CIS - 600 Intro To Machine Learning

Final Project

Anurima Swarup (853430232)
Nirmit Patel (992323441)
Manoj Kumar Gundapaneni (239719638)
Purushothama Rajanna (770079902)

Table of Contents

Introduction:

- As part of the final project in our Introduction to Machine Learning class, we were provided with flight data for United Airlines departing from four different airports: Newark (EWR), Washington (IAD), Denver (DEN), and Chicago (ORD), all of which were destined for Syracuse (SYR) Airport.
- Our objective was to predict the flight status for each flight, which we classified as "early" if it arrived more than 10 minutes ahead of schedule, "on-time" if it arrived within plus or minus 10 minutes of the scheduled time, "late" if it arrived more than 10 but less than 30 minutes late, and "severely late" if it was more than 30 minutes late.
- The project consisted of two stages: initial prediction and final prediction. To accomplish these, we were provided with two different datasets containing flight data for the periods of April 12-15 and April 21-24, respectively. We had no restrictions on the data we could use to train our model or on the type of machine learning model we could employ.

Initial Prediction:

- For our initial prediction we were given a .csv file with the following data : Date, Day, Origin Airport, Flight Number, Arrival Time and a blank column Status. Our goal is to predict the status of these flights.

  Approach:
  - For our initial prediction we went for a very simple approach of just using historical flight data for our train data without adding any additional data to it.
  - Our thought process behind this approach was to find just how accurate of a prediction we can get just by using the historical flight data without introducing any additional factors to our train data.
  - For this approach we decided to use a machine learning model to predict the delay based on the historical data, whatever delay we get out of it we then categorized them into late, severely late, on-time and early.

Data Gathering and Analysis:

- We obtained information from the Bureau of Transportation Statistics (BTS), a federal body that gathers and disseminates statistics on airline performance and operations.
- We began by obtaining data for departures from four places (Washington, Denver, Newark, and Chicago) and arrival statistics of Syracuse.
- Aside from the data suggested by the professor, we gathered a few more metrics (which are useful in making accurate predictions).
- We integrated arrivals and departures based on date, origin airport, and flight number after cleaning the data. This process assisted us in removing any inconsistencies in the data.
- To train the model, we hot encoded the origin airports and scaled the data.
- We used a linear regression model for training and got a score of 0.89.
- Test predictions are moderately off. Mean absolute error is 13.9 and Error ratio is 7.22
- Finally, we exported the predictions into a CSV file.

Output:

- For our train data we first read arrival and departure csv data into our dataframe and then merged it together to get our final flightData

```
arrivalData = pd.read_csv('/Users/puru/Desktop/Syracuse Courses/Sem 3/Intro to ML/Project/Dataset/Flight_data/Detailed_Statistics_Arrivals -final.csv',parse_dates=['Date (MM/DD/YYYY)','Scheduled Arr

departureData = pd.read_csv('/Users/puru/Desktop/Syracuse Courses/Sem 3/Intro to ML/Project/Dataset/Flight_data/final departures-details.csv',parse_dates=['Date (MM/DD/YYYY)','Scheduled departure ti
```

```
flightData = pd.merge(arrivalData, departureData, on = ['Date (MM/DD/YYYY)','Flight Number','Tail Number','Carrier Code'])

flightData.drop(columns=['Tail Number','Carrier Code','Source Airport','Destination Airport'],inplace=True)
flightData.dtypes
flightData
```

Python Python

	Date (MM/DD/YYYY)	Flight Number	Origin Airport	Scheduled Arrival Time	Actual Arrival Time	Arrival Delay (Minutes)	Scheduled departure time	Actual departure time	Scheduled elapsed time (Minutes)	Actual elapsed time (Minutes)	Departure delay (Minutes)	Wheels-off time	Taxi-Out time (Minutes)	Delay Carrier (Minutes)	Delay Weather (Minutes)	Delay National Aviation System (Minutes)	Delay Security (Minutes)	Delay Late Aircraft Arrival (Minutes)
0	2022-01-10	604	DEN	2023-04-11 15:09:00	2023-04-11 15:04:00	-5	2023-04-11 09:49:00	2023-04-11 09:46:00	200	198	-3	2023-04-11 10:03:00	17	0	0	0	0	0
1	2022-01-10	1917	ORD	2023-04-11 16:57:00	2023-04-11 16:54:00	-3	2023-04-11 14:10:00	2023-04-11 14:06:00	107	108	-4	2023-04-11 14:32:00	26	0	0	0	0	0
2	2022-01-10	1998	ORD	2023-04-11 21:18:00	2023-04-11 21:01:00	-17	2023-04-11 18:21:00	2023-04-11 18:15:00	117	106	-6	2023-04-11 18:35:00	20	0	0	0	0	0
3	2022-01-10	2198	IAD	2023-04-11 23:31:00	2023-04-11 23:52:00	21	2023-04-11 22:20:00	2023-04-11 22:47:00	71	65	27	2023-04-11 23:01:00	14	21	0	0	0	0
4	2022-02-10	604	DEN	2023-04-11 15:09:00	2023-04-11 14:55:00	-14	2023-04-11 09:49:00	2023-04-11 09:44:00	200	191	-5	2023-04-11 09:58:00	14	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
266	2022-12-30	1998	ORD	2023-04-11 21:07:00	2023-04-11 20:56:00	-11	2023-04-11 18:14:00	2023-04-11 18:11:00	113	105	-3	2023-04-11 18:33:00	22	0	0	0	0	0
267	2022-12-30	2488	EWR	2023-04-11 23:14:00	2023-04-11 23:07:00	-7	2023-04-11 21:59:00	2023-04-11 22:17:00	75	50	18	2023-04-11 22:31:00	14	0	0	0	0	0
268	2022-12-31	604	DEN	2023-04-11 14:58:00	2023-04-11 14:46:00	-12	2023-04-11 09:45:00	2023-04-11 09:48:00	193	178	3	2023-04-11 10:00:00	12	0	0	0	0	0
269	2022-12-31	1998	ORD	2023-04-11 21:08:00	2023-04-11 20:44:00	-24	2023-04-11 18:15:00	2023-04-11 18:06:00	113	98	-9	2023-04-11 18:22:00	16	0	0	0	0	0
270	2022-12-31	2488	EWR	2023-04-11 23:14:00	2023-04-11 00:46:00	92	2023-04-11 21:59:00	2023-04-11 23:33:00	75	73	94	2023-04-11 00:03:00	30	92	0	0	0	0

271 rows × 18 columns

Do you mind taking a quick feedback survey?

