This code is belong to Hong Phuc Pham, please do not copy and use for your university assignments.

Project brief

This project will observer the US domestic flight data in 2018. The data is published by US Department of Transport and can be also found by the following on Kaggle: https://www.kaggle.com/yuanyuwendymu/airline-delay-and-cancellation-data-2009-2018?select=2018.csv

or Google Drive folder: https://drive.google.com/file/d/1XVbLb7bpHjj_l7U8IZh8EDDhK3eeCeGh/view?usp=sharing

This report will follow machine learning pipelines to predict the delay time of flights to Texas operated by Delta Air Lines Inc.

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True)
```

Dataset

This dataset has 28 columns, where

- FL_DATE: Flight Date (**yyyymmdd**)
- OP_CARRIER: Operation Carrier - Supported lookup table: https://www.transtats.bts.gov/FieldInfo.asp?Svryq_Qr5p=h0v37r%FDPn44vr4%FDP1qr.%FDjur0%FD6ur%FD5nzr%FDp1qr%FDun5%FDorr0%FD75rq%FDoB%FDz7y6v2yr%FDpn44vr45%FP%FDn%FD07zr4vp%FD57ssvA%FDv5%FD75rq%FDs14%FDrn4yvr4%FD75r45%FP%FDs14%FDrAnz2yr%FP%FDcN%FP%FDcN%FLE%FM%FP%FDcN%FLF%FM.%FDh5r%FD6uv5%FDsvryq%FDs14%FDn0nyB5v5%FDnp4155%FDn%FD4n0tr%FD1s%FDBrn45.&Svryq_gB2r=Pun4&Y11x72_gnoyr=Y_haVdhR_PNeeVRef&gnoyr_VQ=FGK&flf_gnoyr_anzr=g_bagVZR_ZNeXRgVaT&fB5_Svryq_anzr=bc_haVdhR_PNeeVRe
- OP_CARRIER_FL_NUM: Operation Carrier Flight Number
- ORIGIN: Origin Airport - Supported lookup table: https://www.transtats.bts.gov/FieldInfo.asp?Svryq_Qr5p=b4vtv0%FDNv42146&Svryq_gB2r=Pun4&Y11x72_gnoyr=Y_NVecbeg&gnoyr_VQ=FGK&flf_gnoyr_anzr=g_bagVZR_ZNeXRgVaT&fB5_Svryq_anzr=beVTVa
- DEST: Destination Airport - Supported lookup table: https://www.transtats.bts.gov/FieldInfo.asp?Svryq_Qr5p=Qr56v0n6v10%FDNv42146&Svryq_gB2r=Pun4&Y11x72_gnoyr=Y_NVecbeg&gnoyr_VQ=FGK&flf_gnoyr_anzr=g_bagVZR_ZNeXRgVaT&fB5_Svryq_anzr=QRfg
- CRS_DEP_TIME: CRS Departure Time (local time: **hhmm**)
- DEP_TIME: Actual Departure Time (local time: **hhmm**)
- DEP_DELAY: Difference in minutes between scheduled and actual departure time. *Early departures show **negative** numbers*.
- TAXI_OUT: Taxi Out Time, in **Minutes**
- WHEELS_OFF: Wheels Off Time (local time: **hhmm**)
- WHEELS_ON: Wheels On Time (local time: **hhmm**)
- TAXI_IN: Taxi In Time, in **Minutes**
- CRS_ARR_TIME: CRS Arrival Time (local time: **hhmm**)
- ARR_TIME: Actual Arrival Time (local time: **hhmm**)
- ARR_DELAY: Difference in minutes between scheduled and actual arrival time. *Early departures show **negative** numbers*.
- CANCELED: Canceled Flight Indicator (**1=Yes**)
- CANCELLATION_CODE: Specifies The Reason For Cancellation - Supported lookup table: https://www.transtats.bts.gov/FieldInfo.asp?Svryq_Qr5p=f2rpvsvr5%FDgur%FDern510%FDS14%FDPn0pryyn6v10&Svryq_gB2r=Pun4&Y11x72_gnoyr=Y_PNaPRYYNgVba&gnoyr_VQ=FGK&flf_gnoyr_anzr=g_bagVZR_ZNeXRgVaT&fB5_Svryq_anzr=PNaPRYYNgVba_PbQR
- DIVERTED: Diverted Flight Indicator (**1=Yes**)
- CRS_ELAPSED_TIME: CRS Elapsed Time of Flight, in **Minutes**
- ACTUAL_ELAPSED_TIME: Elapsed Time of Flight, in **Minutes**
- AIR_TIME: Flight Time,in **Minutes**
- DISTANCE: Distance between airports **(miles)**
- CARRIER_DELAY: Carrier Delay, in **Minutes**
- WEATHER_DELAY: Weather Delay, in **Minutes**
- NAS_DELAY: National Air System Delay, in **Minutes**
- SECURITY_DELAY: Security Delay, in **Minutes**
- LATE_AIRCRAFT_DELAY: Late Aircraft Delay, in **Minutes**
- Unnammed: 27 (not included in US Department of Transport document)

Dataset contains about 7 million rows.

Different data types are in delay/ canceled dataset include: int, float, string, datetime.

Challenges

Flight dataset is not a stand-alone sheet, it need to merge and connect with other dataset provided by US Department of Transport to make the data set more meaning. Likes: ORIGIN, DES

There are various datatype in this dataset.

Technical Summary

Libraries use:

```
- Pandas
- Plotly
- Seaborn
- Sci-kit learn
- Numpy
- Joblib
```

Solution

Cleaning data

As the main data set has unclear data need to take care.

Many columns containing float data types where they should be integer.

Some integer columns are used to represent the time and need to formated to datetime object.

Some empty cells need to fill up. As they are about the minute counting therefore any Nan cells can be filled by 0 values.

Because, this dataset is big and with the limited computing resource, only the flight to Texas operated by Delta Airline Co. will be used.

Pseudo Code

1. Getting the Carrier operator code and Texas destination code
2. Get the dataset with the wanted column and records that has Texas code.
3. Replace Nan value to 0, set datatype to float and remove duplicated
4. Check the correlation between features and target. Then use PCA to reduce the demensional
5. Data segregation
6. Training
7. Hyperparameter Tuning and Evaluating
8. Visualise the predict values and actual values
9. Save model

Solution Code

Problem Definition

This data set contain various features about the flight. Those features will be used to predict the delayed time of that flight which is an arrival_delay. The negative values mean that airplans arrive early and positive values for late or delay.

Data Ingestion

```
# Import libraries
import numpy as np
import pandas as pd
import datetime as dt
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.graph_objects as go
import plotly.express as px
import joblib

from numpy import array
from sklearn.preprocessing import MinMaxScaler, PolynomialFeatures
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, KFold, StratifiedKFold, cross_val_score, crc
from sklearn.pipeline import make_pipeline
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
# Load dataset
op_df =  pd.read_csv('/content/drive/MyDrive/Delay_Prediction/op_carrier.csv')
# Delta Air Lines Inc.
# Get operational carrier
op_df = op_df[op_df['Description'].str.contains('Delta Air Lines Inc.')]
op_code = [str(x) for x in op_df['Code']][0]

print(op_code)
```

```
     DL
```

```
# Load dataset
des_df =  pd.read_csv('/content/drive/MyDrive/Delay_Prediction/destination.csv')

# Get Texas destination
tx_df = des_df[des_df['Description'].str.contains('TX')]
tx_df
```

	Code	Description
17	1TX	Dumas, TX: Moore County
24	2TX	Brenham, TX: Brenham Municipal
28	3TX	Lancaster, TX: Lancaster Regional
30	4TX	Lajitas, TX: Lajitas International
31	5TX	Edinburg, TX: South Texas International at Edi...
...	...	...
5906	VWH	Midland/Odessa, TX: Odessa Schlemeyer Field
5920	VWX	Hondo, TX: South Texas Regional at Hondo
5934	VZH	Mineola, TX: Mineola Wisener Field
6005	WHT	Wharton, TX: Wharton Regional
6187	XXD	Clarendon, TX: Smiley Johnson Municipal/Bass F...

158 rows × 2 columns

