In [60]:

```
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from datetime import datetime,timedelta

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
```

In [61]:

```
flight_data = pd.read_csv('22_23.csv')
```

In [62]:

```
flight_data.head()
flight_data.columns
```

Out[62]:

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Scheduled Arrival Time	Actual Arrival Time	Scheduled Elapsed Time (Minutes)	Actual Elapsed Time (Minutes)	Arrival Delay (Minutes)
0	UA	01-01-2022	1,282.00	N4901U	IAD	23:10	00:01	70.00	76.00	51.00
1	UA	01-01-2023	604.00	N814UA	DEN	14:58	14:52	193.00	177.00	-6.00
2	UA	01-01-2023	2,488.00	N38458	EWR	23:14	23:15	75.00	62.00	1.00
3	UA	01-01-2023	2,645.00	N23721	ORD	23:57	23:47	107.00	100.00	-10.00
4	UA	01-02-2022	1,282.00	N4901U	IAD	23:10	23:27	70.00	64.00	17.00

Out[62]:

```
Index(['Carrier Code', 'Date (MM/DD/YYYY)', 'Flight Number', 'Tail Number',
       'Origin Airport', 'Scheduled Arrival Time', 'Actual Arrival Time',
       'Scheduled Elapsed Time (Minutes)', 'Actual Elapsed Time (Minutes)',
       'Arrival Delay (Minutes)', 'Wheels-on Time', 'Taxi-In time (Minutes)',
       'Delay Carrier (Minutes)', 'Delay Weather (Minutes)',
       'Delay National Aviation System (Minutes)', 'Delay Security (Minutes)',
       'Delay Late Aircraft Arrival (Minutes)'],
      dtype='object')
```

In [63]:

```
conditions = [
    (flight_data['Arrival Delay (Minutes)'] < -10),
    (abs(flight_data['Arrival Delay (Minutes)']) <= 10),
    (flight_data['Arrival Delay (Minutes)'] > 10) & (flight_data['Arrival Delay (Minut
    (flight_data['Arrival Delay (Minutes)'] > 30)
]

classes = ['early', 'on-time', 'late', 'severely late']

flight_data['delay_class'] = np.select(conditions, classes)

flight_data.head()
```

Out[63]:

	Carrier Code	Date (MM/DD/YYYY)	Flight Number	Tail Number	Origin Airport	Scheduled Arrival Time	Actual Arrival Time	Scheduled Elapsed Time (Minutes)	Actual Elapsed Time (Minutes)	Arrival Delay (Minutes)
0	UA	01-01-2022	1,282.00	N4901U	IAD	23:10	00:01	70.00	76.00	51.00
1	UA	01-01-2023	604.00	N814UA	DEN	14:58	14:52	193.00	177.00	-6.00
2	UA	01-01-2023	2,488.00	N38458	EWR	23:14	23:15	75.00	62.00	1.00
3	UA	01-01-2023	2,645.00	N23721	ORD	23:57	23:47	107.00	100.00	-10.00
4	UA	01-02-2022	1,282.00	N4901U	IAD	23:10	23:27	70.00	64.00	17.00

In [64]:

```
flight_data['Arrival Time'] = flight_data['Scheduled Arrival Time']
subset_data = flight_data.drop(['Carrier Code', 'Flight Number', 'Tail Number','Schedu
            'Actual Elapsed Time (Minutes)','Arrival Delay (Minutes)','Wheels-on Time
            'Delay Carrier (Minutes)','Delay Weather (Minutes)', 'Delay National Aviat
            'Delay Security (Minutes)', 'Delay Late Aircraft Arrival (Minutes)', 'Actu
            'Scheduled Arrival Time'], axis = 1)

subset_data.head()
```

Out[64]:

	Date (MM/DD/YYYY)	Origin Airport	delay_class	Arrival Time
0	01-01-2022	IAD	severely late	23:10
1	01-01-2023	DEN	on-time	14:58
2	01-01-2023	EWR	on-time	23:14
3	01-01-2023	ORD	on-time	23:57
4	01-02-2022	IAD	late	23:10

In [65]:

```
subset_data.dtypes
```

Out[65]:

```
Date (MM/DD/YYYY)     object
Origin Airport        object
delay_class           object
Arrival Time          object
dtype: object
```