Take Survey | Remind Me later | Don't Show Again

- After removing NaN, we removed unnecessary columns, changed the data type of a few columns into datetime and perform a one hot encoding on “Origin Airport” column

```
flightData['Scheduled Arrival Time(hour)'] = flightData['Scheduled Arrival Time'].dt.hour
flightData['Scheduled Arrival Time(minutes)'] = flightData['Scheduled Arrival Time'].dt.minute

flightData['Scheduled departure time(hour)'] = flightData['Scheduled departure time'].dt.hour
flightData['Scheduled departure time(minutes)'] = flightData['Scheduled departure time'].dt.minute

flightData['Actual Arrival Time(hour)'] = flightData['Actual Arrival Time'].dt.hour
flightData['Actual Arrival Time(minutes)'] = flightData['Actual Arrival Time'].dt.minute

flightData['Actual departure time(hour)'] = flightData['Actual departure time'].dt.hour
flightData['Actual departure time(minutes)'] = flightData['Actual departure time'].dt.minute

flightData['Wheels-off time(hour)'] = flightData['Wheels-off time'].dt.hour
flightData['Wheels-off time(minutes)'] = flightData['Wheels-off time'].dt.minute

flightData.drop(columns=['Date (MM/DD/YYYY)','Scheduled Arrival Time','Actual Arrival Time','Scheduled departure time','Actual departure time','Wheels-off time'],inplace=True)
```

```
from sklearn.preprocessing import OneHotEncoder

def get_ohe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse=False, dtype='int')
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
    # If you have a newer version, replace with columns=ohe.get_feature_names_out()
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df
Final_flightData = get_ohe(flightData, 'Origin Airport')
Final_flightData.columns
```

- After that we performed a train test split and used standard scalar on the data so we can fit it into our model.

```
X_train, X_test, y_train, y_test = train_test_split(Final_flightData.drop(columns = ['Arrival Delay (Minutes)']), Final_flightData['Arrival Delay (Minutes)'], test_size=0.05, random_state=35)

X_train
X_test
y_train
y_test
```

Python

```python
if True:
    from sklearn.preprocessing import StandardScaler
    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)
```
Python

- We use Linear Regression, first we used test data to find out the score of our model.

```python
model1 = LinearRegression(fit_intercept = True)
model1.fit(X_train, y_train)

# The following gives the R-square score
model1.score(X_train, y_train)

# This is the coefficient Beta_1, ..., Beta_17
model1.coef_

# This is the coefficient Beta_0
model1.intercept_
```
Python

```
LinearRegression()

0.08861550471227952

array([  7.36501267,   5.66910053,  -2.4970423 ,  -0.0778173 ,
       -19.7207302 ,  -3.35876662])

7.155642023346306
```

```python
test_output_a = pd.DataFrame(model1.predict(X_test), index = X_test.index, columns = ['pred_Arrival Delay'])
# When extending to multiple features remove .array.reshape(-1, 1)
test_output_a.head()
```
Python

	pred_Arrival Delay
78	8.54
17	12.28
255	-6.22
23	-14.20
267	16.72

```python
test_output_a = pd.DataFrame(model1.predict(X_test), index = X_test.index, columns = ['pred_Arrival Delay'])
test_output_a = test_output_a.merge(y_test, left_index = True, right_index = True)
test_output_a.head()
mean_absolute_error = abs(test_output_a['pred_Arrival Delay'] - test_output_a['Arrival Delay (Minutes)']).mean()
print('Mean absolute error is :',mean_absolute_error)
error_ratio_a=mean_absolute_error/y_test.mean()
print('Error ratio is:',error_ratio_a)
```
Python

	pred_Arrival Delay	Arrival Delay (Minutes)
78	8.54	4
17	12.28	-4
255	-6.22	41
23	-14.20	-14
267	16.72	-7

```
Mean absolute error is : 13.939754615793914
Error ratio is: 7.22802091189314
```

- After checking the score of our model we moved on with our actual test data. We added that data into another dataframe and performed all the necessary scaling operations and one hot encoding to get the data in the same format as our test data that we used for our model.

```python
X_TEST_Actual.drop(columns='Status (Early, On-time, Late, Severly Late)',inplace=True)
X_TEST_Actual.isna().sum()
```

```
Date              22
Day               22
Origin Airport    22
Flight Number     22
Arrival Time      22
dtype: int64
```

```python
X_TEST_Actual.dropna(inplace=True)
```

```python
def get_ohe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse=False, dtype='int')
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
    # If you have a newer version, replace with columns=ohe.get_feature_names_out()
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df
X_TEST_Actual = get_ohe(X_TEST_Actual, 'Origin Airport')
```

```python
X_TEST_Actual['Actual Arrival Time(hour)'] = X_TEST_Actual['Arrival Time'].dt.hour
X_TEST_Actual['Actual Arrival Time(minutes)'] = X_TEST_Actual['Arrival Time'].dt.minute
X_TEST_Actual['Flight Number'] = X_TEST_Actual['Flight Number'].str.replace('UA', '')
X_TEST_Actual.drop(columns=['Date','Arrival Time','Day'],inplace=True)

X_TEST_Actual['Flight Number'] = X_TEST_Actual['Flight Number'].astype(int)
X_TEST_Actual.dtypes
```

```python
sc = StandardScaler()
X_TEST_Actual = pd.DataFrame(sc.fit_transform(X_TEST_Actual), columns = X_TEST_Actual.columns, index = X_TEST_Actual.index)
```

