Team2 Vaama Shrikrishna Nikam
Samyuktha Chaparla Vinay Anjaneya Desiraju Raviteja Reddy Bayapu

In [189…
```
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression

from sklearn import tree
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import BaggingRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error

from sklearn.ensemble import StackingClassifier
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier


pd.options.display.float_format = '{:,.2f}'.format

# setup interactive notebook mode
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

from IPython.display import display, HTML
```

First Flight Model

In [190…
```
import pandas as pd

# Read the First flight details
first_df = pd.read_excel("Datasets/First_Flight_Details.xlsx")

first_df.head()
```

Out[190…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)
0	UA	2023-01-01	2645	N23721	ORD	SYR	21:10:00	21:07:00	-3	23:57:00	23:47:00	-10
1	UA	2023-01-02	1998	N802UA	ORD	SYR	18:14:00	18:17:00	3	21:07:00	20:46:00	-21
2	UA	2023-01-03	1998	N854UA	ORD	SYR	18:14:00	18:13:00	-1	21:07:00	21:04:00	-3
3	UA	2023-01-04	1998	N893UA	ORD	SYR	18:24:00	18:41:00	17	21:17:00	21:31:00	14
4	UA	2020-01-05	2012	N825UA	ORD	SYR	19:00:00	19:21:00	21	21:47:00	21:55:00	8

In [191…
```
first_df.dtypes
```

Out[191…
```
Carrier Code                      object
Date (MM/DD/YYYY)           datetime64[ns]
Flight Number                        int64
Tail Number                         object
Origin Airport                      object
Destination Airport                 object
Scheduled departure time            object
Actual departure time               object
Departure delay (Minutes)            int64
Scheduled Arrival Time              object
Actual Arrival Time                 object
Arrival Delay (Minutes)              int64
dtype: object
```

In [192…
```
first_df.isna().sum()
```

Out[192…
```
Carrier Code                 0
Date (MM/DD/YYYY)            0
Flight Number                0
Tail Number                 53
Origin Airport               0
Destination Airport          0
Scheduled departure time     0
Actual departure time        0
Departure delay (Minutes)    0
Scheduled Arrival Time       0
Actual Arrival Time          0
Arrival Delay (Minutes)      0
dtype: int64
```

In [193…
```
# Drop the Nan values
first_df.dropna(subset=['Tail Number'], inplace=True)
```

Weather data preprocessing

In [194…
```
# Read weather data for all origin airports
All_weather = pd.read_csv("Datasets/Weather_Data.csv")

All_weather.head()
```

Out[194...

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather_description	pres	slp	vis	snow	Location
0	2019-01-01:05	2019-01-01T00:00:00	9.70	9.70	100	2.50	100	5.15	11.60	160	Light rain	1008	1009	3	0.00	JFK
1	2019-01-01:06	2019-01-01T01:00:00	9.40	9.40	100	2.00	100	5.09	10.40	180	Light rain	1007	1007	0	0.00	JFK
2	2019-01-01:07	2019-01-01T02:00:00	8.90	8.90	100	1.00	100	3.22	11.60	180	Light rain	1004	1005	0	0.00	JFK
3	2019-01-01:08	2019-01-01T03:00:00	8.90	8.90	100	0.00	100	4.34	13.60	205	Fog	1002	1003	1	0.00	JFK
4	2019-01-01:09	2019-01-01T04:00:00	9.40	9.40	100	0.00	100	1.50	13.60	280	Fog	1002	1003	2	0.00	JFK

In [195...

```
# Read weather data for SYR airport
syr_weather = pd.read_csv("Datasets/SYR_Weather_Data.csv")

syr_weather.head()
```

Out[195...

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp	vis	snow
0	2019-01-01:05	2019-01-01T00:00:00	7.20	4.30	87.00	1.50	85.00	4.59	9.80	150.00	Light rain	982.00	997.00	16.00	0.00
1	2019-01-01:06	2019-01-01T01:00:00	8.90	8.90	100.00	0.50	79.00	5.09	11.80	190.00	Overcast clouds	984.00	999.00	16.00	0.00
2	2019-01-01:07	2019-01-01T02:00:00	9.40	9.40	100.00	0.00	83.00	4.59	12.90	180.00	Overcast clouds	982.00	997.00	16.00	0.00
3	2019-01-01:08	2019-01-01T03:00:00	11.10	11.10	87.00	0.00	80.00	8.19	13.90	200.00	Overcast clouds	980.00	996.00	16.00	0.00
4	2019-01-01:09	2019-01-01T04:00:00	10.60	10.60	100.00	0.50	85.00	6.20	16.80	230.00	Overcast clouds	981.00	996.00	16.00	0.00

In [196...

```
All_weather.dtypes
```

Out[196...

```
datetime                object
timestamp_local         object
temp                   float64
app_temp               float64
clouds                   int64
precip                 float64
rh                       int64
wind_spd               float64
wind_gust_spd          float64
wind_dir                 int64
weather_description     object
pres                     int64
slp                      int64
vis                      int64
snow                   float64
Location                object
dtype: object
```

In [197...

```
# Convert the timestamp in weather data to datetime
All_weather['timestamp_local'] = pd.to_datetime(All_weather['timestamp_local'])
syr_weather['timestamp_local'] = pd.to_datetime(syr_weather['timestamp_local'])
syr_weather.head()
```

Out[197...

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp	vis	snow
0	2019-01-01:05	2019-01-01 00:00:00	7.20	4.30	87.00	1.50	85.00	4.59	9.80	150.00	Light rain	982.00	997.00	16.00	0.00
1	2019-01-01:06	2019-01-01 01:00:00	8.90	8.90	100.00	0.50	79.00	5.09	11.80	190.00	Overcast clouds	984.00	999.00	16.00	0.00
2	2019-01-01:07	2019-01-01 02:00:00	9.40	9.40	100.00	0.00	83.00	4.59	12.90	180.00	Overcast clouds	982.00	997.00	16.00	0.00
3	2019-01-01:08	2019-01-01 03:00:00	11.10	11.10	87.00	0.00	80.00	8.19	13.90	200.00	Overcast clouds	980.00	996.00	16.00	0.00
4	2019-01-01:09	2019-01-01 04:00:00	10.60	10.60	100.00	0.50	85.00	6.20	16.80	230.00	Overcast clouds	981.00	996.00	16.00	0.00

In [198...

```
# Extract Date and Hour
All_weather['Date'] = All_weather['timestamp_local'].dt.date
All_weather['Hour'] = All_weather['timestamp_local'].dt.hour

syr_weather['Date'] = syr_weather['timestamp_local'].dt.date
syr_weather['Hour'] = syr_weather['timestamp_local'].dt.hour

syr_weather.head()
```

Out[198...

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp	vis	snow	Date	Hour
0	2019-01-01:05	2019-01-01 00:00:00	7.20	4.30	87.00	1.50	85.00	4.59	9.80	150.00	Light rain	982.00	997.00	16.00	0.00	2019-01-01	0
1	2019-01-01:06	2019-01-01 01:00:00	8.90	8.90	100.00	0.50	79.00	5.09	11.80	190.00	Overcast clouds	984.00	999.00	16.00	0.00	2019-01-01	1
2	2019-01-01:07	2019-01-01 02:00:00	9.40	9.40	100.00	0.00	83.00	4.59	12.90	180.00	Overcast clouds	982.00	997.00	16.00	0.00	2019-01-01	2
3	2019-01-01:08	2019-01-01 03:00:00	11.10	11.10	87.00	0.00	80.00	8.19	13.90	200.00	Overcast clouds	980.00	996.00	16.00	0.00	2019-01-01	3
4	2019-01-01:09	2019-01-01 04:00:00	10.60	10.60	100.00	0.50	85.00	6.20	16.80	230.00	Overcast clouds	981.00	996.00	16.00	0.00	2019-01-01	4

In [199...

```
# Convert Date column to Datetime type
All_weather['Date'] = pd.to_datetime(All_weather['Date'])

syr_weather['Date'] = pd.to_datetime(syr_weather['Date'])
```

In [200…
```
syr_weather.drop(['datetime', 'timestamp_local'], axis=1, inplace=True)

All_weather.drop(['datetime', 'timestamp_local'], axis=1, inplace=True)
```

In [201…
```
syr_weather.dtypes
```

Out[201…
```
temp                   float64
app_temp               float64
clouds                 float64
precip                 float64
rh                     float64
wind_spd               float64
wind_gust_spd          float64
wind_dir               float64
weather                 object
pres                   float64
slp                    float64
vis                    float64
snow                   float64
Date            datetime64[ns]
Hour                     int32
dtype: object
```

In [202…
```
len(syr_weather)
len(first_df)
```

Out[202…
```
46321
```

Out[202…
```
4488
```

Previous flights data preprocessing and feature extraction

In [203…
```
first_df.dtypes
```

Out[203…
```
Carrier Code                        object
Date (MM/DD/YYYY)           datetime64[ns]
Flight Number                        int64
Tail Number                         object
Origin Airport                      object
Destination Airport                 object
Scheduled departure time            object
Actual departure time               object
Departure delay (Minutes)            int64
Scheduled Arrival Time              object
Actual Arrival Time                 object
Arrival Delay (Minutes)              int64
dtype: object
```

In [204…
```
# Extract Day of Week from Date and add all Day names as categories
first_df['Day of Week'] = first_df['Date (MM/DD/YYYY)'].dt.dayofweek

# Map the day of the week to day names
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
first_df['Day Name'] = first_df['Day of Week'].map(lambda x: day_names[x])

first_df['Day Name'] = pd.Categorical(first_df['Day Name'], categories=day_names)

print(first_df['Day Name'].cat.categories)
```

```
Index(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
       'Sunday'],
      dtype='object')
```

In [205…
```
first_df.head()
```

Out[205…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Day of Week	Day Name
0	UA	2023-01-01	2645	N23721	ORD	SYR	21:10:00	21:07:00	-3	23:57:00	23:47:00	-10	6	Sunday
1	UA	2023-01-02	1998	N802UA	ORD	SYR	18:14:00	18:17:00	3	21:07:00	20:46:00	-21	0	Monday
2	UA	2023-01-03	1998	N854UA	ORD	SYR	18:14:00	18:13:00	-1	21:07:00	21:04:00	-3	1	Tuesday
3	UA	2023-01-04	1998	N893UA	ORD	SYR	18:24:00	18:41:00	17	21:17:00	21:31:00	14	2	Wednesday
4	UA	2020-01-05	2012	N825UA	ORD	SYR	19:00:00	19:21:00	21	21:47:00	21:55:00	8	6	Sunday

In [206…
```
columns_to_keep = ['Carrier Code', 'Date (MM/DD/YYYY)','Origin Airport', 'Scheduled departure time', 'Actual departure time','Arrival Delay (Minutes)',
                   'Scheduled Arrival Time', 'Actual Arrival Time','Day Name']

first_df = first_df[columns_to_keep]
```