```
# Load main dataset
df = pd.read_csv('/content/drive/MyDrive/Delay_Prediction/2018_flight_data.csv')

# Slicing data frame to not take the redundant column 'Unnamed: 27'

# df.query('OP_CARRIER == "DL"', inplace = True)
df = df.loc[df['OP_CARRIER'] == op_code]
df = df[['DEST','DEP_DELAY','TAXI_OUT','TAXI_IN','DIVERTED','ACTUAL_ELAPSED_TIME',
         'AIR_TIME','DISTANCE','CARRIER_DELAY','WEATHER_DELAY',
         'NAS_DELAY','SECURITY_DELAY','LATE_AIRCRAFT_DELAY','ARR_DELAY']]
df.head(5)
```

	DEST	DEP_DELAY	TAXI_OUT	TAXI_IN	DIVERTED	ACTUAL_ELAPSED_TIME	AIR_TIME	DISTAN
16112	TPA	-4.0	27.0	2.0	0.0	187.0	158.0	100
16113	JFK	0.0	29.0	10.0	0.0	257.0	218.0	199
16114	SLC	-2.0	24.0	4.0	0.0	111.0	83.0	59
16115	LAX	-3.0	26.0	8.0	0.0	119.0	85.0	59
16116	MCI	34.0	13.0	7.0	0.0	80.0	60.0	39

```
# Filter to get wanted flights
df = df.loc[df['DEST'].isin(tx_df['Code'])]

# Checking shape
print(df.shape)
```

```
(26128, 14)
```

```
# Refine dataset
df = df.iloc[::,1:]
```

Data Preparation - Preprocessing

Double-click (or enter) to edit

```
# Replace Nan to 0 value.
delay_df = df.replace(np.NaN, 0)