- Finally, we used our test data to predict our initial prediction, converted that prediction into the status that was asked from the question and created a csv file of our initial predictions.

```python
test_output_a = pd.DataFrame(model1.predict(X_TEST_Actual), index = X_TEST_Actual.index, columns = ['pred_Arrival Delay'])
# When extending to multiple features remove .array.reshape(-1, 1)
test_output_a.head()
```

	pred_Arrival Delay
0	41.50
1	4.89
2	-28.24
3	41.50
4	5.22

```python
def flight_status(delay):
    if delay <= -10:
        return "Early"
    elif delay >= -10 and delay <= 10:
        return "On-time"
    elif delay > 10 and delay <= 30:
        return "Late"
    else:
        return "Severely late"

test_output_a['Flight Status'] = test_output_a['pred_Arrival Delay'].apply(lambda x: flight_status(x))
```

```python
test_output_a.head()
```

	pred_Arrival Delay	Flight Status
0	41.50	Severely late
1	4.89	On-time
2	-28.24	Early
3	41.50	Severely late
4	5.22	On-time

```python
#Test_data_Final = pd.merge(Test_data, test_output_a)
test_output_a.to_csv('/Users/puru/Desktop/Syracuse Courses/Sem 3/Intro to ML/Project/Dataset/Prediction_Data.csv', index=False)
```

- According to the ground truth our model predicted 8 out of 32 correctly with the accuracy of 25%

Final Prediction:

- After our initial prediction we can see that just by using the historical data in this particular situation will not give us very accurate predictions. So, to increase the accuracy of our prediction we have to incorporate multiple datasets.

Approach:

- For our final prediction(21-24) we have considered both flight data and weather data. Because weather is one of the most common causes of flight delays, taking into account both flight data and weather data is a wise approach. This may improve our understanding of the influence of weather conditions on flight schedules and generate more accurate predictions by including weather data.
- We planned to consider the weather on both the arrival and departure city. It is critical to collect information about weather conditions in both the departure and arrival cities when utilizing weather data for prediction. This data assisted us in identifying possible difficulties that may develop during the journey, such as severe rain or thunderstorms, which may cause flight delays or cancellations.
- Our thought about using weather data is getting the reasons behind the delay and interpolate with the timings. After gathering weather data, we evaluated it to see how it will affect the flight schedule. For example, if thunderstorms are expected at the destination airport during the scheduled arrival time, the flight may be delayed or even canceled. If the weather is clear and there are no serious complications, the flight may arrive on time, if not early.
- By merging weather and flight data, we can create a more accurate forecast model that accounts for the specific aspects that might affect the flight schedule. For example, based on current weather conditions, we may utilize meteorological data to interpolate timings and predict the most likely time of arrival or departure.

Data Gathering and Analysis:

- We obtained information from the Bureau of Transportation Statistics (BTS), a federal body that gathers and disseminates statistics on airline performance and operations. We acquired insights into different elements of airline operations, such as estimated flight delays, cancellations, on-time performance by using data from the BTS. This information has been used to discover trends, patterns, and concerns affecting airline performance, as well as to construct predictive models for anticipating future performance.
- For weather data we used weather bit api to get weather from the past and also forecast data.This information can be used to create predictive models that foresee probable weather-related concerns, allowing us to take proactive efforts to avoid delays and enhance overall flight operations.
- We have used date as a key to merge flight and weather data to create a complete dataset for predicting aircraft delays.Because flight schedules and weather conditions are both time-dependent, using date as a key to merge these datasets is our analysis.
- While using weather data we considered metrics like temp, pressure, wind speed, wind direction,precipitation and others which are key factors.
- Later the extra metrics have been divided into four categories : Cloudy, rain, snow and clear. Which gives us little clarity in data.
- Weather conditions such as cloudy, rain, snow, and clear may all have an affect on flight operations in different ways. Cloudy circumstances, for example, might produce reduced visibility, resulting in aircraft delays or cancellations. Rain and snow can have an effect on airport operations, such as runway conditions and aircraft de-icing needs. Clear weather, on the other hand, may have no effect on aircraft operations.
- We have used the same approach as initial prediction while training the model but this time we have trained multiple models to get the best predictions.
- LinearRegression : 0.12
    - Mean absolute error is 31.06
    - Error ratio is 2.14
- GradientBoostingRegressor
    - Mean absolute error is 32.00
    - Error ratio is 2.20
- Lasso Regression
    - Score: 0.11

    - Mean absolute error is 30.89
- DecisionTreeRegressor
    - We have used cross validation technique to get best prediction but MAE is same for both the results
    - Mean absolute error is 28.2
- RandomForestRegressor
    - Mean absolute error is 30.89
    - 31.44

Output:

- We started by adding the flight data for our source airport and destination airport to a dataframe. In this data we had some unnecessary columns, we removed them, dropped NaN rows and merged two datasets on date, flight number and origin airport so we can get our final flight data.

```
syrData = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/flight data/syrData.csv", parse_dates = ['Date (MM/DD/YYYY)', 'Scheduled Arrival Time', 'Actual Arrival Time'])
originData = pd.read_excel("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/flight data/Merged_Departures_Apr18.xlsx", parse_dates = ['Date (MM/DD/YYYY)', 'Scheduled departure time',

syrData.head()
originData.head()
```

```
# Cleaning the data

# syrData['Scheduled Arrival Time'] = syrData['Scheduled Arrival Time'].dt.time
# syrData['Actual Arrival Time'] = syrData['Actual Arrival Time'].dt.time
syrData.drop(columns=['Carrier Code','Actual Arrival Time','Actual Elapsed Time (Minutes)','Wheels-on Time',
                      'Delay Carrier (Minutes)', 'Delay Weather (Minutes)', 'Delay National Aviation System (Minutes)',
                      'Delay Security (Minutes)', 'Delay Late Aircraft Arrival (Minutes)', 'Tail Number'], inplace = True)
syrData.dtypes
syrData.head()


# originData['Scheduled departure time'] = originData['Scheduled departure time'].dt.time
# originData['Actual departure time'] = originData['Actual departure time'].dt.time
originData.drop(columns=['Carrier Code', 'Actual departure time', 'Actual elapsed time (Minutes)', 'Wheels-off time',
                        'Delay Carrier (Minutes)', 'Delay Weather (Minutes)', 'Delay National Aviation System (Minutes)',
                        'Delay Security (Minutes)', 'Delay Late Aircraft Arrival (Minutes)', 'Scheduled elapsed time (Minutes)', 'Tail Number'], inplace = True)
originData.dtypes
originData.head()
```

```
# Looking for NAN
print('Syracuse flight NaN :')
syrData.isna().sum()

print('origin flight NaN :')
originData.isna().sum()
```

```
Syracuse flight NaN :

Date (MM/DD/YYYY)                   0
Flight Number                       0
Origin Airport                      0
Scheduled Arrival Time              0
Scheduled Elapsed Time (Minutes)    0
Arrival Delay (Minutes)             0
Taxi-In time (Minutes)              0
dtype: int64

origin flight NaN :

Date (MM/DD/YYYY)            0
Flight Number                0
Destination Airport          0
Scheduled departure time     0
Departure delay (Minutes)    0
Taxi-Out time (Minutes)      0
Source Airport               0
dtype: int64
```

```
# Merging arrival and departure data into flightData
flightData = pd.merge(syrData, originData, left_on = ['Date (MM/DD/YYYY)', 'Flight Number', 'Origin Airport'], right_on= ['Date (MM/DD/YYYY)', 'Flight Number', 'Source Airport'])
flightData.shape
flightData.head()
```

Python

(1038, 12)

	Date (MM/DD/YYYY)	Flight Number	Origin Airport	Scheduled Arrival Time	Scheduled Elapsed Time (Minutes)	Arrival Delay (Minutes)	Taxi-In time (Minutes)	Destination Airport	Scheduled departure time	Departure delay (Minutes)	Taxi-Out time (Minutes)	Source Airport
0	2022-01-01	1282	IAD	2023-04-20 23:10:00	70	51	6	SYR	2023-04-20 22:00:00	45	21	IAD
1	2022-01-02	1282	IAD	2023-04-20 23:10:00	70	17	8	SYR	2023-04-20 22:00:00	23	9	IAD
2	2022-01-03	1282	IAD	2023-04-20 23:10:00	70	21	6	SYR	2023-04-20 22:00:00	23	13	IAD
3	2022-01-04	1282	IAD	2023-04-20 23:44:00	69	135	4	SYR	2023-04-20 22:35:00	115	43	IAD
4	2022-01-05	1282	IAD	2023-04-20 23:44:00	69	-14	5	SYR	2023-04-20 22:35:00	-6	12	IAD

- After getting our flight we started working on getting weather data. We read weather data for Syracuse, Chicago, Newark, Washington D.C. and Denver into dataframes, dropped all unnecessary columns and removed NaNs to get our final weather data.

```
# Reading Weather data
syrWeather = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/Weather_2022_final/Weather_2022/syracuse_weather_2022.csv",parse_dates=['datetime', 'timestamp_utc', 'timestamp_
chiWeather = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/Weather_2022_final/Weather_2022/chicago_weather_2022.csv",parse_dates=['datetime', 'timestamp_utc', 'timestamp_lo
wasWeather = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/Weather_2022_final/Weather_2022/washingtondc_weather_2022.csv",parse_dates=['datetime', 'timestamp_utc', 'timesta
newWeather = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/Weather_2022_final/Weather_2022/newark_weather_2022.csv",parse_dates=['datetime', 'timestamp_utc', 'timestamp_lo
denWeather = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/Weather_2022_final/Weather_2022/denver_weather_2022.csv",parse_dates=['datetime', 'timestamp_utc', 'timestamp_lo

syrWeather.head()
chiWeather.head()
wasWeather.head()
newWeather.head()
denWeather.head()
```

```
# checking for NaN
syrWeather.isna().sum()
chiWeather.isna().sum()
wasWeather.isna().sum()
newWeather.isna().sum()
denWeather.isna().sum()

chiWeather.dropna(inplace= True)
wasWeather.dropna(inplace= True)
```

```
# Cleaning the weather data

denWeather['Origin Airport'] = 'DEN'
chiWeather['Origin Airport'] = 'ORD'
wasWeather['Origin Airport'] = 'IAD'
newWeather['Origin Airport'] = 'EWR'
syrWeather['Destination Airport'] = 'SYR'

syrWeather['date'] = syrWeather['timestamp_local'].dt.date
syrWeather['time'] = syrWeather['timestamp_local'].dt.time

chiWeather['date'] = chiWeather['timestamp_local'].dt.date
chiWeather['time'] = chiWeather['timestamp_local'].dt.time

wasWeather['date'] = wasWeather['timestamp_local'].dt.date
wasWeather['time'] = wasWeather['timestamp_local'].dt.time

newWeather['date'] = newWeather['timestamp_local'].dt.date
newWeather['time'] = newWeather['timestamp_local'].dt.time

denWeather['date'] = denWeather['timestamp_local'].dt.date
denWeather['time'] = denWeather['timestamp_local'].dt.time

syrWeather.drop(columns=['datetime', 'timestamp_utc', 'uv', 'solar_rad', 'clouds', 'pod', 'timestamp_local'],inplace = True)
chiWeather.drop(columns=['datetime', 'timestamp_utc', 'uv', 'solar_rad', 'clouds', 'pod', 'timestamp_local'],inplace = True)
wasWeather.drop(columns=['datetime', 'timestamp_utc', 'uv', 'solar_rad', 'clouds', 'pod', 'timestamp_local'],inplace = True)
newWeather.drop(columns=['datetime', 'timestamp_utc', 'uv', 'solar_rad', 'clouds', 'pod', 'timestamp_local'],inplace = True)
denWeather.drop(columns=['datetime', 'timestamp_utc', 'uv', 'solar_rad', 'clouds', 'pod', 'timestamp_local'],inplace = True)

syrWeather.head()
chiWeather.head()
wasWeather.head()
newWeather.head()
denWeather.head()
```