In [207…
```
first_df.head()
```

Out[207…

	Carrier Code	Date (MM/DD/YYYY)	Origin Airport	Scheduled departure time	Actual departure time	Arrival Delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Day Name
0	UA	2023-01-01	ORD	21:10:00	21:07:00	-10	23:57:00	23:47:00	Sunday
1	UA	2023-01-02	ORD	18:14:00	18:17:00	-21	21:07:00	20:46:00	Monday
2	UA	2023-01-03	ORD	18:14:00	18:13:00	-3	21:07:00	21:04:00	Tuesday
3	UA	2023-01-04	ORD	18:24:00	18:41:00	14	21:17:00	21:31:00	Wednesday
4	UA	2020-01-05	ORD	19:00:00	19:21:00	8	21:47:00	21:55:00	Sunday

In [208…
```
first_df.dtypes
```

Out[208…
```
Carrier Code                       object
Date (MM/DD/YYYY)          datetime64[ns]
Origin Airport                     object
Scheduled departure time           object
Actual departure time              object
Arrival Delay (Minutes)             int64
Scheduled Arrival Time             object
Actual Arrival Time                object
Day Name                         category
dtype: object
```

In [209…
```
# Convert 'Scheduled departure time' and 'Scheduled Arrival Time' to datetime
first_df['Scheduled departure time'] = pd.to_datetime(first_df['Scheduled departure time'], format='%H:%M:%S')

first_df['Scheduled Arrival Time'] = pd.to_datetime(first_df['Scheduled Arrival Time'], format='%H:%M:%S')
```

In [210…
```
# Extract hour and minute components
first_df['Scheduled departure hour'] = first_df['Scheduled departure time'].dt.hour
first_df['Scheduled departure minute'] = first_df['Scheduled departure time'].dt.minute

first_df['Scheduled Arrival hour'] = first_df['Scheduled Arrival Time'].dt.hour
first_df['Scheduled Arrival minute'] = first_df['Scheduled Arrival Time'].dt.minute
```

In [211…
```
first_df.dtypes
```

Out[211…
```
Carrier Code                       object
Date (MM/DD/YYYY)          datetime64[ns]
Origin Airport                     object
Scheduled departure time   datetime64[ns]
Actual departure time              object
Arrival Delay (Minutes)             int64
Scheduled Arrival Time     datetime64[ns]
Actual Arrival Time                object
Day Name                         category
Scheduled departure hour            int32
Scheduled departure minute          int32
Scheduled Arrival hour              int32
Scheduled Arrival minute            int32
dtype: object
```

In [212…
```
first_df.drop(['Scheduled departure time', 'Actual departure time', 'Scheduled Arrival Time', 'Actual Arrival Time'], axis=1, inplace=True)
```

In [213…
```
first_df.head()
```

Out[213…

	Carrier Code	Date (MM/DD/YYYY)	Origin Airport	Arrival Delay (Minutes)	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute
0	UA	2023-01-01	ORD	-10	Sunday	21	10	23	57
1	UA	2023-01-02	ORD	-21	Monday	18	14	21	7
2	UA	2023-01-03	ORD	-3	Tuesday	18	14	21	7
3	UA	2023-01-04	ORD	14	Wednesday	18	24	21	17
4	UA	2020-01-05	ORD	8	Sunday	19	0	21	47

In [214…
```
# Define a function to determine the arrival status
def determine_arrival_status(delay):
    if delay > 5:
        return 2
    elif delay < -5:
        return 1
    else:
        return 0

# Apply the function to create the "Arrival Status" column
first_df['Arrival Status'] = first_df['Arrival Delay (Minutes)'].apply(determine_arrival_status)
```

In [215…
```
first_df.drop('Arrival Delay (Minutes)', axis=1, inplace=True)
```

In [216…
```
first_df.dtypes
```

Out[216…
```
Carrier Code                        object
Date (MM/DD/YYYY)           datetime64[ns]
Origin Airport                      object
Day Name                          category
Scheduled departure hour             int32
Scheduled departure minute           int32
Scheduled Arrival hour               int32
Scheduled Arrival minute             int32
Arrival Status                       int64
dtype: object
```

Merge Weather data with flight data

In [217…
```
# Merge the Destination weather data (SYR) with flight data, mapped using Date and Arrival Hour
merged_df = pd.merge(first_df, syr_weather, left_on=["Date (MM/DD/YYYY)", "Scheduled Arrival hour"], right_on=["Date", "Hour"], how="left")
```

In [218…
```
merged_df.head()
```

Out[218…

	Carrier Code	Date (MM/DD/YYYY)	Origin Airport	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	Arrival Status	temp	…	wind_spd	wind_gust_spd	wind_dir	weather	pres	
0	UA	2023-01-01	ORD	Sunday	21	10	23	57	1	3.90	…	1.60	3.60	280.00	Overcast clouds	1,000.00	1,01
1	UA	2023-01-02	ORD	Monday	18	14	21	7	1	4.40	…	1.20	2.40	55.00	Overcast clouds	1,006.00	1,02
2	UA	2023-01-03	ORD	Tuesday	18	14	21	7	0	5.00	…	1.50	4.80	70.00	Overcast clouds	991.00	1,00
3	UA	2023-01-04	ORD	Wednesday	18	24	21	17	2	7.20	…	3.10	13.20	80.00	Overcast clouds	990.00	1,00
4	UA	2020-01-05	ORD	Sunday	19	0	21	47	2	-0.60	…	2.60	3.20	210.00	Overcast clouds	998.00	1,01

5 rows × 24 columns

In [219…
```
merged_df.isna().sum()
```

Out[219…
```
Carrier Code                0
Date (MM/DD/YYYY)           0
Origin Airport              0
Day Name                    0
Scheduled departure hour    0
Scheduled departure minute  0
Scheduled Arrival hour      0
Scheduled Arrival minute    0
Arrival Status              0
temp                        0
app_temp                    0
clouds                      0
precip                      0
rh                          0
wind_spd                    0
wind_gust_spd               0
wind_dir                    0
weather                     0
pres                        0
slp                         0
vis                         0
snow                        0
Date                        0
Hour                        0
dtype: int64
```

In [220…
```
len(merged_df)
len(All_weather)
```

Out[220… 4488

Out[220… 177793

In [221…
```
# Merge the Origin weather data with flight data, mapped using Date, Departure Hour and Origin Airport
merged_df = pd.merge(merged_df, All_weather, how='left', left_on=['Date (MM/DD/YYYY)', 'Scheduled departure hour', 'Origin Airport'], right_on=['Date', 'Hour',
```

In [222…
```
merged_df.drop(['Location', 'Date_x','Hour_x','Date_y', 'Hour_y'], axis=1, inplace=True)
merged_df.head()
```

Out[222...

	Carrier Code	Date (MM/DD/YYYY)	Origin Airport	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	Arrival Status	temp_x	...	precip_y	rh_y	wind_spd_y	wind_gust_spd_y	wind_dir_y
0	UA	2023-01-01	ORD	Sunday	21	10	23	57	1	3.90	...	0.00	87	1.60	3.60	250
1	UA	2023-01-02	ORD	Monday	18	14	21	7	1	4.40	...	0.00	85	4.59	6.40	80
2	UA	2023-01-03	ORD	Tuesday	18	14	21	7	0	5.00	...	0.00	88	2.10	2.80	20
3	UA	2023-01-04	ORD	Wednesday	18	24	21	17	2	7.20	...	0.00	88	3.60	8.40	220
4	UA	2020-01-05	ORD	Sunday	19	0	21	47	2	-0.60	...	0.00	69	7.32	11.60	280

5 rows × 35 columns

In [223...
```
len(merged_df)
```

Out[223...
```
4488
```

In [224...
```
merged_df.isna().sum()
```

Out[224...
```
Carrier Code                0
Date (MM/DD/YYYY)           0
Origin Airport              0
Day Name                    0
Scheduled departure hour    0
Scheduled departure minute  0
Scheduled Arrival hour      0
Scheduled Arrival minute    0
Arrival Status              0
temp_x                      0
app_temp_x                  0
clouds_x                    0
precip_x                    0
rh_x                        0
wind_spd_x                  0
wind_gust_spd_x             0
wind_dir_x                  0
weather                     0
pres_x                      0
slp_x                       0
vis_x                       0
snow_x                      0
temp_y                      0
app_temp_y                  0
clouds_y                    0
precip_y                    0
rh_y                        0
wind_spd_y                  0
wind_gust_spd_y             0
wind_dir_y                  0
weather_description         0
pres_y                      0
slp_y                       0
vis_y                       0
snow_y                      0
dtype: int64
```

In [225...
```
# Convert columns to object type
merged_df['Scheduled departure hour'] = merged_df['Scheduled departure hour'].astype(str)
merged_df['Scheduled departure minute'] = merged_df['Scheduled departure minute'].astype(str)
merged_df['Scheduled Arrival hour'] = merged_df['Scheduled Arrival hour'].astype(str)
merged_df['Scheduled Arrival minute'] = merged_df['Scheduled Arrival minute'].astype(str)
```

In [226...
```
merged_df.dtypes
```

Out[226…
```
Carrier Code                        object
Date (MM/DD/YYYY)           datetime64[ns]
Origin Airport                      object
Day Name                          category
Scheduled departure hour            object
Scheduled departure minute          object
Scheduled Arrival hour              object
Scheduled Arrival minute            object
Arrival Status                       int64
temp_x                             float64
app_temp_x                         float64
clouds_x                           float64
precip_x                           float64
rh_x                               float64
wind_spd_x                         float64
wind_gust_spd_x                    float64
wind_dir_x                         float64
weather                             object
pres_x                             float64
slp_x                              float64
vis_x                              float64
snow_x                             float64
temp_y                             float64
app_temp_y                         float64
clouds_y                             int64
precip_y                           float64
rh_y                                 int64
wind_spd_y                         float64
wind_gust_spd_y                    float64
wind_dir_y                           int64
weather_description                 object
pres_y                               int64
slp_y                                int64
vis_y                                int64
snow_y                             float64
dtype: object
```

In [227…
```
all_hours = pd.Series(range(24))

# Convert scheduled departure and arrival hour to categorical variable with specified categories
merged_df['Scheduled departure hour'] = pd.Categorical(merged_df['Scheduled departure hour'], categories=all_hours, ordered=True)
merged_df['Scheduled Arrival hour'] = pd.Categorical(merged_df['Scheduled Arrival hour'], categories=all_hours, ordered=True)
```

In [228…
```
all_minutes = pd.Series(range(60))

# Convert scheduled departure and arrival minute to categorical variable with specified categories
merged_df['Scheduled departure minute'] = pd.Categorical(merged_df['Scheduled departure minute'], categories=all_minutes, ordered=True)
merged_df['Scheduled Arrival minute'] = pd.Categorical(merged_df['Scheduled Arrival minute'], categories=all_minutes, ordered=True)
```

In [229…
```
merged_df.columns
```

Out[229…
```
Index(['Carrier Code', 'Date (MM/DD/YYYY)', 'Origin Airport', 'Day Name',
       'Scheduled departure hour', 'Scheduled departure minute',
       'Scheduled Arrival hour', 'Scheduled Arrival minute', 'Arrival Status',
       'temp_x', 'app_temp_x', 'clouds_x', 'precip_x', 'rh_x', 'wind_spd_x',
       'wind_gust_spd_x', 'wind_dir_x', 'weather', 'pres_x', 'slp_x', 'vis_x',
       'snow_x', 'temp_y', 'app_temp_y', 'clouds_y', 'precip_y', 'rh_y',
       'wind_spd_y', 'wind_gust_spd_y', 'wind_dir_y', 'weather_description',
       'pres_y', 'slp_y', 'vis_y', 'snow_y'],
      dtype='object')
```