In [66]:

```
subset_data.isna().sum()
```

Out[66]:

```
Date (MM/DD/YYYY)     2
Origin Airport        2
delay_class           0
Arrival Time          2
dtype: int64
```

In [67]:

```
subset_data = subset_data.dropna()
```

In [68]:

```
# Add day to dataset
for index, row in subset_data.iterrows():
    if "-" in row['Date (MM/DD/YYYY)']:
        subset_data.at[index, 'Day'] = datetime.strptime(row['Date (MM/DD/YYYY)'], "%d
    else:
        subset_data.at[index, 'Day'] = datetime.strptime(row['Date (MM/DD/YYYY)'], "%m
```

In [69]:
```
subset_data.head()
```

Out[69]:

	Date (MM/DD/YYYY)	Origin Airport	delay_class	Arrival Time	Day
0	01-01-2022	IAD	severely late	23:10	Saturday
1	01-01-2023	DEN	on-time	14:58	Sunday
2	01-01-2023	EWR	on-time	23:14	Sunday
3	01-01-2023	ORD	on-time	23:57	Sunday
4	01-02-2022	IAD	late	23:10	Tuesday

In [70]:
```
airport_dummies = pd.get_dummies(subset_data['Origin Airport'], drop_first=True)

day_dummies = pd.get_dummies(subset_data['Day'], drop_first=True)

subset_data = pd.concat([subset_data, airport_dummies, day_dummies], axis=1)
```

In [71]:
```
subset_data = subset_data.drop(['Origin Airport', 'Day'], axis = 1)
subset_data.head()
```

Out[71]:

	Date (MM/DD/YYYY)	delay_class	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	We
0	01-01-2022	severely late	23:10	0	1	0	0	1	0	0	0	
1	01-01-2023	on-time	14:58	0	0	0	0	0	1	0	0	
2	01-01-2023	on-time	23:14	1	0	0	0	0	1	0	0	
3	01-01-2023	on-time	23:57	0	0	1	0	0	1	0	0	
4	01-02-2022	late	23:10	0	1	0	0	0	0	0	1	

In [72]:
```
def standardize_time_dataset(df, col):
    for index, row in df.iterrows():
        if "-" in row[col]:
            df.at[index, col] = datetime.strptime(str(row[col]) + " " + str(row['Arriv
        else:
            df.at[index, col] = datetime.strptime(str(row[col]) + " " + str(row['Arriv

standardize_time_dataset(subset_data, 'Date (MM/DD/YYYY)')
```

In [73]:
```
subset_data.head()
```

Out[73]:

	Date (MM/DD/YYYY)	delay_class	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	We
0	2022-01-01 23:10:00	severely late	23:10	0	1	0	0	1	0	0	0	
1	2023-01-01 14:58:00	on-time	14:58	0	0	0	0	0	1	0	0	
2	2023-01-01 23:14:00	on-time	23:14	1	0	0	0	0	1	0	0	
3	2023-01-01 23:57:00	on-time	23:57	0	0	1	0	0	1	0	0	

4	2022-01-02 23:10:00	late	23:10	0	1	0	0	0	0	0	1

In [15]:

```
weather_data = pd.read_csv('./syrWeatherData/combined_csv.csv')
weather_data.head()
```

Out[15]:

	wind_spd	temp	wind_dir	weather	precip	pres	vis	clouds	dewpt	rh	wind_gust_spd	datetime	time
0	5.10	1.10	310	804	0.00	973.00	10	100	-0.70	88	5.80	2022-04-01 00:00:00	2
1	6.20	0.60	310	804	0.00	973.40	11	100	-1.20	88	7.20	2022-04-01 01:00:00	2
2	3.60	0.60	300	804	0.00	974.70	16	100	-1.20	88	3.90	2022-04-01 02:00:00	2
3	2.60	0.60	300	804	0.00	975.30	5	100	-0.70	91	2.80	2022-04-01 03:00:00	2
4	3.10	0.60	280	804	0.00	975.70	11	100	-0.70	91	3.30	2022-04-01 04:00:00	2

In [16]:

```
weather_data = weather_data.drop(columns = ['datetime', 'timestamp_utc'], axis = 1)
weather_data.head()
```