	temp	pres	dewpt	wind_spd	wind_dir	precip	snow	rh	vis	description	Destination Airport	date	time
0	5.00	996.70	4.30	1.50	140	0.00	0.00	95	16	Clear Sky	SYR	2021-12-31	19:00:00
1	3.30	996.40	3.30	2.10	110	0.00	0.00	100	14	Fog	SYR	2021-12-31	20:00:00
2	2.20	996.00	1.60	1.50	90	0.00	0.00	96	0	Haze	SYR	2021-12-31	21:00:00
3	1.70	995.70	1.70	2.60	110	0.00	0.00	100	11	Fog	SYR	2021-12-31	22:00:00
4	1.70	995.00	1.70	2.60	80	0.00	0.00	100	13	Fog	SYR	2021-12-31	23:00:00

	temp	pres	dewpt	wind_spd	wind_dir	precip	snow	rh	vis	description	Origin Airport	date	time
1	4.90	981.60	3.60	3.60	340	0.00	0.00	91	0.00	Haze	ORD	2021-12-31	19:00:00
2	4.90	981.60	3.60	3.10	330	0.00	0.00	91	0.00	Haze	ORD	2021-12-31	20:00:00
3	4.80	982.20	3.50	4.10	330	0.00	0.00	91	0.00	Haze	ORD	2021-12-31	21:00:00
4	4.50	982.90	3.30	5.10	340	0.25	0.00	92	0.00	Haze	ORD	2021-12-31	22:00:00
5	3.70	983.10	2.70	5.10	350	0.00	0.00	93	0.00	Haze	ORD	2021-12-31	23:00:00

	temp	pres	dewpt	wind_spd	wind_dir	precip	snow	rh	vis	description	Origin Airport	date	time
1	13.30	1,002.50	11.20	0.50	130	0.00	0.00	87	2.00	Haze	IAD	2021-12-31	20:00:00
2	13.10	1,002.20	11.00	2.60	160	3.00	0.00	87	2.00	Light rain	IAD	2021-12-31	21:00:00
3	13.30	1,002.20	12.20	1.50	220	0.50	0.00	93	10.00	Overcast clouds	IAD	2021-12-31	22:00:00
4	12.80	1,002.20	11.70	2.10	160	0.50	0.00	93	10.00	Overcast clouds	IAD	2021-12-31	23:00:00
5	12.20	1,001.50	10.10	1.00	150	0.00	0.00	87	10.00	Overcast clouds	IAD	2022-01-01	00:00:00

	temp	pres	dewpt	wind_spd	wind_dir	precip	snow	rh	vis	description	Origin Airport	date	time
0	10.00	1,013.10	8.30	2.10	210	0.00	0.00	89	11	Overcast clouds	EWR	2021-12-31	19:00:00
1	10.60	1,013.10	8.20	1.50	160	0.00	0.00	85	10	Overcast clouds	EWR	2021-12-31	20:00:00
2	10.00	1,012.80	8.30	2.00	160	0.00	0.00	89	10	Overcast clouds	EWR	2021-12-31	21:00:00
3	10.00	1,012.50	8.30	2.00	160	0.00	0.00	89	10	Overcast clouds	EWR	2021-12-31	22:00:00
4	10.00	1,012.10	8.30	2.10	160	0.00	0.00	89	11	Overcast clouds	EWR	2021-12-31	23:00:00

	temp	pres	dewpt	wind_spd	wind_dir	precip	snow	rh	vis	description	Origin Airport	date	time
0	-3.00	809.70	-3.00	4.60	60	0.50	10.00	100	1	Light snow	DEN	2021-12-31	17:00:00
1	-3.00	814.90	-4.10	5.10	40	0.50	10.00	92	1	Light snow	DEN	2021-12-31	18:00:00
2	-7.00	814.90	-8.10	5.70	350	1.00	25.00	92	1	Light snow	DEN	2021-12-31	19:00:00
3	-8.00	814.90	-9.10	4.10	360	1.00	30.00	92	2	Light snow	DEN	2021-12-31	20:00:00
4	-9.00	814.90	-10.10	2.10	10	1.00	30.00	92	2	Light snow	DEN	2021-12-31	21:00:00