In [230…
```
merged_df.drop(['weather','weather_description','Carrier Code'], axis=1, inplace=True)
```

In [231…
```
date_column = first_df['Date (MM/DD/YYYY)']

# Drop the date column before encoding
# Perform get_dummies
first_df_encoded = pd.get_dummies(merged_df.drop(columns=['Date (MM/DD/YYYY)']), drop_first=True)
```

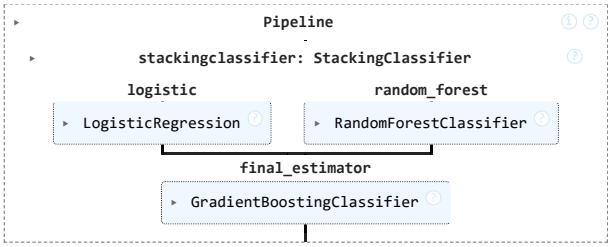
In [232…
```
# Convert 'Arrival Status' to categorical variable
first_df_encoded['Arrival Status'] = first_df_encoded['Arrival Status'].astype('category')
```

Training Model 1

In [233…
```
# Define base model
base_models = [
    ('logistic', LogisticRegression(fit_intercept=True, solver = 'newton-cg',multi_class='multinomial', penalty = None, max_iter = 100000)),
   ('random_forest', RandomForestClassifier(
    n_estimators=200,  # Number of trees in the forest
    max_depth=None,  # Maximum depth of the tree
    min_samples_split=2,  # Minimum number of samples required to split an internal node
    min_samples_leaf=1,  # Minimum number of samples required to be at a leaf node
    max_features=10,  # Number of features to consider when looking for the best split
    random_state=42  # Random state for reproducibility
))
]

# Define meta-model
meta_model = GradientBoostingClassifier(learning_rate = 0.001, max_features = 'log2', n_estimators = 500, min_samples_leaf = 1)

# Create the stacking classifier
stacking_clf = StackingClassifier(estimators=base_models, final_estimator=meta_model)

# Create pipeline with stacking classifier
pipeline = make_pipeline(stacking_clf)
```

```
# Train_test_split
X_train_1, X_test_1, y_train_1, y_test_1 = train_test_split(first_df_encoded.drop(columns = ['Arrival Status']), first_df_encoded['Arrival Status'], test_size=

# Train the stacking model
pipeline.fit(X_train_1, y_train_1)

# Print Train and Test scores
train_accuracy = pipeline.score(X_train_1, y_train_1)
test_accuracy = pipeline.score(X_test_1, y_test_1)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

Out[233…




```
Train Accuracy: 0.5649025069637883
Test Accuracy: 0.5011135857461024
```

Second Flight Model

In [234…

```
# Read the data for second and first flight details
second_df = pd.read_excel("Datasets/Second_Flight_Details.xlsx")
first_df_reload = pd.read_excel("Datasets/First_Flight_Details.xlsx")
second_df.head()
first_df_reload.head()
```

Out[234…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)
0	MQ	2019-01-01	3538	N523AE	ORD	SYR	11:45:00	11:43:00	-2	14:27:00	14:35:00	8
1	MQ	2019-01-01	3685	N538EG	ORD	SYR	18:50:00	18:50:00	0	21:39:00	21:59:00	20
2	MQ	2019-01-01	3946	N256NN	ORD	SYR	14:47:00	16:30:00	103	17:34:00	19:31:00	117
3	MQ	2019-01-01	4011	N223NN	ORD	SYR	22:10:00	22:09:00	-1	00:54:00	00:54:00	0
4	MQ	2022-01-01	4316	N900AE	ORD	SYR	15:28:00	00:00:00	0	18:15:00	00:00:00	0

Out[234…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)
0	UA	2023-01-01	2645	N23721	ORD	SYR	21:10:00	21:07:00	-3	23:57:00	23:47:00	-10
1	UA	2023-01-02	1998	N802UA	ORD	SYR	18:14:00	18:17:00	3	21:07:00	20:46:00	-21
2	UA	2023-01-03	1998	N854UA	ORD	SYR	18:14:00	18:13:00	-1	21:07:00	21:04:00	-3
3	UA	2023-01-04	1998	N893UA	ORD	SYR	18:24:00	18:41:00	17	21:17:00	21:31:00	14
4	UA	2020-01-05	2012	N825UA	ORD	SYR	19:00:00	19:21:00	21	21:47:00	21:55:00	8

In [235…

```
second_df.isna().sum()
first_df_reload.isna().sum()
```

Out[235…

```
Carrier Code                0
Date (MM/DD/YYYY)           0
Flight Number               0
Tail Number                31
Origin Airport              0
Destination Airport         0
Scheduled departure time    0
Actual departure time       0
Departure delay (Minutes)   0
Scheduled Arrival Time      0
Actual Arrival Time         0
Arrival Delay (Minutes)     0
dtype: int64
```

Out[235…

```
Carrier Code                0
Date (MM/DD/YYYY)           0
Flight Number               0
Tail Number                53
Origin Airport              0
Destination Airport         0
Scheduled departure time    0
Actual departure time       0
Departure delay (Minutes)   0
Scheduled Arrival Time      0
Actual Arrival Time         0
Arrival Delay (Minutes)     0
dtype: int64
```

Second flight data preprocessing

Merging second flight data with its previous flight data

Merging origin and destination weather data to flight data

In [236…
```
# Dropping Nans from both dataframes
second_df.dropna(subset=['Tail Number'], inplace=True)
second_df.isna().sum()

first_df_reload.dropna(subset=['Tail Number'], inplace=True)
first_df_reload.isna().sum()
```

Out[236…
```
Carrier Code                0
Date (MM/DD/YYYY)           0
Flight Number               0
Tail Number                 0
Origin Airport              0
Destination Airport         0
Scheduled departure time    0
Actual departure time       0
Departure delay (Minutes)   0
Scheduled Arrival Time      0
Actual Arrival Time         0
Arrival Delay (Minutes)     0
dtype: int64
```

Out[236…
```
Carrier Code                0
Date (MM/DD/YYYY)           0
Flight Number               0
Tail Number                 0
Origin Airport              0
Destination Airport         0
Scheduled departure time    0
Actual departure time       0
Departure delay (Minutes)   0
Scheduled Arrival Time      0
Actual Arrival Time         0
Arrival Delay (Minutes)     0
dtype: int64
```

In [237…
```
len(second_df)
```

Out[237…
```
7584
```

In [238…
```
# Apply the function to create the "Arrival Status" column
second_df['Arrival Status Second'] = second_df['Arrival Delay (Minutes)'].apply(determine_arrival_status)
first_df_reload['Arrival Status First'] = first_df_reload['Arrival Delay (Minutes)'].apply(determine_arrival_status)

first_df_reload.head()
second_df.head()
```

Out[238…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status First
0	UA	2023-01-01	2645	N23721	ORD	SYR	21:10:00	21:07:00	-3	23:57:00	23:47:00	-10	1
1	UA	2023-01-02	1998	N802UA	ORD	SYR	18:14:00	18:17:00	3	21:07:00	20:46:00	-21	1
2	UA	2023-01-03	1998	N854UA	ORD	SYR	18:14:00	18:13:00	-1	21:07:00	21:04:00	-3	0
3	UA	2023-01-04	1998	N893UA	ORD	SYR	18:24:00	18:41:00	17	21:17:00	21:31:00	14	2
4	UA	2020-01-05	2012	N825UA	ORD	SYR	19:00:00	19:21:00	21	21:47:00	21:55:00	8	2

Out[238…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status Second
0	MQ	2019-01-01	3538	N523AE	ORD	SYR	11:45:00	11:43:00	-2	14:27:00	14:35:00	8	2
1	MQ	2019-01-01	3685	N538EG	ORD	SYR	18:50:00	18:50:00	0	21:39:00	21:59:00	20	2
2	MQ	2019-01-01	3946	N256NN	ORD	SYR	14:47:00	16:30:00	103	17:34:00	19:31:00	117	2
3	MQ	2019-01-01	4011	N223NN	ORD	SYR	22:10:00	22:09:00	-1	00:54:00	00:54:00	0	0
4	MQ	2022-01-01	4316	N900AE	ORD	SYR	15:28:00	00:00:00	0	18:15:00	00:00:00	0	0

In [239…
```
first_df_reload.head()
second_df.head()
```

Out[239…

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status First
0	UA	2023-01-01	2645	N23721	ORD	SYR	21:10:00	21:07:00	-3	23:57:00	23:47:00	-10	1
1	UA	2023-01-02	1998	N802UA	ORD	SYR	18:14:00	18:17:00	3	21:07:00	20:46:00	-21	1
2	UA	2023-01-03	1998	N854UA	ORD	SYR	18:14:00	18:13:00	-1	21:07:00	21:04:00	-3	0
3	UA	2023-01-04	1998	N893UA	ORD	SYR	18:24:00	18:41:00	17	21:17:00	21:31:00	14	2
4	UA	2020-01-05	2012	N825UA	ORD	SYR	19:00:00	19:21:00	21	21:47:00	21:55:00	8	2

Out[239...

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status Second
0	MQ	2019-01-01	3538	N523AE	ORD	SYR	11:45:00	11:43:00	-2	14:27:00	14:35:00	8	2
1	MQ	2019-01-01	3685	N538EG	ORD	SYR	18:50:00	18:50:00	0	21:39:00	21:59:00	20	2
2	MQ	2019-01-01	3946	N256NN	ORD	SYR	14:47:00	16:30:00	103	17:34:00	19:31:00	117	2
3	MQ	2019-01-01	4011	N223NN	ORD	SYR	22:10:00	22:09:00	-1	00:54:00	00:54:00	0	0
4	MQ	2022-01-01	4316	N900AE	ORD	SYR	15:28:00	00:00:00	0	18:15:00	00:00:00	0	0

In [240...

```
columns_to_keep = ['Carrier Code','Date (MM/DD/YYYY)','Origin Airport','Scheduled departure time','Arrival Status First']

first_df_reload = first_df_reload[columns_to_keep]

first_df_reload.head()
```

Out[240...

	Carrier Code	Date (MM/DD/YYYY)	Origin Airport	Scheduled departure time	Arrival Status First
0	UA	2023-01-01	ORD	21:10:00	1
1	UA	2023-01-02	ORD	18:14:00	1
2	UA	2023-01-03	ORD	18:14:00	0
3	UA	2023-01-04	ORD	18:24:00	2
4	UA	2020-01-05	ORD	19:00:00	2

In [241...

