Objectives

1. To apply feature engineering on real world datasets to remove unwanted data and outliers and also extract useful data from the given data
2. To use multiple machine learning algorithms to predict the price of flights.
3. To predict the accuracy of machine algorithms used and find the most accurate algorithms.

Software Used

1. Anaconda Navigator
2. Python
3. Jupyter Notebook

Libraries Used

1. Pandas
2. Numpy
3. Sklearn
4. Matplotlib
5. Label Encoder
6. Datetime
7. Seaborn

Procedures

1. **To apply feature engineering on real world datasets to remove unwanted data and outliers and also extract useful data from the given data:**

We will apply feature engineering on the real world dataset provided to us. We remove the unwanted columns and also the rows where “Route 1==NULL” as we cannot have a flight without it. We will then assign value 0 throughout the dataset to all the NULL values. We will then use the departure date to extract the month and day from the given date. After this we will check whether the given date is a weekend or not. We will add all this column to our training and test dataset as it will help us in predicting the flight price because the flight price is usually higher on weekends.

We will use label encoder to encode all the non integer values to integer values like “Route” and “Is Weekend”. Then we will use Standard Scaler to scale all the values . Standard Scaler means to standardize features by removing the scaling to unit variance. The standard score of a sample x is calculated as:

$$z = (x - u) / s$$

where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

2. **To use multiple machine learning algorithms to predict the price of flights:**

We will apply three different machine learning algorithms to predict the price of the flights and then check the accuracy of different algorithms. The algorithms used are namely, linear regression , random forest and support vector regression.

Linear Regression: It is a machine learning algorithm based on supervised learning. It performs a regression task. Regression models a target prediction value based on independent variables.
Linear regression performs the task to predict a dependent variable value (y) based on a given independent variable (x). So, this regression technique finds out a linear relationship between x (input) and y(output). Hence, the name is Linear Regression.

Random Forest: It consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. It is an ensemble method which is better than a single decision tree because it reduces the over-fitting by averaging the result. Random forests work well for a large range of data items and also have less variance.

Support Vector Regression (SVR): In simple regression we try to minimise the error rate. While in SVR we try to fit the error within a certain threshold.
See in fig 1 how all the points are within the boundary line(Red Line). Our objective when we are moving on with SVR is to basically consider the points that are within the boundary line. Our best fit line is the line hyperplane that has a maximum number of points.