- Finally to get our final data we merged all the data using different keys and performed some cleaning to get our final data.

```python
# Merging the weather data with flight data

flightData['Scheduled Arrival Time'] = flightData['Scheduled Arrival Time'].dt.round('H')
flightData['Scheduled departure time'] = flightData['Scheduled departure time'].dt.round('H')
flightData['Scheduled Arrival Time'] = flightData['Scheduled Arrival Time'].dt.time
flightData['Scheduled departure time'] = flightData['Scheduled departure time'].dt.time
flightData.drop(columns = ['Source Airport'], inplace = True)
flightData
flightData.dtypes
```

```python
sourceColumnNames = {'temp': 'temp_s', 'pres': 'pres_s', 'dewpt': 'dewpt_s', 'wind_spd': 'wind_spd_s', 'wind_dir': 'wind_dir_s'
                    , 'precip': 'precip_s', 'snow': 'snow_s', 'rh': 'rh_s', 'vis': 'vis_s', 'description': 'description_s'
                    , 'date': 'date_s', 'time': 'time_s', 'Origin Airport': 'Origin Airport'}
destinationColumnNames = {'temp': 'temp_d', 'pres': 'pres_d', 'dewpt': 'dewpt_d', 'wind_spd': 'wind_spd_d', 'wind_dir': 'wind_dir_d'
                    , 'precip': 'precip_d', 'snow': 'snow_d', 'rh': 'rh_d', 'vis': 'vis_d', 'description': 'description_d'
                    , 'date': 'date_d', 'time': 'time_d', 'Destination Airport': 'Destination Airport'}

syrWeather = syrWeather.rename(columns=destinationColumnNames)
denWeather = denWeather.rename(columns=sourceColumnNames)
wasWeather = wasWeather.rename(columns=sourceColumnNames)
chiWeather = chiWeather.rename(columns=sourceColumnNames)
newWeather = newWeather.rename(columns=sourceColumnNames)

syrWeather['date_d'] = pd.to_datetime(syrWeather['date_d'])
denWeather['date_s'] = pd.to_datetime(denWeather['date_s'])
wasWeather['date_s'] = pd.to_datetime(wasWeather['date_s'])
chiWeather['date_s'] = pd.to_datetime(chiWeather['date_s'])
newWeather['date_s'] = pd.to_datetime(newWeather['date_s'])

syrWeather.head()
chiWeather.head()
wasWeather.head()
newWeather.head()
denWeather.head()

originWeatherData = pd.concat([denWeather, wasWeather, chiWeather, newWeather], ignore_index=True)
originWeatherData.shape
originWeatherData
```

```python
finalData = pd.merge(flightData, syrWeather, right_on=['Destination Airport', 'date_d', 'time_d'], left_on=['Destination Airport','Date (MM/DD/YYYY)','Scheduled Arrival Time'])
finalData = pd.merge(finalData, originWeatherData, right_on=['Origin Airport', 'date_s', 'time_s'], left_on=['Origin Airport','Date (MM/DD/YYYY)','Scheduled departure time'])
finalData
```

```python
finalData.columns
finalData.drop(columns = ['Date (MM/DD/YYYY)', 'Destination Airport', 'date_d', 'time_d', 'date_s', 'time_s'], inplace = True)
finalData
```

Python

```
Index(['Date (MM/DD/YYYY)', 'Flight Number', 'Origin Airport',
       'Scheduled Arrival Time', 'Scheduled Elapsed Time (Minutes)',
       'Arrival Delay (Minutes)', 'Taxi-In time (Minutes)',
       'Destination Airport', 'Scheduled departure time',
       'Departure delay (Minutes)', 'Taxi-Out time (Minutes)', 'temp_d',
       'pres_d', 'dewpt_d', 'wind_spd_d', 'wind_dir_d', 'precip_d', 'snow_d',
       'rh_d', 'vis_d', 'description_d', 'date_d', 'time_d', 'temp_s',
       'pres_s', 'dewpt_s', 'wind_spd_s', 'wind_dir_s', 'precip_s', 'snow_s',
       'rh_s', 'vis_s', 'description_s', 'date_s', 'time_s'],
      dtype='object')
```

	Flight Number	Origin Airport	Scheduled Arrival Time	Scheduled Elapsed Time (Minutes)	Arrival Delay (Minutes)	Taxi-In time (Minutes)	Scheduled departure time	Departure delay (Minutes)	Taxi-Out time (Minutes)	temp_d	...	temp_s	pres_s	dewpt_s	wind_spd_s	wind_dir_s	precip_s	snow_s	rh_s	vis_s	description_s
0	1282	IAD	23:00:00	70	51	6	22:00:00	45	21	3.30	...	16.30	991.70	14.30	2.10	350	3.00	0.00	88	10.00	Light rain
1	1282	IAD	23:00:00	70	17	8	22:00:00	23	9	-7.20	...	9.80	1,003.90	1.70	2.10	350	0.25	0.00	57	10.00	Overcast clouds
2	1282	IAD	23:00:00	70	21	6	22:00:00	23	13	-11.10	...	-1.60	1,016.10	-7.70	2.80	65	0.00	0.00	63	10.00	Scattered clouds
3	1282	IAD	00:00:00	69	135	4	23:00:00	115	43	-11.40	...	-1.20	1,016.40	-4.20	1.00	140	0.00	0.00	80	10.00	Broken clouds
4	1282	IAD	00:00:00	69	-14	5	23:00:00	-6	12	-1.70	...	5.10	998.10	1.90	2.10	180	0.00	0.00	80	10.00	Overcast clouds
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
1028	2488	EWR	23:00:00	75	5	7	22:00:00	13	21	2.20	...	4.40	1,025.30	-4.60	3.60	230	0.00	0.00	52	16.00	Overcast clouds
1029	604	DEN	15:00:00	193	76	5	10:00:00	72	17	10.60	...	-1.00	818.00	-7.10	3.10	320	0.00	0.00	63	16.00	Broken clouds
1030	1998	ORD	21:00:00	113	21	6	18:00:00	17	30	10.00	...	13.30	988.60	9.50	7.80	190	0.00	0.00	78	16.00	Overcast clouds
1031	2488	EWR	23:00:00	75	-23	6	22:00:00	-3	15	10.00	...	4.40	1,025.00	-2.20	2.10	210	0.00	0.00	62	16.00	Scattered clouds
1032	604	DEN	15:00:00	193	17	5	10:00:00	9	26	17.20	...	-3.00	820.00	-11.00	3.60	160	0.00	0.00	54	16.00	Scattered clouds