In [ ]:

```
def mergeWeather(df, wd_data, date_col):
    for index, row in df.iterrows():
        temp_date = datetime.strptime(row[date_col], '%Y-%m-%d %H:%M:%S')
        temp_dates = [temp_date - timedelta(hours=1), temp_date, temp_date + timedelta
        find_dates = list(map(lambda x: x.strftime('%Y-%m-%d %H'), temp_dates))
        count = 0
        arr = []
        for index1, row1 in wd_data.iterrows():
            if count == 3:
                break
            w_date = datetime.strptime(row1['timestamp_local'], '%Y-%m-%d %H:%M:%S').s
            if w_date in find_dates:
                count += 1
                arr.append(row1)
        if len(arr) > 0:
            res = pd.DataFrame(arr).mean()
            df.at[index, 'wind_spd'] = res['wind_spd']
            df.at[index, 'temp'] = res['temp']
            df.at[index, 'wind_dir'] = res['wind_dir']
            df.at[index, 'weather'] = res['weather']
            df.at[index, 'precip'] = res['precip']
            df.at[index, 'pres'] = res['pres']
            df.at[index, 'vis'] = res['vis']
            df.at[index, 'clouds'] = res['clouds']
            df.at[index, 'dewpt'] = res['dewpt']
            df.at[index, 'rh'] = res['rh']
            df.at[index, 'wind_gust_spd'] = res['wind_gust_spd']
        else:
            print("NOT FOUND")
            print(find_dates)
            print(temp_date)
            break
```

```
# mergeWeather(subset_data, weather_data, 'Date (MM/DD/YYYY)')
```

In [ ]:

```
# Uncomment to save
# subset_data.to_csv('mega_data.csv', index=False)
```

In [89]:

```
test_data = pd.read_csv('project csv(Apr 21-24).csv')
test_data.head()
```

Out[89]:

	Date	Day	Origin Airport	Flight Number	Arrival Time	Status (Early, On-time, Late, Severly Late)
0	4/21/2023	Friday	ORD	UA 3839	10:00 AM	NaN
1	4/21/2023	Friday	ORD	UA 3524	4:50 PM	NaN
2	4/21/2023	Friday	ORD	UA 538	9:34 PM	NaN
3	4/22/2023	Saturday	ORD	UA 3839	10:00 AM	NaN
4	4/22/2023	Saturday	ORD	UA 3524	4:50 PM	NaN

In [90]:

```
airport_dummies = pd.get_dummies(test_data['Origin Airport'], drop_first=True)

day_dummies = pd.get_dummies(test_data['Day'], drop_first=True)

test_data = pd.concat([test_data, airport_dummies, day_dummies], axis=1)

test_data = test_data.drop(['Origin Airport', 'Day', 'Status (Early, On-time, Late, Se
                        'Flight Number'], axis = 1)
```

In [91]:

```
test_data = test_data.dropna()
```

In [92]:

```
test_data.head()
```

Out[92]:

	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday
0	4/21/2023	10:00 AM	0	0	1	0	0	0
1	4/21/2023	4:50 PM	0	0	1	0	0	0
2	4/21/2023	9:34 PM	0	0	1	0	0	0
3	4/22/2023	10:00 AM	0	0	1	0	1	0
4	4/22/2023	4:50 PM	0	0	1	0	1	0

In [93]:

```
test_data['Thursday'] = 0
test_data['Tuesday'] = 0
test_data['Wednesday'] = 0
```

In [94]:

```
def standardize_time_test(df, col):
    for index, row in df.iterrows():
        if "-" in row[col]:
            df.at[index, col] = datetime.strptime(str(row[col]) + " " + str(str(row['A
        else:
            df.at[index, col] = datetime.strptime(str(row[col]) + " " + str(str(row['A
```

In [95]:

```
standardize_time_test(test_data, 'Date')
```

In [96]:

```
test_data.head()
```

Out[96]:

	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
0	2023-04-21 10:00:00	10:00 AM	0	0	1	0	0	0	0	0	0
1	2023-04-21 04:50:00	4:50 PM	0	0	1	0	0	0	0	0	0
2	2023-04-21 09:34:00	9:34 PM	0	0	1	0	0	0	0	0	0
3	2023-04-22 10:00:00	10:00 AM	0	0	1	0	1	0	0	0	0
4	2023-04-22 04:50:00	4:50 PM	0	0	1	0	1	0	0	0	0