```
# Merge the second_df with first_df, mapped using Date and Origin Airport
merged_df_new = pd.merge(second_df, first_df_reload, on=['Origin Airport', 'Date (MM/DD/YYYY)'], how='left')

merged_df_new.head()
```

Out[241...

	Carrier Code_x	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time_x	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status Second	Carrier Code_y	Scheduled departure time_y	Arrival Status First
0	MQ	2019-01-01	3538	N523AE	ORD	SYR	11:45:00	11:43:00	-2	14:27:00	14:35:00	8	2	NaN	NaN	NaN
1	MQ	2019-01-01	3685	N538EG	ORD	SYR	18:50:00	18:50:00	0	21:39:00	21:59:00	20	2	NaN	NaN	NaN
2	MQ	2019-01-01	3946	N256NN	ORD	SYR	14:47:00	16:30:00	103	17:34:00	19:31:00	117	2	NaN	NaN	NaN
3	MQ	2019-01-01	4011	N223NN	ORD	SYR	22:10:00	22:09:00	-1	00:54:00	00:54:00	0	0	NaN	NaN	NaN
4	MQ	2022-01-01	4316	N900AE	ORD	SYR	15:28:00	00:00:00	0	18:15:00	00:00:00	0	0	NaN	NaN	NaN

In [242...

```
len(merged_df_new)
merged_df_new.dtypes
```

Out[242...

```
10799
```

Out[242...

```
Carrier Code_x                    object
Date (MM/DD/YYYY)         datetime64[ns]
Flight Number                      int64
Tail Number                       object
Origin Airport                    object
Destination Airport               object
Scheduled departure time_x        object
Actual departure time             object
Departure delay (Minutes)          int64
Scheduled Arrival Time            object
Actual Arrival Time               object
Arrival Delay (Minutes)            int64
Arrival Status Second              int64
Carrier Code_y                    object
Scheduled departure time_y        object
Arrival Status First             float64
dtype: object
```

In [243...

```
syr_weather.head()
```

Out[243...

	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp	vis	snow	Date	Hour
0	7.20	4.30	87.00	1.50	85.00	4.59	9.80	150.00	Light rain	982.00	997.00	16.00	0.00	2019-01-01	0
1	8.90	8.90	100.00	0.50	79.00	5.09	11.80	190.00	Overcast clouds	984.00	999.00	16.00	0.00	2019-01-01	1
2	9.40	9.40	100.00	0.00	83.00	4.59	12.90	180.00	Overcast clouds	982.00	997.00	16.00	0.00	2019-01-01	2
3	11.10	11.10	87.00	0.00	80.00	8.19	13.90	200.00	Overcast clouds	980.00	996.00	16.00	0.00	2019-01-01	3
4	10.60	10.60	100.00	0.50	85.00	6.20	16.80	230.00	Overcast clouds	981.00	996.00	16.00	0.00	2019-01-01	4

In [244...

```
# Convert 'Scheduled departure time_x' and 'Scheduled departure time_y' columns to datetime format
merged_df_new['Scheduled departure time_x'] = pd.to_datetime(merged_df_new['Scheduled departure time_x'], format='%H:%M:%S')
merged_df_new['Scheduled departure time_y'] = pd.to_datetime(merged_df_new['Scheduled departure time_y'], format='%H:%M:%S')

# Calculate the time difference in hours (This is the difference between the departure times of the current and previous flights)
merged_df_new['time_diff'] = (merged_df_new['Scheduled departure time_x'] - merged_df_new['Scheduled departure time_y']).dt.total_seconds() / 3600

# Drop rows where the time difference is greater than or equal to 3 hours or less than 0
# Effectively, we are considering a time gap between the second and first flight between 3 hours.
merged_df_new = merged_df_new.drop(merged_df_new[(merged_df_new['time_diff'] > 3) | (merged_df_new['time_diff'] < 0)].index)
```

```
# Drop the 'time_diff' column
merged_df_new = merged_df_new.drop(columns=['time_diff'])

merged_df_new.head()
```

Out[244…

	Carrier Code_x	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time_x	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status Second	Carrier Code_y	Scheduled departure time_y	Arrival Status First
0	MQ	2019-01-01	3538	N523AE	ORD	SYR	1900-01-01 11:45:00	11:43:00	-2	14:27:00	14:35:00	8	2	NaN	NaT	NaN
1	MQ	2019-01-01	3685	N538EG	ORD	SYR	1900-01-01 18:50:00	18:50:00	0	21:39:00	21:59:00	20	2	NaN	NaT	NaN
2	MQ	2019-01-01	3946	N256NN	ORD	SYR	1900-01-01 14:47:00	16:30:00	103	17:34:00	19:31:00	117	2	NaN	NaT	NaN
3	MQ	2019-01-01	4011	N223NN	ORD	SYR	1900-01-01 22:10:00	22:09:00	-1	00:54:00	00:54:00	0	0	NaN	NaT	NaN
4	MQ	2022-01-01	4316	N900AE	ORD	SYR	1900-01-01 15:28:00	00:00:00	0	18:15:00	00:00:00	0	0	NaN	NaT	NaN

In [245…
```
merged_df_new.isna().sum()
```

Out[245…
```
Carrier Code_x                  0
Date (MM/DD/YYYY)               0
Flight Number                   0
Tail Number                     0
Origin Airport                  0
Destination Airport             0
Scheduled departure time_x      0
Actual departure time           0
Departure delay (Minutes)       0
Scheduled Arrival Time          0
Actual Arrival Time             0
Arrival Delay (Minutes)         0
Arrival Status Second           0
Carrier Code_y               1810
Scheduled departure time_y   1810
Arrival Status First         1810
dtype: int64
```

In [246…
```
merged_df_new.dropna(inplace=True)
```

In [247…
```
merged_df_new.head()
len(merged_df_new)
```

Out[247…

	Carrier Code_x	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Destination Airport	Scheduled departure time_x	Actual departure time	Departure delay (Minutes)	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Arrival Status Second	Carrier Code_y	Scheduled departure time_y	Arrival Status First
48	MQ	2019-01-07	3685	N533AE	ORD	SYR	1900-01-01 18:50:00	19:09:00	19	21:37:00	21:48:00	11	2	UA	1900-01-01 18:00:00	1.00
54	MQ	2019-01-08	3685	N527EA	ORD	SYR	1900-01-01 18:50:00	18:47:00	-3	21:37:00	21:27:00	-10	1	UA	1900-01-01 18:00:00	2.00
61	MQ	2019-01-09	3685	N543EA	ORD	SYR	1900-01-01 18:50:00	18:42:00	-8	21:37:00	21:47:00	10	2	UA	1900-01-01 18:00:00	1.00
67	MQ	2019-01-10	3685	N532EA	ORD	SYR	1900-01-01 18:50:00	18:48:00	-2	21:37:00	21:48:00	11	2	UA	1900-01-01 18:00:00	2.00
71	MQ	2019-01-11	3685	N239NN	ORD	SYR	1900-01-01 18:50:00	18:46:00	-4	21:37:00	21:29:00	-8	1	UA	1900-01-01 18:00:00	0.00

Out[247…
```
1465
```

In [248…
```
columns_to_keep = ['Carrier Code_x', 'Date (MM/DD/YYYY)','Origin Airport', 'Scheduled departure time_x', 'Actual departure time',
                   'Scheduled Arrival Time', 'Actual Arrival Time','Arrival Status Second','Carrier Code_y','Arrival Status First']

merged_df_new = merged_df_new[columns_to_keep]
merged_df_new.head()
```

Out[248…

	Carrier Code_x	Date (MM/DD/YYYY)	Origin Airport	Scheduled departure time_x	Actual departure time	Scheduled Arrival Time	Actual Arrival Time	Arrival Status Second	Carrier Code_y	Arrival Status First
48	MQ	2019-01-07	ORD	1900-01-01 18:50:00	19:09:00	21:37:00	21:48:00	2	UA	1.00
54	MQ	2019-01-08	ORD	1900-01-01 18:50:00	18:47:00	21:37:00	21:27:00	1	UA	2.00
61	MQ	2019-01-09	ORD	1900-01-01 18:50:00	18:42:00	21:37:00	21:47:00	2	UA	1.00
67	MQ	2019-01-10	ORD	1900-01-01 18:50:00	18:48:00	21:37:00	21:48:00	2	UA	2.00
71	MQ	2019-01-11	ORD	1900-01-01 18:50:00	18:46:00	21:37:00	21:29:00	1	UA	0.00

In [249…
```
merged_df_new.dtypes
```

Out[249…
```
Carrier Code_x                    object
Date (MM/DD/YYYY)         datetime64[ns]
Origin Airport                    object
Scheduled departure time_x    datetime64[ns]
Actual departure time             object
Scheduled Arrival Time            object
Actual Arrival Time               object
Arrival Status Second              int64
Carrier Code_y                    object
Arrival Status First             float64
dtype: object
```

In [250…
```
merged_df_new['Day of Week'] = merged_df_new['Date (MM/DD/YYYY)'].dt.dayofweek

# Map the day of the week to day names
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
merged_df_new['Day Name'] = merged_df_new['Day of Week'].map(lambda x: day_names[x])

merged_df_new['Day Name'] = pd.Categorical(merged_df_new['Day Name'], categories=day_names)
```

In [251…
```
# Convert 'Scheduled Arrival Time' to datetime
merged_df_new['Scheduled Arrival Time'] = pd.to_datetime(merged_df_new['Scheduled Arrival Time'], format='%H:%M:%S')
```

In [252…
```
# Extract hour and minute components
merged_df_new['Scheduled departure hour'] = merged_df_new['Scheduled departure time_x'].dt.hour
merged_df_new['Scheduled departure minute'] = merged_df_new['Scheduled departure time_x'].dt.minute

