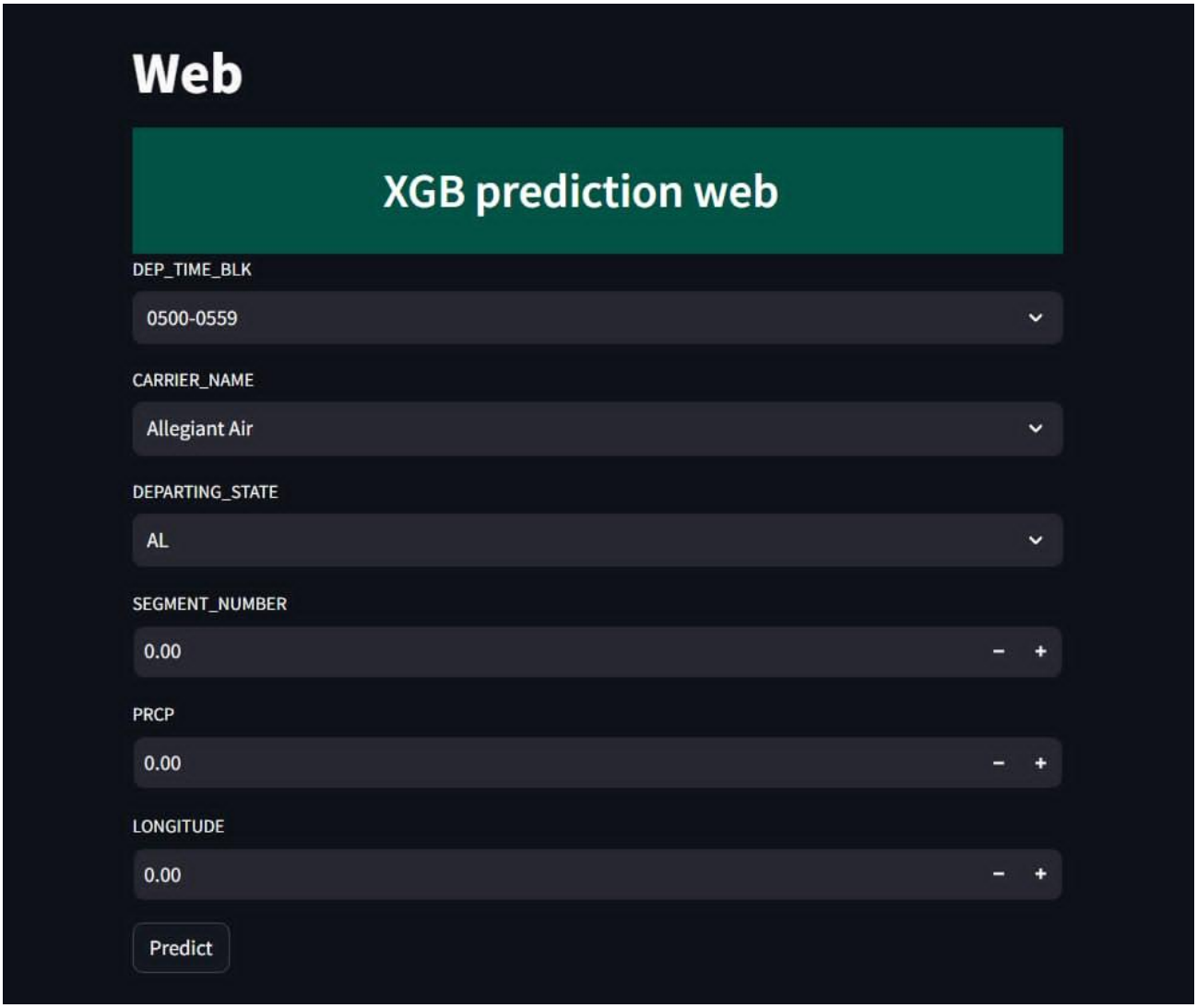


Based on the results of the feature selection test, the 6 attributes with the most significant impact on flight delays for the website prediction are: ['DEP_TIME_BLK', 'CARRIER_NAME', 'DEPARTING_STATE', 'SEGMENT_NUMBER', 'PRCP', 'LONGITUDE'].