In [97]:

```
test_data.to_csv('to_predict.csv', index=False)
```

In [98]:

```
# test_output = pd.DataFrame(model.predict(test_data), index = test_data.index, colum
# test_output = test_output.merge(test_data, left_index = True, right_index = True)
```

In [99]:

```
to_predict = pd.read_csv('to_predict.csv')
to_predict.head()
```

Out[99]:

	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday
0	2023-04-21 10:00:00	10:00 AM	0	0	1	0	0	0	0	0	0
1	2023-04-21 04:50:00	4:50 PM	0	0	1	0	0	0	0	0	0
2	2023-04-21 09:34:00	9:34 PM	0	0	1	0	0	0	0	0	0
3	2023-04-22 10:00:00	10:00 AM	0	0	1	0	1	0	0	0	0
4	2023-04-22 04:50:00	4:50 PM	0	0	1	0	1	0	0	0	0

In [100…

```
forecast_data = pd.read_csv('forecast.csv')
forecast_data.head()
```

Out[100…

	wind_spd	temp	wind_dir	weather	precip	pres	vis	clouds	dewpt	rh	wind_gust_spd	datetime	ti
0	3.98	12.00	141	804	0.00	983.00	37.09	100	2.20	51	6.20	2023-04-20 04:00:00	
1	3.58	13.20	137	804	0.00	982.00	33.98	81	4.10	54	6.20	2023-04-20 05:00:00	
2	3.55	13.50	143	804	0.00	982.00	32.69	100	4.90	56	6.57	2023-04-20	

												06:00:00
3	2.89	14.10	142	804	0.00	982.00	30.69	100	6.20	59	5.96	2023-04-20 07:00:00
4	3.46	13.30	119	803	0.00	982.50	28.80	68	6.00	61	5.31	2023-04-20 08:00:00

In [101…

```
mergeWeather(test_data, forecast_data, 'Date')
```

```
C:\Users\Ryan\AppData\Local\Temp\ipykernel_19404\127673818.py:16: FutureWarning: The d
efault value of numeric_only in DataFrame.mean is deprecated. In a future version, it
will default to False. In addition, specifying 'numeric_only=None' is deprecated. Sele
ct only valid columns or specify the value of numeric_only to silence this warning.
  res = pd.DataFrame(arr).mean()
C:\Users\Ryan\AppData\Local\Temp\ipykernel_19404\127673818.py:16: FutureWarning: The d
efault value of numeric_only in DataFrame.mean is deprecated. In a future version, it
will default to False. In addition, specifying 'numeric_only=None' is deprecated. Sele
ct only valid columns or specify the value of numeric_only to silence this warning.
  res = pd.DataFrame(arr).mean()
```

In [102…

```
test_data.head()
```

Out[102…

	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	…	temp	wind_dir	we
0	2023-04-21 10:00:00	10:00 AM	0	0	1	0	0	0	0	0	…	10.17	261.00	
1	2023-04-21 04:50:00	4:50 PM	0	0	1	0	0	0	0	0	…	10.80	283.00	
2	2023-04-21 09:34:00	9:34 PM	0	0	1	0	0	0	0	0	…	9.50	272.67	
3	2023-04-22 10:00:00	10:00 AM	0	0	1	0	1	0	0	0	…	6.93	271.33	
4	2023-04-22 04:50:00	4:50 PM	0	0	1	0	1	0	0	0	…	7.40	307.33	

5 rows × 22 columns

In [103…

```
test_data.to_csv('test_data_merged.csv', index=False)
```

In [ ]:

In [68]:

```python
# Generic inputs for most ML tasks
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from datetime import datetime,timedelta
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.datasets import make_classification
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler

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
```

In [69]:

```python
flight_data = pd.read_csv('mega_data.csv')
flight_data.head()
```

Out[69]:

	Date (MM/DD/YYYY)	delay_class	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	...	temp
0	2022-01-01 23:10:00	severely late	23:10	0	1	0	0	1	0	0	...	-3.67
1	2023-01-01 14:58:00	on-time	14:58	0	0	0	0	0	1	0	...	8.50
2	2023-01-01 23:14:00	on-time	23:14	1	0	0	0	0	1	0	...	6.10
3	2023-01-01 23:57:00	on-time	23:57	0	0	1	0	0	1	0	...	6.10
4	2022-01-02 23:10:00	late	23:10	0	1	0	0	0	0	0	...	-9.43