merged_df_new['Scheduled Arrival hour'] = merged_df_new['Scheduled Arrival Time'].dt.hour
merged_df_new['Scheduled Arrival minute'] = merged_df_new['Scheduled Arrival Time'].dt.minute
```

In [253…
```
# Merge the merged_df_new with syr_weather data, mapped using Date and Scheduled Arrival Hour
merged_df_new = pd.merge(merged_df_new, syr_weather, left_on=["Date (MM/DD/YYYY)", "Scheduled Arrival hour"], right_on=["Date", "Hour"], how="left")
merged_df_new.head()
```

Out[253…

	Carrier Code_x	Date (MM/DD/YYYY)	Origin Airport	Scheduled departure time_x	Actual departure time	Scheduled Arrival Time	Actual Arrival Time	Arrival Status Second	Carrier Code_y	Arrival Status First	...	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp
0	MQ	2019-01-07	ORD	1900-01-01 18:50:00	19:09:00	1900-01-01 21:37:00	21:48:00	2	UA	1.00	...	5.70	13.20	130.00	Light snow	1,001.00	1,017.0(
1	MQ	2019-01-08	ORD	1900-01-01 18:50:00	18:47:00	1900-01-01 21:37:00	21:27:00	1	UA	2.00	...	2.10	7.60	100.00	Mix snow/rain	985.00	1,000.0(
2	MQ	2019-01-09	ORD	1900-01-01 18:50:00	18:42:00	1900-01-01 21:37:00	21:47:00	2	UA	1.00	...	4.59	11.30	280.00	Light snow	989.00	1,004.0(
3	MQ	2019-01-10	ORD	1900-01-01 18:50:00	18:48:00	1900-01-01 21:37:00	21:48:00	2	UA	2.00	...	8.80	11.80	290.00	Overcast clouds	1,002.00	1,018.0(
4	MQ	2019-01-11	ORD	1900-01-01 18:50:00	18:46:00	1900-01-01 21:37:00	21:29:00	1	UA	0.00	...	3.60	6.00	250.00	Overcast clouds	1,015.00	1,031.0(

5 rows × 31 columns

In [254…
```
# Merge the merged_df_new with All_weather data, mapped using Date, Scheduled Arrival Hour and Origin Airport
merged_df_new = pd.merge(merged_df_new, All_weather, how='left', left_on=['Date (MM/DD/YYYY)', 'Scheduled departure hour', 'Origin Airport'], right_on=['Date',
```

In [255…
```
merged_df_new.drop(['weather_description', 'weather', 'Location', 'Scheduled departure time_x', 'Actual departure time', 'Scheduled Arrival Time', 'Actual Arri
```

In [256…
```
merged_df_new.drop(['Date_x','Hour_x','Date_y', 'Hour_y'], axis=1, inplace=True)
```

In [257…
```
merged_df_new.head()
```

Out[257…

	Carrier Code_x	Origin Airport	Arrival Status Second	Carrier Code_y	Arrival Status First	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	...	clouds_y	precip_y	rh_y	wind_spd_y	wind_gust_spd_y	wind_dir_y
0	MQ	ORD	2	UA	1.00	Monday	18	50	21	37	...	50	0.00	82	9.80	15.40	25(
1	MQ	ORD	1	UA	2.00	Tuesday	18	50	21	37	...	100	0.00	59	10.30	17.00	28(
2	MQ	ORD	2	UA	1.00	Wednesday	18	50	21	37	...	25	0.00	58	3.60	10.00	33(
3	MQ	ORD	2	UA	2.00	Thursday	18	50	21	37	...	87	0.00	71	0.80	1.20	19(
4	MQ	ORD	1	UA	0.00	Friday	18	50	21	37	...	100	0.00	68	1.50	3.20	19(

5 rows × 34 columns

In [258…
```
# Convert columns to object type
merged_df_new['Scheduled departure hour'] = merged_df_new['Scheduled departure hour'].astype(str)
merged_df_new['Scheduled departure minute'] = merged_df_new['Scheduled departure minute'].astype(str)
merged_df_new['Scheduled Arrival hour'] = merged_df_new['Scheduled Arrival hour'].astype(str)
merged_df_new['Scheduled Arrival minute'] = merged_df_new['Scheduled Arrival minute'].astype(str)
```

In [259…
```
all_hours = pd.Series(range(24))

# Convert to categorical variable with specified categories
merged_df_new['Scheduled departure hour'] = pd.Categorical(merged_df_new['Scheduled departure hour'], categories=all_hours, ordered=True)
merged_df_new['Scheduled Arrival hour'] = pd.Categorical(merged_df_new['Scheduled departure hour'], categories=all_hours, ordered=True)
```

In [260…
```
all_minutes = pd.Series(range(60))

# Convert to categorical variable with specified categories
merged_df_new['Scheduled departure minute'] = pd.Categorical(merged_df_new['Scheduled departure minute'], categories=all_minutes, ordered=True)
merged_df_new['Scheduled Arrival minute'] = pd.Categorical(merged_df_new['Scheduled Arrival minute'], categories=all_minutes, ordered=True)
```

In [261…
```
merged_df_new.dtypes
```

Out[261…
```
Carrier Code_x                 object
Origin Airport                 object
Arrival Status Second           int64
Carrier Code_y                 object
Arrival Status First          float64
Day Name                     category
Scheduled departure hour     category
Scheduled departure minute   category
Scheduled Arrival hour       category
Scheduled Arrival minute     category
temp_x                        float64
app_temp_x                    float64
clouds_x                      float64
precip_x                      float64
rh_x                          float64
wind_spd_x                    float64
wind_gust_spd_x               float64
wind_dir_x                    float64
pres_x                        float64
slp_x                         float64
vis_x                         float64
snow_x                        float64
temp_y                        float64
app_temp_y                    float64
clouds_y                        int64
precip_y                      float64
rh_y                            int64
wind_spd_y                    float64
wind_gust_spd_y               float64
wind_dir_y                      int64
pres_y                          int64
slp_y                           int64
vis_y                           int64
snow_y                        float64
dtype: object
```

In [262…
```
len(merged_df_new)
```

Out[262…
```
1465
```

In [263…
```
merged_df_new['Arrival Status Second'] = merged_df_new['Arrival Status Second'].astype('category')
merged_df_new['Arrival Status First'] = merged_df_new['Arrival Status First'].astype('category')
```

In [264…
```
Arr_status = merged_df_new['Arrival Status Second']

merged_df_new = pd.get_dummies(merged_df_new.drop(columns=['Arrival Status Second','Carrier Code_x','Carrier Code_y']), drop_first=True)

merged_df_new['Arrival Status Second'] = Arr_status

merged_df_new.head()
```

Out[264…

	temp_x	app_temp_x	clouds_x	precip_x	rh_x	wind_spd_x	wind_gust_spd_x	wind_dir_x	pres_x	slp_x	…	Scheduled Arrival minute_51	Scheduled Arrival minute_52	Scheduled Arrival minute_53	Scheduled Arrival minute_54	Scheduled Arrival minute_55
0	-0.60	-6.10	87.00	0.50	49.00	5.70	13.20	130.00	1,001.00	1,017.00	…	False	False	False	False	False
1	1.10	-1.30	100.00	0.50	88.00	2.10	7.60	100.00	985.00	1,000.00	…	False	False	False	False	False
2	-2.80	-8.10	100.00	0.50	100.00	4.59	11.30	280.00	989.00	1,004.00	…	False	False	False	False	False
3	-7.20	-16.10	100.00	0.00	80.00	8.80	11.80	290.00	1,002.00	1,018.00	…	False	False	False	False	False
4	-9.40	-15.40	87.00	0.00	69.00	3.60	6.00	250.00	1,015.00	1,031.00	…	False	False	False	False	False

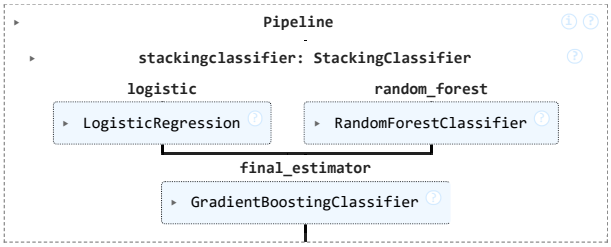
5 rows × 199 columns

Training Model 2

In [265…
```
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import GradientBoostingClassifier

# Define base models
base_models = [
    ('logistic', LogisticRegression(fit_intercept=True, solver = 'newton-cg',multi_class='multinomial', penalty = None, max_iter = 100000)),
    ('random_forest', RandomForestClassifier(
    n_estimators=100,  # Number of trees in the forest
    max_depth=None,  # Maximum depth of the tree
    min_samples_split=2,  # Minimum number of samples required to split an internal node
```

```
    min_samples_leaf=1,  # Minimum number of samples required to be at a leaf node
    max_features=10,  # Number of features to consider when looking for the best split
    random_state=42  # Random state for reproducibility
))]

# Define meta-model
meta_model = GradientBoostingClassifier(learning_rate = 0.01, max_features = 'sqrt', n_estimators = 300, min_samples_leaf = 2)

# Create the stacking classifier
stacking_clf = StackingClassifier(estimators=base_models, final_estimator=meta_model)

# Create pipeline with stacking classifier
pipeline_2 = make_pipeline(stacking_clf)

# Train_test_split
X_train_2, X_test_2, y_train_2, y_test_2 = train_test_split(merged_df_new.drop(columns = ['Arrival Status Second']), merged_df_new['Arrival Status Second'], te

# Train the stacking model
pipeline_2.fit(X_train_2, y_train_2)

# Print Train and Test accuracies
train_accuracy = pipeline_2.score(X_train_2, y_train_2)
test_accuracy = pipeline_2.score(X_test_2, y_test_2)

print("Train Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
```

Out[265…




```
Train Accuracy: 0.5750853242320819
Test Accuracy: 0.5290102389078498
```

Final Predictions

In [266…

```
# Divided the final predictions csv into Final_Earlier_flights.xlsx and Final_Later_flights.xlsx for simplicity.
# They are being read separately.
even_rows = pd.read_excel("Datasets/Final_Earlier_flights.xlsx")

odd_rows = pd.read_excel("Datasets/Final_Later_flights.xlsx")
```

In [267…

```
even_rows.head()
odd_rows.head()
```

Out[267…

	DATE	DAY	FLIGHT NUMBER	ORIGIN	DEPARTURE TIME	ARRIVAL TIME	ARRIVAL STATUS	ARRIVAL STATUS_Prev_flight_early	ARRIVAL STATUS_Prev_flight_ontime	ARRIVAL STATUS_Prev_flight_late
0	2024-04-19	FRIDAY	UA 1400	ORD	18:52:00	21:47:00	NaN	NaN	NaN	NaN
1	2024-04-19	FRIDAY	B6 116	JFK	13:34:00	14:51:00	NaN	NaN	NaN	NaN
2	2024-04-19	FRIDAY	WN 5285	MCO	11:35:00	14:20:00	NaN	NaN	NaN	NaN
3	2024-04-20	SATURDAY	UA 1400	ORD	18:52:00	21:47:00	NaN	NaN	NaN	NaN
4	2024-04-20	SATURDAY	B6 116	JFK	13:25:00	14:41:00	NaN	NaN	NaN	NaN

Out[267…

	DATE	DAY	FLIGHT NUMBER	ORIGIN	DEPARTURE TIME	ARRIVAL TIME	ARRIVAL STATUS	ARRIVAL STATUS_Prev_flight_early	ARRIVAL STATUS_Prev_flight_ontime	ARRIVAL STATUS_Prev_flight_late
0	2024-04-19	FRIDAY	AA 3402	ORD	19:59:00	22:52:00	NaN	NaN	NaN	NaN
1	2024-04-19	FRIDAY	DL 5182	JFK	14:55:00	16:21:00	NaN	NaN	NaN	NaN
2	2024-04-19	FRIDAY	B6 656	MCO	13:35:00	16:25:00	NaN	NaN	NaN	NaN
3	2024-04-20	SATURDAY	AA 3402	ORD	19:59:00	22:52:00	NaN	NaN	NaN	NaN
4	2024-04-20	SATURDAY	DL 5182	JFK	14:55:00	16:21:00	NaN	NaN	NaN	NaN

Read and preprocess weather forecast data.

In [268…