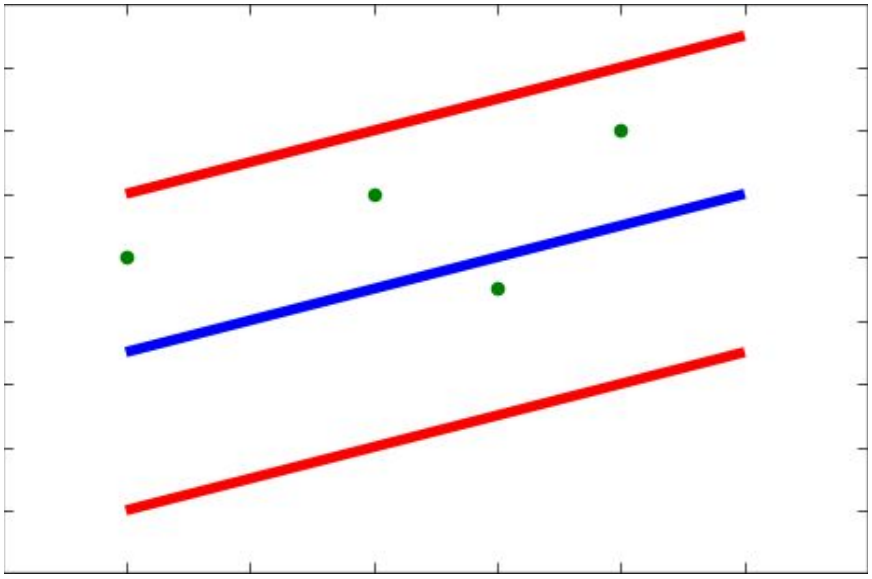

Fig 1

3. To predict the accuracy of machine algorithms used and find the most accurate algorithms:

```
In [59]: from sklearn.linear_model import LinearRegression
         from sklearn.metrics import accuracy_score
         def LinearRegressionModel(X,y):
          X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=42, test_size=0.3)
          regressor = LinearRegression()
          regressor.fit(X_train, y_train)
          y_pred = regressor.predict(X_test)
          print(print_accuracy_report(y_test, y_pred, X_test, regressor))
          return regressor
         linearModel = LinearRegressionModel(X,y)

         R Squared(Accuracy) 0.5905580302239338
         Mean Absolute Error: 0.2663007341496871
         Mean Squared Error: 0.10789533935250462
         Root Mean Squared Error: 0.32847425980205
         Root Mean Squared Log Error 0.032934077650799545
         None
```

Fig 2: Accuracy measure of Linear Regression Model

```
In [60]: from sklearn.ensemble import RandomForestRegressor
         def RandomForestRegressorModel(X,y):
             X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=42, test_size=0.3)
             rf = RandomForestRegressor(random_state=42)
             rf.fit(X_train, y_train)
             y_pred = rf.predict(X_test)
             print(print_accuracy_report(y_test, y_pred, X_test, rf))
             return rf
         randomForestModel = RandomForestRegressorModel(X,y)

C:\Users\Adarsh Shubham\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value
of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

R Squared(Accuracy) 0.9266568036075542
Mean Absolute Error: 0.07609541910308047
Mean Squared Error: 0.019327254282916713
Root Mean Squared Error: 0.13902249560023267
Root Mean Squared Log Error 0.013879577542090911
None
```

Fig 3: Accuracy measure of Random Forest Model

```
In [202]: from sklearn.svm import SVR
          def SVRModel(X,y):
              X_train, X_test, y_train, y_test = train_test_split(X, y,random_state=42, test_size=0.3)
              regressor=SVR(kernel='linear',degree=1)
              regressor.fit(X_train, y_train)
              y_pred = regressor.predict(X_test)
              print(print_accuracy_report(y_test, y_pred, X_test, regressor))
              return regressor
          svRegressorModel = SVRModel(X,y)

R Squared(Accuracy) 0.5985541128012077
Mean Absolute Error: 0.2596930551489981
Mean Squared Error: 0.10578822746156323
Root Mean Squared Error: 0.3252510222298513
Root Mean Squared Log Error 0.032581338039202414
None
```

Fig 4: Accuracy measure of Support Vector Regression

From these screen snips we can see that the Random Forest algorithm is the most accurate algorithm for the price prediction system. So the random forest algorithm will predict the most accurate price for the flight on a given certain day.

Screen Snips for the Work Done

```
In [25]: flight_train.head()
```

Out[25]:

	Route 1	Route 2	Route 3	Route 4	Route 5	Flight 1	Flight 2	Flight 3	Flight 4	Flight 5	Airline 1	Airline 2	Airline 3	Airline 4	Airline 5	base_fare	total_fare	dep_date_utc
0	DEL-BBI	BBI-BOM	NaN	NaN	NaN	AI-473	AI-670	NaN	NaN	NaN	AI	AI	NaN	NaN	NaN	6400	7346	01-05-2020
1	BOM-HYD	HYD-DEL	NaN	NaN	NaN	6.00E-234	6.00E-308	NaN	NaN	NaN	6E	6E	NaN	NaN	NaN	5412	7752	01-05-2020
2	BOM-HYD	HYD-DEL	NaN	NaN	NaN	6E-461	6E-6137	NaN	NaN	NaN	6E	6E	NaN	NaN	NaN	6392	8819	01-05-2020
3	DEL-BOM	NaN	NaN	NaN	NaN	6.00E-191	NaN	NaN	NaN	NaN	6E	NaN	NaN	NaN	NaN	3917	5064	01-05-2020
4	DEL-HYD	HYD-BOM	NaN	NaN	NaN	6E-5055	6E-428	NaN	NaN	NaN	6E	6E	NaN	NaN	NaN	6876	9293	01-05-2020