Before getting our test train split, we performed one hot encoding on the origin airport and weather description columns.

```python
set(finalData['description_s'])
set(finalData['description_d'])

finalData['description_s'] = finalData['description_s'].replace({'Broken clouds': 'Cloudy', 'Few clouds': 'Cloudy', 'Fog': 'Cloudy', 'Haze': 'Cloudy', 'Overcast clouds': 'Cloudy',
                                                    'Scattered clouds': 'Cloudy','Heavy rain': 'Rainy', 'Light rain': 'Rainy', 'Moderate rain': 'Rainy',
                                                    'Thunderstorm with heavy rain': 'Rainy','Light snow': 'Snow', 'Mix snow/rain': 'Snow',
                                                    'Snow' : 'Snow','Clear Sky' : 'Clear'})

finalData['description_d'] = finalData['description_d'].replace({'Broken clouds': 'Cloudy', 'Few clouds': 'Cloudy', 'Fog': 'Cloudy', 'Haze': 'Cloudy', 'Overcast clouds': 'Cloudy',
                                                    'Scattered clouds': 'Cloudy','Heavy rain': 'Rainy', 'Light rain': 'Rainy', 'Moderate rain': 'Rainy',
                                                    'Thunderstorm with heavy rain': 'Rainy','Light snow': 'Snow', 'Mix snow/rain': 'Snow',
                                                    'Snow' : 'Snow', 'Flurries' : 'Snow','Clear Sky' : 'Clear','Moderate rain' : 'Rainy', 'Heavy snow' : 'Snow'})

set(finalData['description_s'])
set(finalData['description_d'])
```

```python
from sklearn.preprocessing import OneHotEncoder
                        (parameter) drop: ArrayLike | None
def getOhe(df, col):
    ohe = OneHotEncoder(drop='first', handle_unknown='error', sparse=False, dtype='int')
    ohe.fit(df[[col]])
    temp_df = pd.DataFrame(data=ohe.transform(df[[col]]), columns=ohe.get_feature_names_out())
    # If you have a newer version, replace with columns=ohe.get_feature_names_out()
    df.drop(columns=[col], axis=1, inplace=True)
    df = pd.concat([df.reset_index(drop=True), temp_df], axis=1)
    return df
```

```python
subsetData = getOhe(finalData, 'description_d')
subsetData = getOhe(subsetData, 'description_s')
subsetData = getOhe(subsetData, 'Origin Airport')
subsetData.columns
subsetData.drop(columns = ['Taxi-In time (Minutes)', 'Departure delay (Minutes)', 'Taxi-Out time (Minutes)','Scheduled Arrival Time', 'Scheduled departure time'], inplace= True)
# subsetData['Scheduled Arrival Time'] = subsetData['Scheduled Arrival Time'].apply(lambda x: int(x.strftime('%H%M%S')))
# subsetData['Scheduled departure time'] = subsetData['Scheduled departure time'].apply(lambda x: int(x.strftime('%H%M%S')))

subsetData.dtypes
```

To find the scores of the ML model we split the data into training and testing data and used a standard scaler to scale the data.

```python
# Scaling the data
xTrain, xTest, yTrain, yTest = train_test_split(subsetData.drop(columns = ['Arrival Delay (Minutes)']), subsetData['Arrival Delay (Minutes)'], test_size=0.20, random_state=40)
xTrain
xTest
yTrain
yTest
```

```python
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
xTrain = pd.DataFrame(sc.fit_transform(xTrain), columns = xTrain.columns, index = xTrain.index)
xTest = pd.DataFrame(sc.transform(xTest), columns = xTest.columns, index = xTest.index)
xTrain
xTest
yTrain
yTest
```

	Flight Number	Scheduled Elapsed Time (Minutes)	temp_d	pres_d	dewpt_d	wind_spd_d	wind_dir_d	precip_d	snow_d	rh_d	...	vis_s	description_d_Cloudy	description_d_Rainy	description_d_Snow	description_s_Cloudy	description_s_Ra
502	-1.07	1.41	1.30	-1.36	1.47	1.75	-0.14	0.34	-0.11	0.10	...	0.56	0.42	-0.29	-0.16	0.57	-
957	0.86	-0.32	-0.96	0.86	-1.07	0.90	0.78	-0.27	-0.11	-0.35	...	0.56	0.42	-0.29	-0.16	0.57	-
262	0.90	-1.15	0.33	-0.20	0.23	0.69	-0.38	-0.27	-0.11	-0.40	...	-1.14	0.42	-0.29	-0.16	0.57	-
341	0.94	-0.34	1.25	-0.39	1.02	0.00	1.13	-0.27	-0.11	-0.68	...	-1.14	0.42	-0.29	-0.16	0.57	-
215	0.90	-1.15	-0.86	-1.18	-0.74	1.11	0.78	-0.27	-0.11	0.21	...	-1.14	0.42	-0.29	-0.16	-1.74	-
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
626	1.72	-1.09	0.69	-0.20	0.97	-0.72	0.20	-0.27	-0.11	0.43	...	0.56	0.42	-0.29	-0.16	0.57	-
1016	1.54	-1.05	-2.14	-1.73	-2.43	2.65	0.09	-0.27	-0.11	-0.68	...	0.56	0.42	-0.29	-0.16	0.57	-
165	0.69	-1.17	-0.76	-1.40	-0.27	-0.21	-1.88	0.95	-0.11	1.27	...	-3.41	-2.39	3.48	-0.16	0.57	-
7	-0.13	-1.17	-2.12	1.41	-1.80	0.69	0.96	-0.27	-0.11	1.05	...	-1.14	0.42	-0.29	-0.16	-1.74	-
219	-0.46	1.39	0.27	0.35	-0.95	1.11	0.78	-0.27	-0.11	-2.08	...	0.56	0.42	-0.29	-0.16	0.57	-