```
# Reading the forecast data.
test_merged_weather_df = pd.read_csv("Datasets/Forecast_Data.csv")
test_merged_weather_df.head()
```

Out[268…

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	weather	pres	slp	vis	snow	Location
0	2024-04-18:20	2024-04-18T16:00:00	5.20	0.90	94	0.00	76	6.40	9.30	50	Overcast clouds	1,019.50	1,019.60	16.00	0	JFK
1	2024-04-18:21	2024-04-18T17:00:00	9.00	9.00	92	0.00	74	6.40	9.80	50	Overcast clouds	1,019.00	1,019.10	17.49	0	JFK
2	2024-04-18:22	2024-04-18T18:00:00	8.90	8.90	90	0.00	73	6.00	9.00	50	Overcast clouds	1,019.50	1,019.60	16.99	0	JFK
3	2024-04-18:23	2024-04-18T19:00:00	8.40	8.40	94	0.00	76	5.20	8.10	50	Overcast clouds	1,021.00	1,021.10	15.80	0	JFK
4	2024-04-19:00	2024-04-18T20:00:00	8.20	8.20	95	0.00	72	5.20	7.70	50	Overcast clouds	1,021.00	1,021.10	15.70	0	JFK

In [269…
```
columns_to_keep = ['datetime', 'timestamp_local', 'temp', 'app_temp', 'clouds', 'precip', 'rh', 'wind_spd', 'wind_gust_spd', 'wind_dir', 'pres', 'slp', 'vis',
test_merged_weather_df = test_merged_weather_df.loc[:, columns_to_keep]

test_merged_weather_df.head()
```

Out[269…

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	pres	slp	vis	snow	Location
0	2024-04-18:20	2024-04-18T16:00:00	5.20	0.90	94	0.00	76	6.40	9.30	50	1,019.50	1,019.60	16.00	0	JFK
1	2024-04-18:21	2024-04-18T17:00:00	9.00	9.00	92	0.00	74	6.40	9.80	50	1,019.00	1,019.10	17.49	0	JFK
2	2024-04-18:22	2024-04-18T18:00:00	8.90	8.90	90	0.00	73	6.00	9.00	50	1,019.50	1,019.60	16.99	0	JFK
3	2024-04-18:23	2024-04-18T19:00:00	8.40	8.40	94	0.00	76	5.20	8.10	50	1,021.00	1,021.10	15.80	0	JFK
4	2024-04-19:00	2024-04-18T20:00:00	8.20	8.20	95	0.00	72	5.20	7.70	50	1,021.00	1,021.10	15.70	0	JFK

In [270…
```
test_merged_weather_df.dtypes
```

Out[270…
```
datetime            object
timestamp_local     object
temp               float64
app_temp           float64
clouds               int64
precip             float64
rh                   int64
wind_spd           float64
wind_gust_spd      float64
wind_dir             int64
pres               float64
slp                float64
vis                float64
snow                 int64
Location            object
dtype: object
```

In [271…
```
test_merged_weather_df['timestamp_local'] = pd.to_datetime(test_merged_weather_df['timestamp_local'])
```

In [272…
```
test_merged_weather_df['Date'] = test_merged_weather_df['timestamp_local'].dt.date
test_merged_weather_df['Hour'] = test_merged_weather_df['timestamp_local'].dt.hour

test_merged_weather_df.head()
```

Out[272…

	datetime	timestamp_local	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	pres	slp	vis	snow	Location	Date	Hour
0	2024-04-18:20	2024-04-18 16:00:00	5.20	0.90	94	0.00	76	6.40	9.30	50	1,019.50	1,019.60	16.00	0	JFK	2024-04-18	16
1	2024-04-18:21	2024-04-18 17:00:00	9.00	9.00	92	0.00	74	6.40	9.80	50	1,019.00	1,019.10	17.49	0	JFK	2024-04-18	17
2	2024-04-18:22	2024-04-18 18:00:00	8.90	8.90	90	0.00	73	6.00	9.00	50	1,019.50	1,019.60	16.99	0	JFK	2024-04-18	18
3	2024-04-18:23	2024-04-18 19:00:00	8.40	8.40	94	0.00	76	5.20	8.10	50	1,021.00	1,021.10	15.80	0	JFK	2024-04-18	19
4	2024-04-19:00	2024-04-18 20:00:00	8.20	8.20	95	0.00	72	5.20	7.70	50	1,021.00	1,021.10	15.70	0	JFK	2024-04-18	20

Preprocess the csv data for both previous and late flights

In [273…
```
even_rows.dtypes
```

Out[273…
```
DATE                                datetime64[ns]
DAY                                         object
FLIGHT NUMBER                               object
ORIGIN                                      object
DEPARTURE TIME                              object
ARRIVAL TIME                                object
ARRIVAL STATUS                             float64
ARRIVAL STATUS_Prev_flight_early           float64
ARRIVAL STATUS_Prev_flight_ontime          float64
ARRIVAL STATUS_Prev_flight_late            float64
dtype: object
```

In [274…
```
even_rows = even_rows.rename(columns={'DATE': 'Date (MM/DD/YYYY)'})
```

In [275…
```
even_rows['Date (MM/DD/YYYY)'] = pd.to_datetime(even_rows['Date (MM/DD/YYYY)'])

even_rows['Day of Week'] = even_rows['Date (MM/DD/YYYY)'].dt.dayofweek

# Map the day of the week to day names
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
```

```
even_rows['Day Name'] = even_rows['Day of Week'].map(lambda x: day_names[x])

even_rows['Day Name'] = pd.Categorical(even_rows['Day Name'], categories=day_names)

print(even_rows['Day Name'].cat.categories)
```

```
Index(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
       'Sunday'],
      dtype='object')
```

In [276…
```
# Convert 'Scheduled departure time' to datetime
even_rows['Scheduled departure time'] = pd.to_datetime(even_rows['DEPARTURE TIME'], format='%H:%M:%S')

even_rows['Scheduled Arrival Time'] = pd.to_datetime(even_rows['ARRIVAL TIME'], format='%H:%M:%S')
```

In [277…
```
# Extract hour, minute, and second components
even_rows['Scheduled departure hour'] = even_rows['Scheduled departure time'].dt.hour
even_rows['Scheduled departure minute'] = even_rows['Scheduled departure time'].dt.minute

even_rows['Scheduled Arrival hour'] = even_rows['Scheduled Arrival Time'].dt.hour
even_rows['Scheduled Arrival minute'] = even_rows['Scheduled Arrival Time'].dt.minute
```

In [278…
```
even_rows.drop(['FLIGHT NUMBER', 'Scheduled departure time', 'Scheduled Arrival Time', 'Day of Week'], axis=1, inplace=True)
even_rows.drop(['DEPARTURE TIME', 'ARRIVAL TIME', 'ARRIVAL STATUS','ARRIVAL STATUS_Prev_flight_early','ARRIVAL STATUS_Prev_flight_ontime', 'ARRIVAL STATUS_Prev
```

In [279…
```
# Define the new column names and order
new_column_names = [ 'Date (MM/DD/YYYY)', 'Origin Airport', 'Day Name',
                     'Scheduled departure hour', 'Scheduled departure minute',
                     'Scheduled Arrival hour', 'Scheduled Arrival minute']

# Rename the columns
even_rows = even_rows.rename(columns={
    'ORIGIN': 'Origin Airport'
})

# Reorder the columns
even_rows = even_rows.reindex(columns=new_column_names)
even_rows.dtypes
```

Out[279…
```
Date (MM/DD/YYYY)             datetime64[ns]
Origin Airport                        object
Day Name                            category
Scheduled departure hour               int32
Scheduled departure minute             int32
Scheduled Arrival hour                 int32
Scheduled Arrival minute               int32
dtype: object
```

In [280…
```
odd_rows.dtypes
```

Out[280…
```
DATE                                datetime64[ns]
DAY                                         object
FLIGHT NUMBER                               object
ORIGIN                                      object
DEPARTURE TIME                              object
ARRIVAL TIME                                object
ARRIVAL STATUS                             float64
ARRIVAL STATUS_Prev_flight_early           float64
ARRIVAL STATUS_Prev_flight_ontime          float64
ARRIVAL STATUS_Prev_flight_late            float64
dtype: object
```

In [281…
```
odd_rows = odd_rows.rename(columns={'DATE': 'Date (MM/DD/YYYY)'})
```

In [282…
```
odd_rows['Date (MM/DD/YYYY)'] = pd.to_datetime(odd_rows['Date (MM/DD/YYYY)'])

odd_rows['Day of Week'] = odd_rows['Date (MM/DD/YYYY)'].dt.dayofweek

# Map the day of the week to day names
day_names = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday']
odd_rows['Day Name'] = odd_rows['Day of Week'].map(lambda x: day_names[x])

odd_rows['Day Name'] = pd.Categorical(odd_rows['Day Name'], categories=day_names)

print(odd_rows['Day Name'].cat.categories)
```

```
Index(['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday',
       'Sunday'],
      dtype='object')
```

In [283…
```
# Convert 'Scheduled departure time' and 'Scheduled Arrival Time' to datetime
odd_rows['Scheduled departure time'] = pd.to_datetime(odd_rows['DEPARTURE TIME'], format='%H:%M:%S')

odd_rows['Scheduled Arrival Time'] = pd.to_datetime(odd_rows['ARRIVAL TIME'], format='%H:%M:%S')
```

In [284…
```
# Extract hour and minute components
odd_rows['Scheduled departure hour'] = odd_rows['Scheduled departure time'].dt.hour
odd_rows['Scheduled departure minute'] = odd_rows['Scheduled departure time'].dt.minute

odd_rows['Scheduled Arrival hour'] = odd_rows['Scheduled Arrival Time'].dt.hour
odd_rows['Scheduled Arrival minute'] = odd_rows['Scheduled Arrival Time'].dt.minute
```

In [285…
```
odd_rows.drop(['FLIGHT NUMBER', 'Scheduled departure time', 'Scheduled Arrival Time', 'Day of Week'], axis=1, inplace=True)
odd_rows.drop(['DEPARTURE TIME', 'ARRIVAL TIME', 'ARRIVAL STATUS','ARRIVAL STATUS_Prev_flight_early','ARRIVAL STATUS_Prev_flight_ontime', 'ARRIVAL STATUS_Prev_
```

In [286…
```
# Map the day of the week to day names
status = ['0', '1', '2']
# Randomly assigning arrival status 1 to all the previous flights at this stage
```

```
odd_rows['Arrival Status First'] = '1'

odd_rows['Arrival Status First'] = pd.Categorical(odd_rows['Arrival Status First'], categories=status)

print(odd_rows['Arrival Status First'].cat.categories)
```

```
Index(['0', '1', '2'], dtype='object')
```

In [287…
```
odd_rows.head()
```

Out[287…

	Date (MM/DD/YYYY)	DAY	ORIGIN	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	Arrival Status First
0	2024-04-19	FRIDAY	ORD	Friday	19	59	22	52	1
1	2024-04-19	FRIDAY	JFK	Friday	14	55	16	21	1
2	2024-04-19	FRIDAY	MCO	Friday	13	35	16	25	1
3	2024-04-20	SATURDAY	ORD	Saturday	19	59	22	52	1
4	2024-04-20	SATURDAY	JFK	Saturday	14	55	16	21	1

In [288…
```
# Define the new column names and order
new_column_names = ['Date (MM/DD/YYYY)', 'Origin Airport',
                    'Arrival Status First', 'Day Name',
                    'Scheduled departure hour', 'Scheduled departure minute',
                    'Scheduled Arrival hour', 'Scheduled Arrival minute']