# Covert datatype to float.
delay_df.astype('float')

# Drop duplicates
delay_df.drop_duplicates()
```

	DEP_DELAY	TAXI_OUT	TAXI_IN	DIVERTED	ACTUAL_ELAPSED_TIME	AIR_TIME	DISTANCE
16124	17.0	22.0	11.0	0.0	146.0	113.0	731.0
16126	-3.0	15.0	19.0	0.0	137.0	103.0	689.0
16276	-2.0	10.0	8.0	0.0	158.0	140.0	1242.0
16321	-3.0	26.0	8.0	0.0	153.0	119.0	989.0
16360	21.0	15.0	12.0	0.0	152.0	125.0	989.0
...	...	...	...	...	...	...	...
7199874	-5.0	15.0	7.0	0.0	135.0	113.0	696.0
7199879	-3.0	10.0	4.0	0.0	124.0	110.0	696.0
7199984	-5.0	19.0	5.0	0.0	160.0	136.0	874.0
7199986	-5.0	12.0	8.0	0.0	168.0	148.0	1235.0
7200050	-2.0	29.0	7.0	0.0	155.0	119.0	731.0

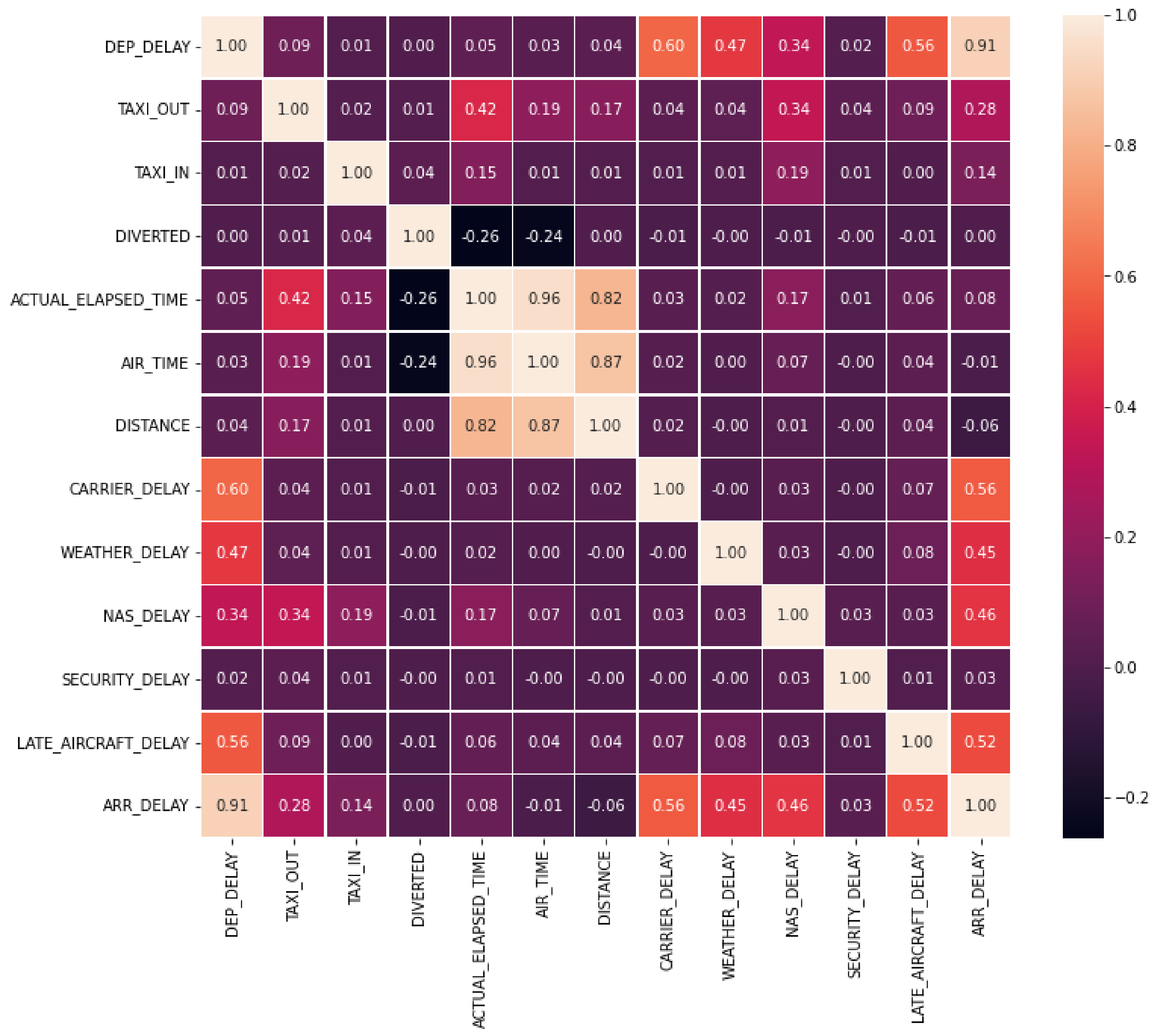
26035 rows × 13 columns

Data Segregation

```
# delay_df.reset_index(drop = True, inplace = True)
X = delay_df.iloc[::, :-1:]
y = delay_df['ARR_DELAY']
```

```
# Checking the correlation
correlation_matrix = delay_df.corr()
```

```
# Plotting the correlation map
plt.figure(figsize=(12,10))
sns.heatmap(data = correlation_matrix, annot = True, fmt = '.2f', linewidths = .5)
plt.show()
```



Base on the correlation map, here are fetures has most correlations: DEP_DELAY, CARRIER_DELAY, WEATHER_DELAY, NAS_DELAY, LATE_AIRCRAFT_DELAY

```
# Features selection
X = X[['DEP_DELAY', 'CARRIER_DELAY', 'WEATHER_DELAY', 'NAS_DELAY', 'LATE_AIRCRAFT_DELAY']]


# Define normalisation using MinMaxScaler
scaler = MinMaxScaler()


# Define PCA
pca = PCA(n_components = 3, whiten = True, random_state = 42)


#Define Regressor
regressor = LinearRegression()