Fig 5: Training Dataset

```
In [5]: flight_test.head()
```

Out[5]:

	one-way	return	itenary	multi-city	time_taken	Route 1	Route 2	Route 3	Route 4	Route 5	departure_date_utc
0	O-154	R-154	I-0	MD-0	14956	DEL-BOM	BOM-DEL	NaN	NaN	NaN	01-05-2020
1	O-87	R-0	I-0	MD-0	61267	DEL-BOM	NaN	NaN	NaN	NaN	28-12-2018
2	O-93	R-0	I-0	MD-0	61507	DEL-BOM	NaN	NaN	NaN	NaN	27-12-2018
3	O-93	R-0	I-0	MD-0	60940	DEL-BOM	NaN	NaN	NaN	NaN	27-12-2018
4	O-61	R-0	I-0	MD-0	61589	DEL-BOM	NaN	NaN	NaN	NaN	28-12-2018

Fig 6: Testing Dataset

```
In [13]: flight_train.loc[flight_train['total_fare'].idxmax()]
```

```
Out[13]: Route 1            AUH-LHR
         Route 2            STN-DXB
         Route 3            DXB-BAH
         Route 4                NaN
         Route 5                NaN
         Flight 1             EY-19
         Flight 2             EK-68
         Flight 3            EK-837
         Flight 4                NaN
         Flight 5                NaN
         Airline 1                EY
         Airline 2                EK
         Airline 3                EK
         Airline 4               NaN
         Airline 5               NaN
         base_fare            910065
         total_fare           975606
         dep_date_utc     20-12-2019
         Name: 645250, dtype: object
```

Fig 7: Route with maximum fare

```
In [12]: flight_train.loc[flight_train['total_fare'].idxmin()]

Out[12]: Route 1            BOM-DEL
         Route 2                NaN
         Route 3                NaN
         Route 4                NaN
         Route 5                NaN
         Flight 1            9W-333
         Flight 2               NaN
         Flight 3               NaN
         Flight 4               NaN
         Flight 5               NaN
         Airline 1               9W
         Airline 2              NaN
         Airline 3              NaN
         Airline 4              NaN
         Airline 5              NaN
         base_fare                0
         total_fare               0
         dep_date_utc    27-11-2018
         Name: 1581, dtype: object
```

Fig 8: Route with minimum fare

```
In [21]: flight_train['Date_of_Journey'] = pd.to_datetime(flight_train['dep_date_utc'])

In [22]: flight_train['Month_of_Journey'] = flight_train['Date_of_Journey'].dt.month
         flight_train['Day_of_Journey'] = flight_train['Date_of_Journey'].dt.day
         flight_train['Date_of_journey_weekname'] = flight_train['Date_of_Journey'].dt.weekday_name
         flight_train['Is_weekend'] = np.where(flight_train['Date_of_journey_weekname'].isin(['Sunday','Saturday']),1,0)

In [23]: flight_test['Date_of_Journey'] = pd.to_datetime(flight_test['departure_date_utc'])

In [24]: flight_test['Month_of_Journey'] = flight_test['Date_of_Journey'].dt.month
         flight_test['Day_of_Journey'] = flight_test['Date_of_Journey'].dt.day
         flight_test['Date_of_journey_weekname'] = flight_test['Date_of_Journey'].dt.weekday_name
         flight_test['Is_weekend'] = np.where(flight_test['Date_of_journey_weekname'].isin(['Sunday','Saturday']),1,0)
```

Fig 9: Extracting month, day and checking whether it is weekend or not from departure date

```
In [55]: n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
         max_features = ['auto', 'sqrt']
         max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
         min_samples_split = [2, 5, 10, 15, 100]
         min_samples_leaf = [1, 2, 5, 10]

In [56]: rf_grid={'n_estimators':n_estimators,'max_features':max_features,'max_depth':max_depth,
                  'min_samples_split':min_samples_split,'min_samples_leaf':min_samples_leaf}

In [57]: rf =RandomForestRegressor()

In [58]: best_rf_tree=RandomizedSearchCV(estimator=rf,param_distributions=rf_grid,n_jobs=-1,cv=3,scoring='neg_mean_squared_error

In [62]: best_rf_tree.fit(X_Train,Y_Train)

Out[62]: RandomizedSearchCV(cv=3, error_score='raise-deprecating',
                   estimator=RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators='warn', n_jobs=None,
                    oob_score=False, random_state=None, verbose=0, warm_start=False),
                   fit_params=None, iid='warn', n_iter=10, n_jobs=-1,
                   param_distributions={'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_
features': ['auto', 'sqrt'], 'max_depth': [5, 10, 15, 20, 25, 30], 'min_samples_split': [2, 5, 10, 15, 100], 'min_sam
ples_leaf': [1, 2, 5, 10]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score='warn', scoring='neg_mean_squared_error',
                   verbose=0)
```

Fig 10: Training model using Linear Regression

```
In [63]: rf_predict =sc.inverse_transform(best_rf_tree.predict(X_Test))

In [64]: rf_predict

Out[64]: array([ 9298.55631142, 22162.29672416, 22162.29672416, ...,
                12284.25317733,  5270.89305116,  7706.33852307])

In [65]: pd.DataFrame(np.around(rf_predict),columns=['Price']).to_excel('Output_grid_f.xlsx',index=False)

In [66]: rf.fit(X_Train,Y_Train)

         C:\Users\Adarsh Shubham\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:246: FutureWarning: The default value
         of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
           "10 in version 0.20 to 100 in 0.22.", FutureWarning)

Out[66]: RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
                    max_features='auto', max_leaf_nodes=None,
                    min_impurity_decrease=0.0, min_impurity_split=None,
                    min_samples_leaf=1, min_samples_split=2,
                    min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=None,
                    oob_score=False, random_state=None, verbose=0, warm_start=False)

In [67]: rf_predict1=sc.inverse_transform( rf.predict(X_Test))
```

Fig 11: Using random forest algorithm

```
In [132]: svr = SVR(kernel='rbf')

In [133]: svr.fit(X_Train,Y_Train)

Out[133]: SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1,
            gamma='auto_deprecated', kernel='rbf', max_iter=-1, shrinking=True,
            tol=0.001, verbose=False)

In [134]: svr_predict = svr.predict(X_Test)

In [135]: svr_predict

Out[135]: array([-0.10152825, -0.10152825, -0.10152825, ..., -0.37051748,
                 -0.10152825, -0.10152825])

In [136]: svr_predict_orginal = sc.inverse_transform(svr_predict)
```

Fig 12: Using Support Vector Regression

O-209	R-0	I-0	MD-0	32865	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	5734	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	6831	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	9478	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	7023	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	6877	DEL-PNQ					22-08-2019	9478
O-217	R-0	I-0	MD-0	11378	DEL-PNQ					22-08-2019	9478
O-212	R-0	I-0	MD-0	10056	DEL-PNQ					22-08-2019	9478
O-31	R-0	I-0	MD-0	223	DEL-BOM					20-03-2019	8038
O-22	R-0	I-0	MD-0	7138	DEL-BOM					28-02-2019	8546
O-22	R-0	I-0	MD-0	5043	DEL-BOM					28-02-2019	8546
O-17	R-0	I-0	MD-0	5510	BLR-DEL					29-03-2019	4578
O-111	R-0	I-0	MD-0	61735	DEL-BOM					07-01-2019	6766
O-111	R-0	I-0	MD-0	61164	DEL-BOM					07-01-2019	6766
O-0	R-0	I-42	MD-105	64147	DEL-BOM	BOM-BLR	BLR-MAA	MAA-DEL		06-12-2018	10783
O-0	R-0	I-42	MD-0	50214	DEL-PNQ					08-01-2020	7697
O-0	R-0	I-0	MD-0	89312	DEL-HYD					07-02-2020	3409
O-1	R-1	I-1	MD-1	89312	BOM-BLR					09-01-2020	4363
O-2	R-2	I-2	MD-2	89312	BLR-DEL					10-01-2020	6756
O-3	R-3	I-3	MD-3	89313	BOM-BLR					15-01-2020	4321
O-4	R-4	I-4	MD-4	89314	DEL-BOM					18-01-2020	6227
O-5	R-5	I-5	MD-5	89315	PNQ-BLR					07-02-2020	2983

Fig 13: Most Accurate Price Predicted