# Rename the columns
odd_rows = odd_rows.rename(columns={
    'ORIGIN': 'Origin Airport'
})

# Reorder the columns
odd_rows = odd_rows.reindex(columns=new_column_names)
odd_rows.dtypes
```

Out[288…
```
Date (MM/DD/YYYY)             datetime64[ns]
Origin Airport                        object
Arrival Status First                category
Day Name                            category
Scheduled departure hour               int32
Scheduled departure minute             int32
Scheduled Arrival hour                 int32
Scheduled Arrival minute               int32
dtype: object
```

In [289…
```
test_merged_weather_df['Date'] = pd.to_datetime(test_merged_weather_df['Date'])
```

In [290…
```
test_merged_weather_df.drop(['datetime', 'timestamp_local'], axis=1, inplace=True)
```

In [291…
```
even_rows.dtypes
```

Out[291…
```
Date (MM/DD/YYYY)             datetime64[ns]
Origin Airport                        object
Day Name                            category
Scheduled departure hour               int32
Scheduled departure minute             int32
Scheduled Arrival hour                 int32
Scheduled Arrival minute               int32
dtype: object
```

In [292…
```
# Filter test_merged_weather_df to include only rows where Location is SYR
filtered_weather_df = test_merged_weather_df[test_merged_weather_df['Location'] == 'SYR']
filtered_weather_df.head()

# Merge even_rows with the filtered_weather_df, which has the destination airport weather info.
merged_df_test_even = pd.merge(even_rows, filtered_weather_df, left_on=["Date (MM/DD/YYYY)", "Scheduled Arrival hour"], right_on=["Date", "Hour"], how="left")

merged_df_test_even.head()
```

Out[292…

	temp	app_temp	clouds	precip	rh	wind_spd	wind_gust_spd	wind_dir	pres	slp	vis	snow	Location	Date	Hour
324	4.80	2.10	87	0.00	76	3.20	4.90	90	1,002.00	1,017.40	12.40	0	SYR	2024-04-18	16
325	12.60	12.60	91	0.76	77	3.20	4.20	70	1,002.50	1,017.90	12.50	0	SYR	2024-04-18	17
326	12.10	12.10	88	0.50	77	3.60	4.20	80	1,003.50	1,018.90	12.40	0	SYR	2024-04-18	18
327	11.60	11.60	92	0.25	78	3.20	4.10	80	1,003.50	1,018.90	12.50	0	SYR	2024-04-18	19
328	11.00	11.00	89	0.00	80	2.80	4.10	90	1,003.50	1,019.00	12.70	0	SYR	2024-04-18	20

Out[292...

	Date (MM/DD/YYYY)	Origin Airport	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	temp	app_temp	clouds	...	wind_spd	wind_gust_spd	wind_dir	pres	slp	vis
0	2024-04-19	ORD	Friday	18	52	21	47	12.30	12.30	68	...	5.20	8.00	260	999.50	1,014.90	18.80
1	2024-04-19	JFK	Friday	13	34	14	51	16.30	16.30	84	...	9.20	14.20	170	996.00	1,011.20	13.30
2	2024-04-19	MCO	Friday	11	35	14	20	16.30	16.30	84	...	9.20	14.20	170	996.00	1,011.20	13.30
3	2024-04-20	ORD	Saturday	18	52	21	47	6.60	2.90	28	...	6.26	9.20	280	998.50	1,014.20	24.13
4	2024-04-20	JFK	Saturday	13	25	14	41	10.60	10.60	75	...	8.40	11.80	280	997.00	1,012.60	17.50

5 rows × 22 columns

In [293...

```
# Merge odd_rows with the filtered_weather_df, which has the destination airport weather info.
merged_df_test_odd = pd.merge(odd_rows, filtered_weather_df, left_on=["Date (MM/DD/YYYY)", "Scheduled Arrival hour"], right_on=["Date", "Hour"], how="left")

merged_df_test_odd.head()
```

Out[293...

	Date (MM/DD/YYYY)	Origin Airport	Arrival Status First	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	temp	app_temp	...	wind_spd	wind_gust_spd	wind_dir	pres	slp	vi
0	2024-04-19	ORD	1	Friday	19	59	22	52	11.50	11.50	...	5.60	8.60	260	1,000.00	1,015.50	20.9
1	2024-04-19	JFK	1	Friday	14	55	16	21	15.90	15.90	...	9.60	14.40	180	996.00	1,011.20	16.5
2	2024-04-19	MCO	1	Friday	13	35	16	25	15.90	15.90	...	9.60	14.40	180	996.00	1,011.20	16.5
3	2024-04-20	ORD	1	Saturday	19	59	22	52	6.00	2.30	...	5.73	8.40	280	999.00	1,014.70	24.1
4	2024-04-20	JFK	1	Saturday	14	55	16	21	10.30	10.30	...	8.66	12.00	280	996.50	1,012.00	21.3

5 rows × 23 columns

In [294...

```
merged_df_test_even.isna().sum()
```

Out[294...

```
Date (MM/DD/YYYY)            0
Origin Airport               0
Day Name                     0
Scheduled departure hour     0
Scheduled departure minute   0
Scheduled Arrival hour       0
Scheduled Arrival minute     0
temp                         0
app_temp                     0
clouds                       0
precip                       0
rh                           0
wind_spd                     0
wind_gust_spd                0
wind_dir                     0
pres                         0
slp                          0
vis                          0
snow                         0
Location                     0
Date                         0
Hour                         0
dtype: int64
```

In [295...

```
merged_df_test_odd.isna().sum()
```

Out[295…

```
Date (MM/DD/YYYY)             0
Origin Airport                0
Arrival Status First          0
Day Name                      0
Scheduled departure hour      0
Scheduled departure minute    0
Scheduled Arrival hour        0
Scheduled Arrival minute      0
temp                          0
app_temp                      0
clouds                        0
precip                        0
rh                            0
wind_spd                      0
wind_gust_spd                 0
wind_dir                      0
pres                          0
slp                           0
vis                           0
snow                          0
Location                      0
Date                          0
Hour                          0
dtype: int64
```

In [296…

```
test_merged_weather_df.dtypes
```

Out[296…

```
temp                    float64
app_temp                float64
clouds                    int64
precip                  float64
rh                        int64
wind_spd                float64
wind_gust_spd           float64
wind_dir                  int64
pres                    float64
slp                     float64
vis                     float64
snow                      int64
Location                 object
Date            datetime64[ns]
Hour                      int32
dtype: object
```

In [297…

```
# Merge merged_df_test_even with the test_merged_weather_df, which has all airports weather info, and map it with the matching origin airport.
merged_df_test_even = pd.merge(merged_df_test_even, test_merged_weather_df, how='left', left_on=['Date (MM/DD/YYYY)', 'Scheduled departure hour', 'Origin Airpo
                               right_on=['Date', 'Hour', 'Location'])
```

In [298…

```
merged_df_test_even.isna().sum()
```

Out[298…

```
Date (MM/DD/YYYY)             0
Origin Airport                0
Day Name                      0
Scheduled departure hour      0
Scheduled departure minute    0
Scheduled Arrival hour        0
Scheduled Arrival minute      0
temp_x                        0
app_temp_x                    0
clouds_x                      0
precip_x                      0
rh_x                          0
wind_spd_x                    0
wind_gust_spd_x               0
wind_dir_x                    0
pres_x                        0
slp_x                         0
vis_x                         0
snow_x                        0
Location_x                    0
Date_x                        0
Hour_x                        0
temp_y                        0
app_temp_y                    0
clouds_y                      0
precip_y                      0
rh_y                          0
wind_spd_y                    0
wind_gust_spd_y               0
wind_dir_y                    0
pres_y                        0
slp_y                         0
vis_y                         0
snow_y                        0
Location_y                    0
Date_y                        0
Hour_y                        0
dtype: int64
```

In [299…

```
len(merged_df_test_even)
```

Out[299…

```
12
```

In [300…

```
# Merge merged_df_test_odd with the test_merged_weather_df, which has all airports weather info, and map it with the matching origin airport.
merged_df_test_odd = pd.merge(merged_df_test_odd, test_merged_weather_df, how='left', left_on=['Date (MM/DD/YYYY)', 'Scheduled departure hour', 'Origin Airport
                              right_on=['Date', 'Hour', 'Location'])
```

In [301…
```
merged_df_test_odd.head()
merged_df_test_even.head()
```

Out[301…

	Date (MM/DD/YYYY)	Origin Airport	Arrival Status First	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	temp_x	app_temp_x	...	wind_spd_y	wind_gust_spd_y	wind_dir_y	pres_y	s
0	2024-04-19	ORD	1	Friday	19	59	22	52	11.50	11.50	...	7.20	11.00	280	996.50	1,02
1	2024-04-19	JFK	1	Friday	14	55	16	21	15.90	15.90	...	5.60	7.40	120	1,020.00	1,02
2	2024-04-19	MCO	1	Friday	13	35	16	25	15.90	15.90	...	2.00	3.20	280	1,013.00	1,01
3	2024-04-20	ORD	1	Saturday	19	59	22	52	6.00	2.30	...	4.00	6.40	300	994.50	1,01
4	2024-04-20	JFK	1	Saturday	14	55	16	21	10.30	10.30	...	4.40	6.20	260	1,009.50	1,00

5 rows × 38 columns

Out[301…

	Date (MM/DD/YYYY)	Origin Airport	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	temp_x	app_temp_x	clouds_x	...	wind_spd_y	wind_gust_spd_y	wind_dir_y	pres_y	
0	2024-04-19	ORD	Friday	18	52	21	47	12.30	12.30	68	...	8.00	12.00	280	996.00	1,(
1	2024-04-19	JFK	Friday	13	34	14	51	16.30	16.30	84	...	5.20	7.20	110	1,020.50	1,(
2	2024-04-19	MCO	Friday	11	35	14	20	16.30	16.30	84	...	1.60	2.40	290	1,015.00	1,(
3	2024-04-20	ORD	Saturday	18	52	21	47	6.60	2.90	28	...	4.53	7.10	300	994.00	1,(
4	2024-04-20	JFK	Saturday	13	25	14	41	10.60	10.60	75	...	4.13	5.73	270	1,010.50	1,(

5 rows × 37 columns

In [302…
```
merged_df_test_odd.isna().sum()
```

Out[302…
```
Date (MM/DD/YYYY)            0
Origin Airport               0
Arrival Status First         0
Day Name                     0
Scheduled departure hour     0
Scheduled departure minute   0
Scheduled Arrival hour       0
Scheduled Arrival minute     0
temp_x                       0
app_temp_x                   0
clouds_x                     0
precip_x                     0
rh_x                         0
wind_spd_x                   0
wind_gust_spd_x              0
wind_dir_x                   0
pres_x                       0
slp_x                        0
vis_x                        0
snow_x                       0
Location_x                   0
Date_x                       0
Hour_x                       0
temp_y                       0
app_temp_y                   0
clouds_y                     0
precip_y                     0
rh_y                         0
wind_spd_y                   0
wind_gust_spd_y              0
wind_dir_y                   0
pres_y                       0
slp_y                        0
vis_y                        0
snow_y                       0
Location_y                   0
Date_y                       0
Hour_y                       0
dtype: int64
```

In [303…
```
len(merged_df_test_odd)
```

Out[303…
```
11
```

In [304…
```
merged_df_test_even.drop(['Date_x','Hour_x','Date_y', 'Hour_y','Location_y'], axis=1, inplace=True)
```

In [305…
```
merged_df_test_odd.drop(['Date_x','Hour_x','Date_y', 'Hour_y','Location_y'], axis=1, inplace=True)
```

In [306…
```
#Even rows
all_hours = pd.Series(range(24))