5 rows × 23 columns

In [70]:

```python
flight_data.isna().sum()
flight_data = flight_data.dropna()
```

Out[70]:

```
Date (MM/DD/YYYY)     0
delay_class           0
Arrival Time          0
EWR                   0
IAD                   0
ORD                   0
Monday                0
Saturday              0
Sunday                0
Thursday              0
Tuesday               0
Wednesday             0
wind_spd             38
temp                 38
wind_dir             38
weather              38
precip               38
pres                 38
vis                  38
clouds               38
dewpt                38
rh                   38
wind_gust_spd        38
dtype: int64
```

In [71]:

```python
for index, row in flight_data.iterrows():
    flight_data.at[index, 'Arrival Time'] = int(row['Arrival Time'].split(':')[0])
```

In [72]:

```python
X_train, X_test, y_train, y_test = train_test_split(flight_data.drop(columns = ['delay

sc = StandardScaler()
X_train = pd.DataFrame(sc.fit_transform(X_train),
                       columns = X_train.columns, index = X_train.index)
X_test = pd.DataFrame(sc.transform(X_test),
                      columns = X_test.columns, index = X_test.index)

X_train
X_test
y_train
y_test
```

Out[72]:

	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday	...	temp
214	0.94	-0.22	1.74	-0.81	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	-1.55
892	-0.85	-0.22	-0.58	-0.81	2.45	-0.41	-0.42	-0.41	-0.41	-0.40	...	0.94
592	0.94	-0.22	1.74	-0.81	2.45	-0.41	-0.42	-0.41	-0.41	-0.40	...	0.73
934	-0.63	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	-0.04
376	-0.85	-0.22	-0.58	-0.81	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	-0.01
...	...	...	...	...	...	...	...	...	...	...	...	...
522	0.94	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	1.04
619	-0.63	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	-0.41	2.46	-0.40	...	1.14
664	0.94	-0.22	1.74	-0.81	-0.41	-0.41	-0.42	-0.41	-0.41	2.50	...	1.19
311	0.49	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	-0.41	-0.41	2.50	...	0.42
940	0.49	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	-0.41	-0.41	-0.40	...	0.42

979 rows × 21 columns

Out[72]:

	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday	...	temp
2	0.94	4.50	-0.58	-0.81	-0.41	-0.41	2.41	-0.41	-0.41	-0.40	...	-0.55
813	0.94	-0.22	-0.58	1.24	-0.41	2.45	-0.42	-0.41	-0.41	-0.40	...	0.81
877	0.49	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	-0.41	-0.41	-0.40	...	0.30
610	-0.85	-0.22	-0.58	-0.81	-0.41	2.45	-0.42	-0.41	-0.41	-0.40	...	1.38
702	-0.63	-0.22	-0.58	1.24	-0.41	-0.41	-0.42	-0.41	2.46	-0.40	...	1.20
...	...	...	...	...	...	...	...	...	...	...	...	...
735	-0.85	-0.22	-0.58	-0.81	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	1.14
221	0.27	-0.22	-0.58	1.24	2.45	-0.41	-0.42	-0.41	-0.41	-0.40	...	-0.06
704	-0.85	-0.22	-0.58	-0.81	-0.41	-0.41	-0.42	-0.41	-0.41	2.50	...	1.24
149	0.94	-0.22	1.74	-0.81	-0.41	-0.41	-0.42	2.46	-0.41	-0.40	...	-1.58
16	0.49	-0.22	-0.58	1.24	2.45	-0.41	-0.42	-0.41	-0.41	-0.40	...	-1.07

109 rows × 21 columns

Out[72]:

```
214          on-time
892            early
592    severely late
934            early
376            early
           ...
522            early
619            early
664          on-time
311    severely late
940            early
Name: delay_class, Length: 979, dtype: object
```

Out[72]:

```
2            on-time
813            early
877          on-time
610          on-time
702            early
           ...
735             late
221            early
704          on-time
149    severely late
16           on-time
Name: delay_class, Length: 109, dtype: object
```