4.3: Interface and Testing

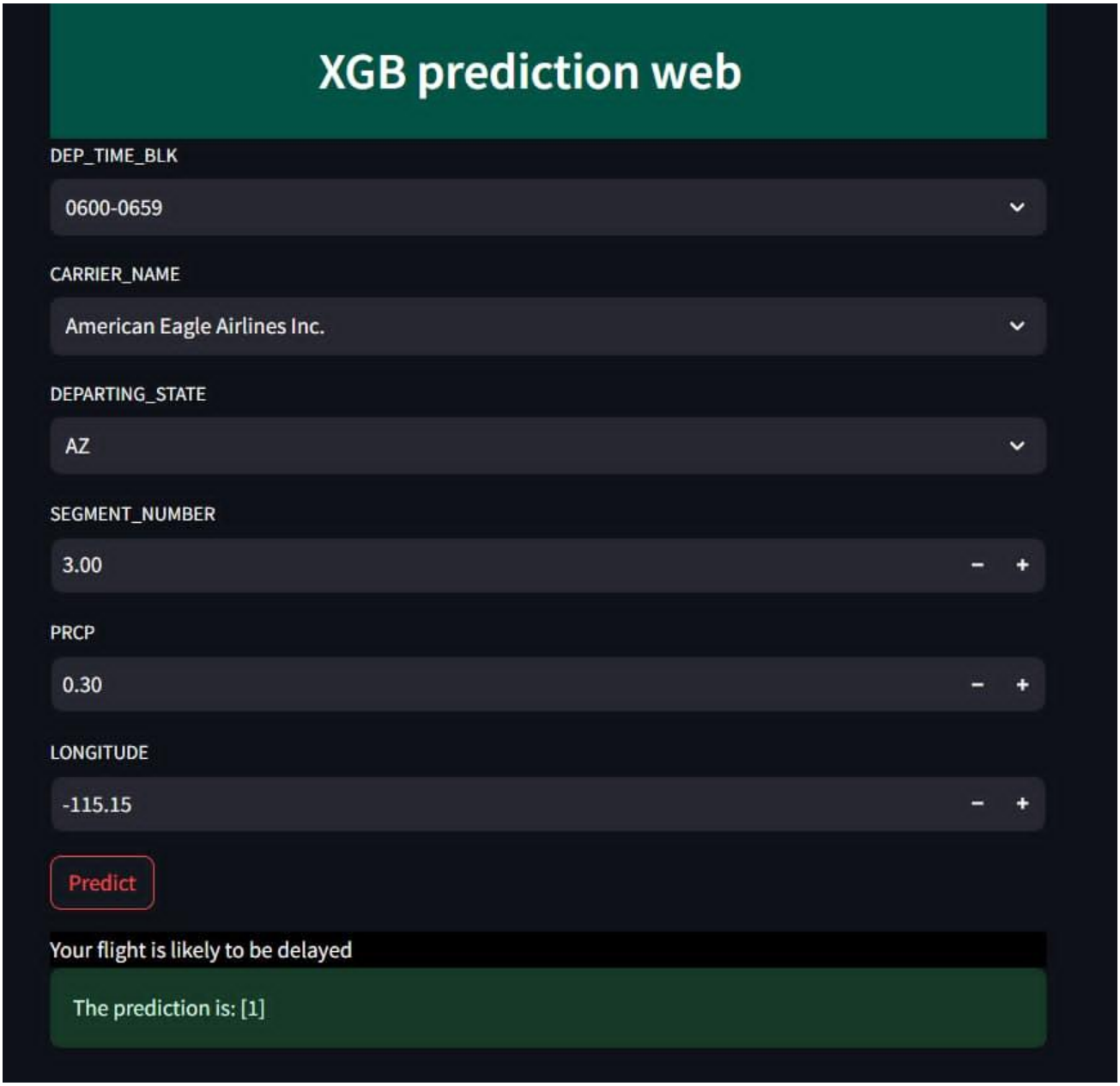
4.3.1: Interface

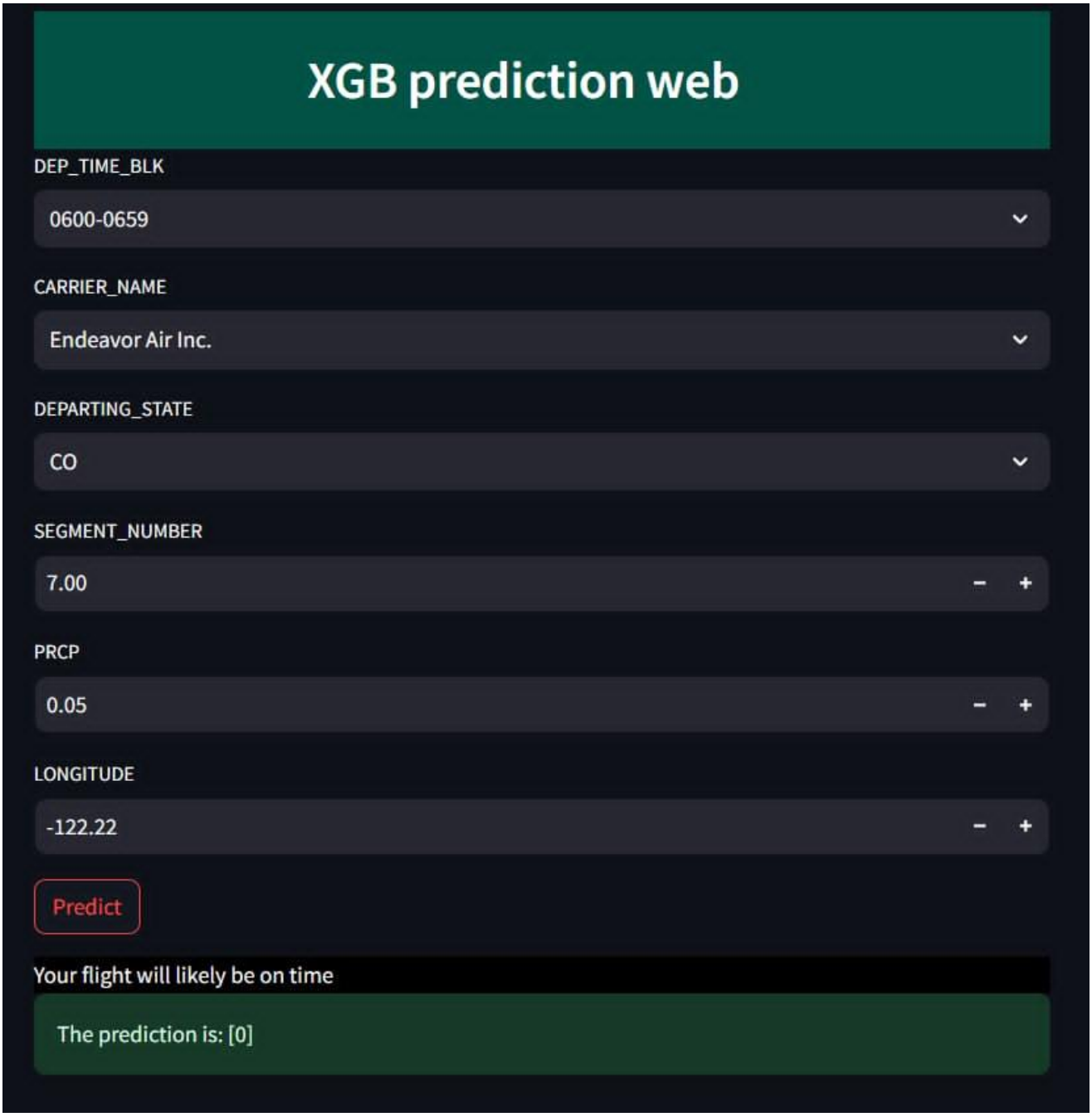
The user interface screen allows the user to input relevant information and provide a forecast on whether the flight will be delayed or ontime.



4.3.2: Testing

Perform a website functionality check.


XGB prediction web
DEP_TIME_BLK
0600-0659
CARRIER_NAME
American Eagle Airlines Inc.
DEPARTING_STATE
AZ
SEGMENT_NUMBER
3.00
PRCP
0.30
LONGITUDE
-115.15
Predict
Your flight is likely to be delayed
The prediction is: [1]



4.4: Source code

```
import streamlit as st
import pickle
import numpy as np
import pandas as pd
model=pickle.load(open('modelNew.pkl','rb'))

def main():
```

```
    st.title("Web")
    html_temp = """
    <div style="background-color:#025246 ;padding:10px">
    <h2 style="color:white;text-align:center;">XGB prediction web</h2>
    </div>
    """
    st.markdown(html_temp, unsafe_allow_html=True)
    DEP_TIME_BLK = st.selectbox("DEP_TIME_BLK", ['0500-0559' ,'0600-0659', '0700-
0759', '0800-0859', '0900-0959', '1000-1059', '1100-1159', '1200-1259', '1300-1359',
'1400-1459', '1500-1559', '1600-1659', '1700-1759', '1800-1859', '1900-1959','2000-
2059', '2100-2159', '2200-2259', '2300-2359'])
    CARRIER_NAME = st.selectbox("CARRIER_NAME", [
                                                 'Allegiant Air',
                                                 'American Airlines Inc.',
                                                 'American Eagle Airlines Inc.',
                                                 'Atlantic Southeast Airlines',
                                                 'Comair Inc.',
                                                 'Delta Air Lines Inc.',
                                                 'Endeavor Air Inc.',
                                                 'Frontier Airlines Inc.',
                                                 'Hawaiian Airlines Inc.',
                                                 'JetBlue Airways',
                                                 'Mesa Airlines Inc.',
                                                 'Midwest Airline, Inc.',
                                                 'SkyWest Airlines Inc.',
                                                 'Southwest Airlines Co.',
                                                 'Spirit Air Lines',
                                                 'United Air Lines Inc.'])

    DEPARTING_STATE = st.selectbox("DEPARTING_STATE", ['AL',
                                                       'AR',
                                                       'AZ',
                                                       'CA',
                                                       'CO',
                                                       'CT',
                                                       'DC',
                                                       'FL',
                                                       'GA',
                                                       'HI',
                                                       'IA',
                                                       'ID',
                                                       'IL',
                                                       'IN',
                                                       'KY',
```

```
                                                    'LA',
                                                    'MA',
                                                    'MD',
                                                    'ME',
                                                    'MI',
                                                    'MN',
                                                    'MO',
                                                    'NC',
                                                    'NE',
                                                    'NJ',
                                                    'NM',
                                                    'NV',
                                                    'NY',
                                                    'OH',
                                                    'OK',
                                                    'OR',
                                                    'PA',
                                                    'PR',
                                                    'RI',
                                                    'SC',
                                                    'TN',