# Convert to categorical variable with specified categories
merged_df_test_even['Scheduled departure hour'] = pd.Categorical(merged_df_test_even['Scheduled departure hour'], categories=all_hours, ordered=True)
merged_df_test_even['Scheduled Arrival hour'] = pd.Categorical(merged_df_test_even['Scheduled Arrival hour'], categories=all_hours, ordered=True)
```

In [307…
```
all_minutes = pd.Series(range(60))

# Convert to categorical variable with specified categories
merged_df_test_even['Scheduled departure minute'] = pd.Categorical(merged_df_test_even['Scheduled departure minute'], categories=all_minutes, ordered=True)
merged_df_test_even['Scheduled Arrival minute'] = pd.Categorical(merged_df_test_even['Scheduled Arrival minute'], categories=all_minutes, ordered=True)
```

In [308…
```
date_column = merged_df_test_even['Date (MM/DD/YYYY)']

# Drop the date column before encoding
merged_df_test_even = pd.get_dummies(merged_df_test_even.drop(columns=['Date (MM/DD/YYYY)']), drop_first=True)

merged_df_test_even.head()
```

Out[308…

	temp_x	app_temp_x	clouds_x	precip_x	rh_x	wind_spd_x	wind_gust_spd_x	wind_dir_x	pres_x	slp_x	…	Scheduled Arrival minute_50	Scheduled Arrival minute_51	Scheduled Arrival minute_52	Scheduled Arrival minute_53	Scheduled Arrival minute_54	S… m
0	12.30	12.30	68	0.50	71	5.20	8.00	260	999.50	1,014.90	…	False	False	False	False	False	
1	16.30	16.30	84	0.50	55	9.20	14.20	170	996.00	1,011.20	…	False	True	False	False	False	
2	16.30	16.30	84	0.50	55	9.20	14.20	170	996.00	1,011.20	…	False	False	False	False	False	
3	6.60	2.90	28	0.00	51	6.26	9.20	280	998.50	1,014.20	…	False	False	False	False	False	
4	10.60	10.60	75	0.00	43	8.40	11.80	280	997.00	1,012.60	…	False	False	False	False	False	

5 rows × 196 columns

In [309…
```
# odd rows
all_hours = pd.Series(range(24))

# Convert to categorical variable with specified categories
merged_df_test_odd['Scheduled departure hour'] = pd.Categorical(merged_df_test_odd['Scheduled departure hour'], categories=all_hours, ordered=True)
merged_df_test_odd['Scheduled Arrival hour'] = pd.Categorical(merged_df_test_odd['Scheduled Arrival hour'], categories=all_hours, ordered=True)
```

In [310…
```
all_minutes = pd.Series(range(60))

# Convert to categorical variable with specified categories
merged_df_test_odd['Scheduled departure minute'] = pd.Categorical(merged_df_test_odd['Scheduled departure minute'], categories=all_minutes, ordered=True)
merged_df_test_odd['Scheduled Arrival minute'] = pd.Categorical(merged_df_test_odd['Scheduled Arrival minute'], categories=all_minutes, ordered=True)
```

In [311…
```
merged_df_test_odd.columns
```

Out[311…
```
Index(['Date (MM/DD/YYYY)', 'Origin Airport', 'Arrival Status First',
       'Day Name', 'Scheduled departure hour', 'Scheduled departure minute',
       'Scheduled Arrival hour', 'Scheduled Arrival minute', 'temp_x',
       'app_temp_x', 'clouds_x', 'precip_x', 'rh_x', 'wind_spd_x',
       'wind_gust_spd_x', 'wind_dir_x', 'pres_x', 'slp_x', 'vis_x', 'snow_x',
       'Location_x', 'temp_y', 'app_temp_y', 'clouds_y', 'precip_y', 'rh_y',
       'wind_spd_y', 'wind_gust_spd_y', 'wind_dir_y', 'pres_y', 'slp_y',
       'vis_y', 'snow_y'],
      dtype='object')
```

In [312…
```
merged_df_test_odd.head()
```

Out[312…

	Date (MM/DD/YYYY)	Origin Airport	Arrival Status First	Day Name	Scheduled departure hour	Scheduled departure minute	Scheduled Arrival hour	Scheduled Arrival minute	temp_x	app_temp_x	…	clouds_y	precip_y	rh_y	wind_spd_y	wind_gust_spd_y
0	2024-04-19	ORD	1	Friday	19	59	22	52	11.50	11.50	…	53	0.00	38	7.20	11.00
1	2024-04-19	JFK	1	Friday	14	55	16	21	15.90	15.90	…	89	0.00	61	5.60	7.40
2	2024-04-19	MCO	1	Friday	13	35	16	25	15.90	15.90	…	5	0.00	45	2.00	3.20
3	2024-04-20	ORD	1	Saturday	19	59	22	52	6.00	2.30	…	69	0.00	41	4.00	6.40
4	2024-04-20	JFK	1	Saturday	14	55	16	21	10.30	10.30	…	59	0.00	44	4.40	6.20

5 rows × 33 columns

In [313…
```
merged_df_test_odd.dtypes
```

Out[313...

```
Date (MM/DD/YYYY)            datetime64[ns]
Origin Airport                       object
Arrival Status First               category
Day Name                           category
Scheduled departure hour           category
Scheduled departure minute         category
Scheduled Arrival hour             category
Scheduled Arrival minute           category
temp_x                              float64
app_temp_x                          float64
clouds_x                              int64
precip_x                            float64
rh_x                                  int64
wind_spd_x                          float64
wind_gust_spd_x                     float64
wind_dir_x                            int64
pres_x                              float64
slp_x                               float64
vis_x                               float64
snow_x                                int64
Location_x                           object
temp_y                              float64
app_temp_y                          float64
clouds_y                              int64
precip_y                            float64
rh_y                                  int64
wind_spd_y                          float64
wind_gust_spd_y                     float64
wind_dir_y                            int64
pres_y                              float64
slp_y                               float64
vis_y                               float64
snow_y                                int64
dtype: object
```

In [314...

```
date_column = merged_df_test_odd['Date (MM/DD/YYYY)']

# Drop the date column before encoding
merged_df_test_odd = pd.get_dummies(merged_df_test_odd.drop(columns=['Date (MM/DD/YYYY)']), drop_first=True)

merged_df_test_odd.head()
```

Out[314...

	temp_x	app_temp_x	clouds_x	precip_x	rh_x	wind_spd_x	wind_gust_spd_x	wind_dir_x	pres_x	slp_x	...	Scheduled Arrival minute_50	Scheduled Arrival minute_51	Scheduled Arrival minute_52	Scheduled Arrival minute_53	Scheduled Arrival minute_54
0	11.50	11.50	65	0.25	72	5.60	8.60	260	1,000.00	1,015.50	...	False	False	True	False	False
1	15.90	15.90	84	0.76	58	9.60	14.40	180	996.00	1,011.20	...	False	False	False	False	False
2	15.90	15.90	84	0.76	58	9.60	14.40	180	996.00	1,011.20	...	False	False	False	False	False
3	6.00	2.30	29	0.00	53	5.73	8.40	280	999.00	1,014.70	...	False	False	True	False	False
4	10.30	10.30	63	0.00	41	8.66	12.00	280	996.50	1,012.00	...	False	False	False	False	False

5 rows × 198 columns

In [315...

```
merged_df_test_odd.rename(columns={'Arrival Status First_1': 'Arrival Status First_1.0', 'Arrival Status First_2': 'Arrival Status First_2.0'}, inplace=True)
```

In [316...

```
gb_y_train_1_pred = pipeline.predict(merged_df_test_even)
gb_y_train_1_pred
```

Out[316...

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1])
```

In [317...

```
print(merged_df_test_even.columns)
```

```
Index(['temp_x', 'app_temp_x', 'clouds_x', 'precip_x', 'rh_x', 'wind_spd_x',
       'wind_gust_spd_x', 'wind_dir_x', 'pres_x', 'slp_x',
       ...
       'Scheduled Arrival minute_50', 'Scheduled Arrival minute_51',
       'Scheduled Arrival minute_52', 'Scheduled Arrival minute_53',
       'Scheduled Arrival minute_54', 'Scheduled Arrival minute_55',
       'Scheduled Arrival minute_56', 'Scheduled Arrival minute_57',
       'Scheduled Arrival minute_58', 'Scheduled Arrival minute_59'],
      dtype='object', length=196)
```

Previous flights final predictions

In [319...

```
# Define a dictionary to map numerical labels to their meanings
label_mapping = {
    2: "late",
    1: "early",
    0: "ontime"
}

# Convert the numerical labels into their corresponding meanings
gb_y_train_1_pred_meanings = [label_mapping[label] for label in gb_y_train_1_pred]

# Print the predicted labels with their corresponding meanings
print(gb_y_train_1_pred_meanings)
```

```
['early', 'early', 'early', 'early', 'early', 'early', 'early', 'early', 'early', 'early', 'early', 'early']
```

Later flights final predictions

In [320…

```python
import pandas as pd
import itertools

# Define a dictionary to map numerical labels to their meanings
label_mapping = {
    2: "late",
    1: "early",
    0: "ontime"
}

# Define the possible combinations of values for two columns
column_combinations = [(True, False), (False, True), (False, False)]

# Initialize an empty dictionary to store predictions
predictions_dict = {}

# Iterate over each combination of column values
for comb in column_combinations:
    # Create a copy of the DataFrame
    df_copy = merged_df_test_odd.copy()

    # Set the values of the columns based on the current combination
    df_copy['Arrival Status First_1.0'] = comb[0]
    df_copy['Arrival Status First_2.0'] = comb[1]

    # Make predictions using the model pipeline
    predictions = pipeline_2.predict(df_copy)

    # Map numerical labels to their corresponding meanings
    predictions_meanings = [label_mapping[label] for label in predictions]

    # Store the predictions in the dictionary with the combination as the key
    predictions_dict[comb] = predictions_meanings

# Access predictions for each combination from the dictionary
for comb, predictions in predictions_dict.items():
    print("Predictions for combination:", comb, ":", predictions)
```

```
Predictions for combination: (True, False) : ['early', 'early', 'late', 'early', 'ontime', 'early', 'early', 'early', 'early', 'early', 'late']
Predictions for combination: (False, True) : ['early', 'early', 'ontime', 'early', 'early', 'early', 'early', 'late', 'early', 'early', 'late']
Predictions for combination: (False, False) : ['early', 'early', 'late', 'early', 'early', 'early', 'early', 'late', 'early', 'early', 'late']
```

Combination (True, False) is for when the previous flight is 'Early'.

Combination (False, True) is for when the previous flight is 'Late'.

Combination (False, False) is for when the previous flight is 'On Time'.