In [81]:

```python
# model = LogisticRegression(fit_intercept = True, solver='lbfgs', multi_class = 'aut
model = GradientBoostingClassifier(max_features=3,max_depth=8,n_estimators=10)
# max_features=3,max_depth=8,n_estimators=10
# model = RandomForestClassifier(criterion='gini')
# model = DecisionTreeClassifier()
# model = KNeighborsClassifier(n_neighbors=7)

# fit the model to the training data
model.fit(X_train, y_train)

model.score(X_train, y_train)

# make predictions on the test data
y_pred = model.predict(X_test)

# evaluate the model performance using accuracy score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(classification_report(y_test, y_pred))

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 score:", f1)
```

Out[81]:

```
▾ GradientBoostingClassifier
GradientBoostingClassifier(max_depth=8, max_features=3, n_estimators=10)
```

Out[81]:

```
0.8345250255362615
               precision    recall  f1-score   support

        early       0.31      0.16      0.21        31
         late       0.00      0.00      0.00        13
      on-time       0.47      0.86      0.61        50
severely late       0.00      0.00      0.00        15

     accuracy                           0.44       109
    macro avg       0.19      0.26      0.20       109
 weighted avg       0.30      0.44      0.34       109

Accuracy: 0.44036697247706424
Precision: 0.303275827682489
Recall: 0.44036697247706424
F1 score: 0.338325090107304
C:\Users\Ryan\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metric
s\_classification.py:1344: UndefinedMetricWarning: Precision is ill-defined and being
set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to contr
ol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Ryan\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metric
s\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Ryan\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metric
s\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
C:\Users\Ryan\AppData\Local\Programs\Python\Python310\lib\site-packages\sklearn\metric
s\_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-defin
ed and being set to 0.0 in labels with no predicted samples. Use `zero_division` param
eter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

In [97]:

```python
test_data = pd.read_csv('test_data_merged.csv')
test_data.head()
```

Out[97]:

	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	...	temp	wind_dir
0	2023-04-21 10:00:00	10:00 AM	0	0	1	0	0	0	0	0	...	10.17	261.00
1	2023-04-21 04:50:00	4:50 PM	0	0	1	0	0	0	0	0	...	10.80	283.00
2	2023-04-21 09:34:00	9:34 PM	0	0	1	0	0	0	0	0	...	9.50	272.67
3	2023-04-22 10:00:00	10:00 AM	0	0	1	0	1	0	0	0	...	6.93	271.33
4	2023-04-22 04:50:00	4:50 PM	0	0	1	0	1	0	0	0	...	7.40	307.33

5 rows × 22 columns

In [98]:

```python
for index, row in test_data.iterrows():
    test_data.at[index, 'Arrival Time'] = int(str(str(row['Arrival Time']).split(' ')
```

In [99]:

```python
final_test = test_data.drop(columns = ['Date'], axis = 1)
```

In [100…

```python
sc = StandardScaler()
final_test = pd.DataFrame(sc.fit_transform(final_test), columns = final_test.columns,
final_test.head()
```

Out[100…

	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	Tuesday	Wednesday	...	temp
0	0.92	-0.58	-0.58	1.29	-0.58	-0.58	-0.58	0.00	0.00	0.00	...	1.10
1	-0.78	-0.58	-0.58	1.29	-0.58	-0.58	-0.58	0.00	0.00	0.00	...	1.26
2	0.64	-0.58	-0.58	1.29	-0.58	-0.58	-0.58	0.00	0.00	0.00	...	0.93
3	0.92	-0.58	-0.58	1.29	-0.58	1.73	-0.58	0.00	0.00	0.00	...	0.28
4	-0.78	-0.58	-0.58	1.29	-0.58	1.73	-0.58	0.00	0.00	0.00	...	0.40

5 rows × 21 columns

In [105…

```python
test_output = pd.DataFrame(model.predict(final_test), index = final_test.index, colum
test_output = test_output.merge(test_data, left_index = True, right_index = True)
```

In [106…