# Make pipeline
Linear_model = make_pipeline(MinMaxScaler(), pca , regressor)


# Kfork cross validation check
kf = KFold(n_splits = 7, shuffle = True, random_state = 42)
scores = cross_val_score(Linear_model, X, y, cv = kf, scoring='neg_mean_squared_error')
print(scores)
print('\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
```

```
[-196.62324581 -142.39865432 -155.38377116 -192.9126971  -146.12986275
 -139.92322124 -148.68012616]

Cross-Validation accuracy: -160.293 +/- 22.292
```

```
# Train and test set split
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 42)
```

Training

```
Linear_model.fit(X_train, y_train)

y_pred = Linear_model.predict(X_test)

# Getting baseline and check with naive RSME baseline
baseline = np.mean(y)
y_baseline = np.repeat(baseline, len(y_test))

naive_RSME = mean_squared_error(y_test, y_baseline)
naive_RSME = np.sqrt(naive_RSME)

RMSE_og_training = np.sqrt(mean_squared_error(y_train, Linear_model.predict(X_train)))
RMSE_og_test=np.sqrt(mean_squared_error(y_test, y_pred))

print('The naive RMSE baseline is ', naive_RSME)
print('The model performance in training (without tuning) is ', RMSE_og_training)
print('The model performance in testing (without tuning) is ', RMSE_og_test)
```

```
     The naive RMSE baseline is  37.61711701501969
     The model performance in training (without tuning) is  13.449726361344204
     The model performance in testing (without tuning) is  13.977664551129523
```

Hyperparameter Tuning

```
# Hyperparameter tuning
#Parameter to calibrate
n_components = [1, 2, 3, 4, 5]
fit_intercept = [True, False]

param_grid = {'linearregression__fit_intercept': fit_intercept,
               'pca__n_components': n_components}


#Searching for the optimal from param_grid
param_search = GridSearchCV(Linear_model, param_grid, cv = 3, n_jobs = -1)

#Find the optimum values
param_search.fit(X_train, y_train)

print('best parameters', param_search.best_params_)
```

```
     best parameters {'linearregression__fit_intercept': True, 'pca__n_components': 5}
```

Model Evaluation

```
# Update the model
Linear_model = param_search.best_estimator_
scores = cross_val_score(Linear_model, X, y, cv = kf, scoring='neg_mean_squared_error')
print(scores)
print('\nCross-Validation accuracy: %.3f +/- %.3f' %(np.mean(scores), np.std(scores)))
```

```
     [-138.49112983 -124.23474734 -125.52824366 -135.71213168 -126.34789435
      -130.04874204 -129.53764391]

     Cross-Validation accuracy: -129.986 +/- 4.950
```

```
# Train model
Linear_model.fit(X_train, y_train)

# Checking with baseline
RMSE_tunned_train = np.sqrt(mean_squared_error(y_train, Linear_model.predict(X_train)))
RMSE_tunned_test = np.sqrt(mean_squared_error(y_test, Linear_model.predict((X_test))))

print('The naive RMSE baseline is ', naive_RSME)
print('The model performance in training (without tuning) is', RMSE_og_training)
print('The model performance in testing (without tuning) is', RMSE_og_test)
print('The model performance in training is', RMSE_tunned_train)
print('The model performance in testing is', RMSE_tunned_test)
```