826 rows × 29 columns

	Flight Number	Scheduled Elapsed Time (Minutes)	temp_d	pres_d	dewpt_d	wind_spd_d	wind_dir_d	precip_d	snow_d	rh_d	...	vis_s	description_d_Cloudy	description_d_Rainy	description_d_Snow	description_s_Cloudy	description_s_Rai
466	1.72	-1.09	0.58	-0.06	0.75	-1.40	0.72	-0.27	-0.11	0.16	...	0.56	0.42	-0.29	-0.16	0.57	-0.
643	-1.07	1.39	1.40	-0.06	0.97	0.43	1.13	-0.27	-0.11	-1.02	...	0.56	0.42	-0.29	-0.16	0.57	-0.
937	-1.07	1.31	-0.60	-0.71	-0.39	1.33	0.78	-0.27	-0.11	0.43	...	0.56	0.42	-0.29	-0.16	-1.74	-0.
498	-1.07	1.41	1.62	-0.20	1.29	-0.85	-2.11	-0.27	-0.11	-0.85	...	0.56	0.42	-0.29	-0.16	0.57	-0.
285	-0.46	1.39	-0.09	-0.25	0.56	-0.64	1.59	0.04	-0.11	1.66	...	-2.56	0.42	-0.29	-0.16	0.57	-0.
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
239	-0.46	1.39	-0.91	-1.54	-0.62	1.33	0.32	-0.27	-0.11	0.71	...	0.56	0.42	-0.29	-0.16	0.57	-0.
1	-0.13	-1.15	-1.89	0.63	-1.69	-0.42	1.48	-0.27	-0.11	0.60	...	-1.14	0.42	-0.29	-0.16	0.57	-0.
15	-0.13	-1.17	-2.94	2.11	-2.85	-1.19	-0.95	-0.27	-0.11	0.49	...	-3.41	0.42	-0.29	-0.16	-1.74	3.
694	-1.07	1.43	0.84	0.12	1.27	-0.21	-2.00	-0.27	-0.11	0.77	...	0.56	0.42	-0.29	-0.16	0.57	-0.
980	0.86	-0.28	-1.22	0.35	-0.66	-0.72	0.26	-0.27	-0.11	1.66	...	0.56	0.42	-0.29	-0.16	0.57	-0.

207 rows × 29 columns

```
502      8
957     -5
262     77
341     56
215      3
       ...
626      2
1016   380
165      6
7       -6
219     -6
Name: Arrival Delay (Minutes), Length: 826, dtype: int64
```

```
466   -9
643   28
937    4
498  -19
285   -6
      ..
239   -6
1     17
15     0
694   -8
980  -18
Name: Arrival Delay (Minutes), Length: 207, dtype: int64
```

- After getting the data we fitted this data into multiple models to find out the best score and accuracy to get our final predictions.
- According to our data and scores we got the decision tree regressor that gave us the best result in terms of accuracy and score vise.

- To get our final prediction we added test data into a dataframe, cleaned the data up and got it into the same format as our test data. We used the decision tree regressor on our train data to get our final predictions and put it into a csv file.

```
# Reading Test Data
finalTesttData = pd.read_csv("C:/Users/nirmi/Desktop/SU_SEM3/IntroToML/Project/Data sets/project_test_data.csv")
finalTesttData.head()
```

```
finalTesttData['description_d'] = finalTesttData['description_d'].replace({'Light shower rain' : 'Rainy'})
finalTesttData
```

```
testSubsetData = getOhe(finalTesttData, 'description_d')
testSubsetData = getOhe(testSubsetData, 'description_s')
testSubsetData = getOhe(testSubsetData, 'Origin Airport')
testSubsetData.drop(columns=['Scheduled Arrival Time', 'Scheduled departure time'],inplace=True)
testSubsetData
```

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
testData = pd.DataFrame(sc.fit_transform(testSubsetData), columns = testSubsetData.columns,
                        index = testSubsetData.index)
testData
```

```
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score

modelDT = DecisionTreeRegressor(max_depth=5, min_samples_split=6, min_samples_leaf=2, random_state=10)
modelDT.fit(xTrain, yTrain)
modelDT.score(xTrain, yTrain)

testOutputDT = pd.DataFrame(modelDT.predict(xTest), index=xTest.index, columns=['predArrivalDelay'])
testOutputDT = testOutputDT.merge(yTest, left_index=True, right_index=True)
mean_absolute_error = abs(testOutputDT['predArrivalDelay'] - testOutputDT['Arrival Delay (Minutes)']).mean()
print('Decision Tree Regression - Mean absolute error is ')
print(mean_absolute_error)
```

```
DecisionTreeRegressor(max_depth=5, min_samples_leaf=2, min_samples_split=6,
                      random_state=10)

0.44291032914796336

Decision Tree Regression - Mean absolute error is
28.41618340194153
```

```
test_output_d = pd.DataFrame(modelDT.predict(testData), index = testData.index, columns = ['pred_Arrival Delay'])
# When extending to multiple features remove .array.reshape(-1, 1)
test_output_d.head()
```

```
c:\Users\nirmi\anaconda3\lib\site-packages\sklearn\base.py:493: FutureWarning: The feature names should match those that were passed during fit. Starting version 1.2, an error will be raised.
Feature names unseen at fit time:
- Scheduled Elapsed Time
Feature names seen at fit time, yet now missing:
- Scheduled Elapsed Time (Minutes)

  warnings.warn(message, FutureWarning)
```

	pred_Arrival Delay
0	5.88
1	2.70
2	5.23
3	5.88
4	2.70

```python
def flight_status(delay):
    if delay < -10:
        return "Early"
    elif delay >= -10 and delay < 10:
        return "On-time"
    elif delay > 10 and delay <= 30:
        return "Late"
    else:
        return "Severely late"

test_output_d['Flight Status'] = test_output_d['pred_Arrival Delay'].apply(lambda x: flight_status(x))

test_output_d
```

- For our final prediction we got 11 out of 32 predictions correct, that is 34.48% accuracy, that is a clear improvement over our initial prediction that was just 20%.

Conclusion:

This project gave us an insight into the power of predictive analysis using Machine Learning algorithms like regression. During the course of this project, we employed all our knowledge from our coursework including scaling the data so that they are in similar ranges, handling categorical data as a feature, using one-hot encoding for such features and employing different models and fine tuning their parameters to get the desired results.We were also able to witness the advantages and limitations of using certain models and their effects on accuracy.

The future scope for this project is to analyze other features apart from weather that affect the flight delays like Air Traffic Control delays, technical issues, Security concerns, etc. and predict these flight delays more accurately.

Reference:

- Flight data: https://www.transtats.bts.gov/ontime/
- Weather data: https://www.weatherbit.io/account/create
- SKlearn: https://scikit-learn.org/stable/