                                                    'TX',
                                                    'UT',
                                                    'VA',
                                                    'WA',
                                                    'WI'])
    SEGMENT_NUMBER = st.number_input("SEGMENT_NUMBER", placeholder="Type a
number...")
    PRCP     = st.number_input("PRCP", placeholder="Type a number...")
    LONGITUDE     = st.number_input("LONGITUDE", placeholder="Type a number...")

    safe_html="""
      <div style="background-color:black;padding:10px >
       <h2 style="color:white;text-align:center;"> Your flight is likely to be
delayed</h2>
       </div>
    """
    danger_html="""
      <div style="background-color:black;padding:10px >
       <h2 style="color:white ;text-align:center;"> Your flight will likely be on
time </h2>
       </div>
    """
```

```
    # When the user clicks the submit button
    if st.button("Predict"):
        # Create a DataFrame from the input data
        data = {
            "DEP_TIME_BLK": [DEP_TIME_BLK],
            "CARRIER_NAME": [CARRIER_NAME],
            "DEPARTING_STATE": [DEPARTING_STATE],
            "SEGMENT_NUMBER": [SEGMENT_NUMBER],
            "PRCP": [PRCP],
            "LONGITUDE": [LONGITUDE],
        }
        df = pd.DataFrame(data)

        # Generate dummy variables for DEP_TIME_BLK
        df = pd.get_dummies(df,
columns=["DEP_TIME_BLK",'CARRIER_NAME','DEPARTING_STATE'])

        # Ensure the DataFrame has all the dummy columns the model expects
        expected_cols = ["SEGMENT_NUMBER","PRCP", "LONGITUDE",
                         "DEP_TIME_BLK_0600-0659",  "DEP_TIME_BLK_0700-0759",
"DEP_TIME_BLK_0800-0859", "DEP_TIME_BLK_0900-0959",   "DEP_TIME_BLK_1000-1059",
"DEP_TIME_BLK_1100-1159", "DEP_TIME_BLK_1200-1259", "DEP_TIME_BLK_1300-1359",
"DEP_TIME_BLK_1400-1459", "DEP_TIME_BLK_1500-1559", "DEP_TIME_BLK_1600-1659",
"DEP_TIME_BLK_1700-1759", "DEP_TIME_BLK_1800-1859", "DEP_TIME_BLK_1900-
1959","DEP_TIME_BLK_2000-2059", "DEP_TIME_BLK_2100-2159", "DEP_TIME_BLK_2200-2259",
"DEP_TIME_BLK_2300-2359",
    "CARRIER_NAME_Allegiant Air",
    "CARRIER_NAME_American Airlines Inc.",
    "CARRIER_NAME_American Eagle Airlines Inc.",
    "CARRIER_NAME_Atlantic Southeast Airlines",
    "CARRIER_NAME_Comair Inc.",
    "CARRIER_NAME_Delta Air Lines Inc.",
    "CARRIER_NAME_Endeavor Air Inc.",
    "CARRIER_NAME_Frontier Airlines Inc.",
    "CARRIER_NAME_Hawaiian Airlines Inc.",
    "CARRIER_NAME_JetBlue Airways",
    "CARRIER_NAME_Mesa Airlines Inc.",
    "CARRIER_NAME_Midwest Airline, Inc.",
    "CARRIER_NAME_SkyWest Airlines Inc.",
    "CARRIER_NAME_Southwest Airlines Co.",
    "CARRIER_NAME_Spirit Air Lines",
    "CARRIER_NAME_United Air Lines Inc.",
    "DEPARTING_STATE_AL",
    "DEPARTING_STATE_AR",
```

```
"DEPARTING_STATE_AZ",
"DEPARTING_STATE_CA",
"DEPARTING_STATE_CO",
"DEPARTING_STATE_CT",
"DEPARTING_STATE_DC",
"DEPARTING_STATE_FL",
"DEPARTING_STATE_GA",
"DEPARTING_STATE_HI",
"DEPARTING_STATE_IA",
"DEPARTING_STATE_ID",
"DEPARTING_STATE_IL",
"DEPARTING_STATE_IN",
"DEPARTING_STATE_KY",
"DEPARTING_STATE_LA",
"DEPARTING_STATE_MA",
"DEPARTING_STATE_MD",
"DEPARTING_STATE_ME",
"DEPARTING_STATE_MI",
"DEPARTING_STATE_MN",
"DEPARTING_STATE_MO",
"DEPARTING_STATE_NC",
"DEPARTING_STATE_NE",
"DEPARTING_STATE_NJ",
"DEPARTING_STATE_NM",
"DEPARTING_STATE_NV",
"DEPARTING_STATE_NY",
"DEPARTING_STATE_OH",
"DEPARTING_STATE_OK",
"DEPARTING_STATE_OR",
"DEPARTING_STATE_PA",
"DEPARTING_STATE_PR",
"DEPARTING_STATE_RI",
"DEPARTING_STATE_SC",
"DEPARTING_STATE_TN",
"DEPARTING_STATE_TX",
"DEPARTING_STATE_UT",
"DEPARTING_STATE_VA",
"DEPARTING_STATE_WA",
"DEPARTING_STATE_WI"]
    for col in expected_cols:
        if col not in df.columns:
            df[col] = 0

    # Reorder columns to match the training data
```

```
        df = df[expected_cols]


        # Convert DataFrame to NumPy array
        input_data = df.to_numpy()

        # Make prediction
        prediction = model.predict(input_data)
        if prediction == 0:
            st.markdown(danger_html,unsafe_allow_html=True)
        else:
            st.markdown(safe_html,unsafe_allow_html=True)
        st.success(f'The prediction is: {prediction}')

if __name__=='__main__':
    main()
```

Chapter 5 CONCLUSION

5.1 Advantages and limitations of each algorithm

5.1.1: Logistic Regression Algorithm

Advantages:

- **Simple and Easy to Understand:** Logistic regression is a relatively simple machine learning algorithm that is easy to understand and implement.
- **Effective:** Logistic regression can achieve high performance in classifying binary problems, especially when the data is linearly separable.
- **Can be used with nonlinear data:** Logistic regression can be used with nonlinear data by incorporating nonlinear independent variables.
- Memory-efficient and fast-executing algorithm.
- **Good performance on processed data:** Logistic regression typically produces good classification results when the data has been cleaned, and normalized, and important attributes have been selected.

Limitations:

- For large datasets with many attributes and nonlinear relationships, logistic regression may not be powerful enough to fully model the data.

- Logistic regression can be sensitive to highly correlated or irrelevant features, leading to poor performance.

5.1.2: Random Forest Algorithm

Advantages:

- **Interpretability:** Despite its complexity, Random Forest remains interpretable through measures like feature importance.
- **Robustness to Noise:** The algorithm exhibits resilience to outliers and missing values in the input data.
- **Effective Feature Handling:** Random Forest excels at handling features of varying importance, even those that are irrelevant.

Limitations:

- Inefficient Classification Performance: F1-score of only 39%
- Slow Execution Time.
- Difficulty Handling Complex Nonlinear Relationships in Data.

5.1.3: Gradient Boosting Trees Algorithm

Advantages:

- Effective Handling of Diverse Features, Both Important and Irrelevant.
- Robustness to Noisy or Missing Data.
- Interpretable Model: Explainable Through Feature Importance Metrics."

Limitations:

- Low classification accuracy and performance (Acc 62%, F1 39%)
- Challenges in handling complex nonlinear relationships in the data.
- Long execution time due to resource-intensive training process.

5.1.4: XGBoost Algorithm

Advantages:

- **Higher classification performance:** The accuracy of 69% and higher F1-score compared to Random Forest indicate that XGBoost has better data modeling capabilities.
- **Efficient execution time:** It only takes 2 minutes to train the model, much faster than the 22 minutes required for Random Forest.
- **Ability to handle complex nonlinear relationships:** can effectively handle complex nonlinear relationships in the data, outperforming Random Forest.

Limitations:

- XGBoost is a more complex ensemble model than Random Forest, making it more difficult to understand and interpret.
- The training process of XGBoost can be more resource-intensive, consuming more memory and computational resources.
- XGBoost may be more susceptible to the influence of outliers or noisy data in certain cases.

Chapter 6 REFERENCES

[1] "1.1.3. Logistic Regression — scikit-learn 0.24.2 documentation," scikit-learn, [Online]. Available: https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression. [Accessed 18 Jun 2024].

[2] "Random Forest," IBM, [Online]. Available: https://www.ibm.com/topics/random-forest. [Accessed 12 Jun 2024].

[3] M. Ibrahim, "An Introduction to Gradient Boosted Trees for Machine Learning," Weights & Biases, 2022.

[4] K. Xie, D. Yang, L. Zhang, D. Ding, B. Xu, J. Zhao, and X. Bian, "Ensemble learning for the early prediction of neonatal jaundice with genetic features," *IEEE Access*, vol. 9, no. IEEE Access, pp. 152158-152170, 2021.

[5] "Phân loại máy bay, sơ đồ máy bay," [Online]. Available: https://www.studocu.com/vn/document/truong-dai-hoc-giao-thong-van-tai/soil-mechanics/phan-loai-may-bay-so-do-may-bay/57056946. [Accessed 10 June 2024].