```
     The naive RMSE baseline is  37.61711701501969
     The model performance in training (without tuning) is 13.449726361344204
```

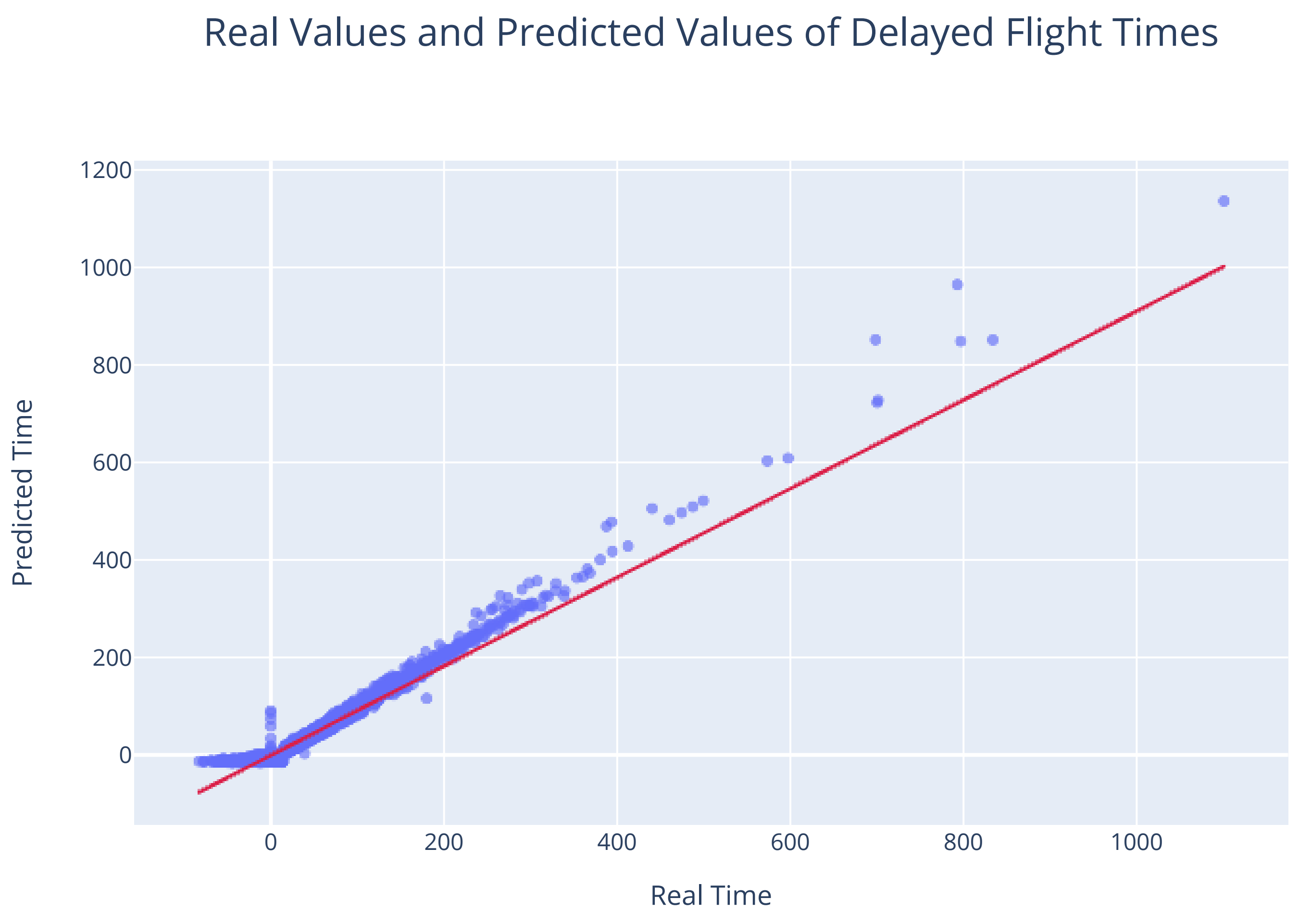
```
The model performance in testing (without tuning) is 13.977664551129523
The model performance in training is 11.32493749839382
The model performance in testing is 11.521269346774403
```

Visualising the Polynominal Regression Result

```
# Get the predict values
y_predict =  Linear_model.predict(X)


# Plotting
fig = px.scatter(
  x = y, y = y_predict, opacity = 0.65,
  trendline = 'ols', trendline_color_override = '#DB1F48',
  title = {
    'text': 'Real Values and Predicted Values of Delayed Flight Times',
    'y': 0.95,
    'x': 0.5,
    'xanchor': 'center',
    'yanchor': 'top',
    'font_size': 20 },
  labels = dict(x = "Real Time", y = "Predicted Time"))

fig.show()
```



Saving model

```
filename = 'linear_reg_model.sav'
# pickle.dump(regressor, open(filename, 'wb'))
joblib.dump(regressor, filename)

     ['linear_reg_model.sav']
```