```python
test_output
```

Out[106…

	pred_arrival_delay	Date	Arrival Time	EWR	IAD	ORD	Monday	Saturday	Sunday	Thursday	...	temp
0	early	2023-04-21 10:00:00	10	0	0	1	0	0	0	0	...	10.17
1	on-time	2023-04-21 04:50:00	4	0	0	1	0	0	0	0	...	10.80
2	early	2023-04-21 09:34:00	9	0	0	1	0	0	0	0	...	9.50
3	early	2023-04-22 10:00:00	10	0	0	1	0	1	0	0	...	6.93
4	on-time	2023-04-22 04:50:00	4	0	0	1	0	1	0	0	...	7.40
5	on-time	2023-04-22 09:34:00	9	0	0	1	0	1	0	0	...	6.23
6	on-time	2023-04-23 10:00:00	10	0	0	1	0	0	1	0	...	2.00
7	on-time	2023-04-23 04:55:00	4	0	0	1	0	0	1	0	...	1.67
8	early	2023-04-23 09:34:00	9	0	0	1	0	0	1	0	...	1.73
9	early	2023-04-24 10:00:00	10	0	0	1	1	0	0	0	...	5.80
10	early	2023-04-24 04:50:00	4	0	0	1	1	0	0	0	...	0.57
11	early	2023-04-24 09:34:00	9	0	0	1	1	0	0	0	...	3.87
12	on-time	2023-04-21 03:12:00	3	0	0	0	0	0	0	0	...	12.63
13	on-time	2023-04-22 03:12:00	3	0	0	0	0	1	0	0	...	7.80
14	on-time	2023-04-23 03:12:00	3	0	0	0	0	0	1	0	...	1.97
15	on-time	2023-04-24 03:12:00	3	0	0	0	1	0	0	0	...	1.07
16	on-time	2023-04-21 10:46:00	10	1	0	0	0	0	0	0	...	10.17
17	on-time	2023-04-21 11:42:00	11	1	0	0	0	0	0	0	...	10.70
18	early	2023-04-22 10:46:00	10	1	0	0	0	1	0	0	...	6.93
19	on-time	2023-04-22 11:17:00	11	1	0	0	0	1	0	0	...	7.20
20	on-time	2023-04-23 10:46:00	10	1	0	0	0	0	1	0	...	2.00
21	on-time	2023-04-23 11:42:00	11	1	0	0	0	0	1	0	...	2.80
22	on-time	2023-04-24 10:46:00	10	1	0	0	1	0	0	0	...	5.80
23	on-time	2023-04-24 11:42:00	11	1	0	0	1	0	0	0	...	7.10
24	on-time	2023-04-21 01:57:00	1	0	1	0	0	0	0	0	...	16.53
25	on-time	2023-04-21 06:59:00	6	0	1	0	0	0	0	0	...	8.90
26	on-time	2023-04-22 01:58:00	1	0	1	0	0	1	0	0	...	6.57
27	on-time	2023-04-22 06:59:00	6	0	1	0	0	1	0	0	...	5.47
28	on-time	2023-04-23 01:57:00	1	0	1	0	0	0	1	0	...	2.87
29	on-time	2023-04-23 06:59:00	6	0	1	0	0	0	1	0	...	1.70
30	on-time	2023-04-24 01:57:00	1	0	1	0	1	0	0	0	...	1.87
31	severely late	2023-04-24 06:59:00	6	0	1	0	1	0	0	0	...	-0.07

32 rows × 23 columns

In [107…

```python
test_output.to_csv('final_prediction.csv', index=False)
```

In [76]:

```python
# encoder = LabelEncoder()

# # fit the encoder to the categorical variable
# encoder.fit(flight_data['delay_class'])

# # transform the categorical variable into numerical variable
# flight_data['delay_class'] = encoder.transform(flight_data['delay_class'])
```

In [77]:

```python
# X_train, X_test, y_train, y_test = train_test_split(flight_data.drop(columns = ['de

# sc = StandardScaler()
# X_train = pd.DataFrame(sc.fit_transform(X_train),
#                        columns = X_train.columns, index = X_train.index)
# X_test = pd.DataFrame(sc.transform(X_test),
#                       columns = X_test.columns, index = X_test.index)
```

In [78]:

```python
# model = GradientBoostingRegressor(max_features=20,max_depth=20,n_estimators=20)
# model = RandomForestRegressor()

# # fit the model to the training data
# model.fit(X_train, y_train)

# model.score(X_train, y_train)

# # make predictions on the test data
# model.score(X_test, y_test)
```

In [ ]:

In [ ]: