DWH LAB MANUAL

Prediction System

JUNE 6, 2022

Prediction System:

This prediction helps us check different anime manga sold through different routes.

Data sets are available on google drive:

https://drive.google.com/drive/folders/1ryz29EIZ9Pc7xYX8h7o9ihCl5jUDEtX4?usp=sharing

Importing dataset:

- Since data is in form of excel file we have to use pandas read excel to load the data
- After loading it is important to check the complete information of data as it can indication many of the hidden information such as null values in a column or a row
- Check whether any null values are there or not. if it is present then following can be done,
- Imputing data using Imputation method in sklearn
- Filling NaN values with mean, median and mode using fillna() method
- Describe data --> which can give statistical analysis

```
[1]  import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns

     sns.set()
```

```
[3]  train_data = pd.read_excel(r"/content/Data_Train.xlsx")
```

```
     pd.set_option('display.max_columns', None)
```

```
[5]  train_data.head()
```

```
     train_data.info()
```

```
[7]  train_data["Duration"].value_counts()
```

```
[8]  train_data.dropna(inplace = True)
```

```
[9]  train_data.isnull().sum()
```

EDA

- From description we can see that Date of Journey is a object data type,
- Therefore, we have to convert this datatype into timestamp so as to use this column properly for prediction

- For this we require pandas to datetime to convert object data type to datetime dtype.

- dt.day method will extract only day of that date
- dt.month method will extract only month of that date

```
[10] train_data["Journey_day"] = pd.to_datetime(train_data.Date_of_Journey, format="%d/%m/%Y").dt.day

[11] train_data["Journey_month"] = pd.to_datetime(train_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month

[12] train_data.head()
```

```
[13] # Since we have converted Date_of_Journey column into integers, Now we can drop as it is of no use.

     train_data.drop(["Date_of_Journey"], axis = 1, inplace = True)
```

```
[14] # Departure time is when a plane leaves the gate.
     # Similar to Date_of_Journey we can extract values from Dep_Time

     # Extracting Hours
     train_data["Dep_hour"] = pd.to_datetime(train_data["Dep_Time"]).dt.hour

     # Extracting Minutes
     train_data["Dep_min"] = pd.to_datetime(train_data["Dep_Time"]).dt.minute

     # Now we can drop Dep_Time as it is of no use
     train_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

```
[15] train_data.head()
```

```
[16] # Arrival time is when the plane pulls up to the gate.
     # Similar to Date_of_Journey we can extract values from Arrival_Time

     # Extracting Hours
     train_data["Arrival_hour"] = pd.to_datetime(train_data.Arrival_Time).dt.hour

     # Extracting Minutes
     train_data["Arrival_min"] = pd.to_datetime(train_data.Arrival_Time).dt.minute

     # Now we can drop Arrival_Time as it is of no use
     train_data.drop(["Arrival_Time"], axis = 1, inplace = True)
```

```
[17] train_data.head()
```

```
[18] # Time taken by plane to reach destination is called Duration
     # It is the differnce betwwen Departure Time and Arrival time


     # Assigning and converting Duration column into list
     duration = list(train_data["Duration"])

     for i in range(len(duration)):
         if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
             if "h" in duration[i]:
                 duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
             else:
                 duration[i] = "0h " + duration[i]           # Adds 0 hour

     duration_hours = []
     duration_mins = []
     for i in range(len(duration)):
         duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
         duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration
```

```
[19] # Adding duration_hours and duration_mins list to train_data dataframe

     train_data["Duration_hours"] = duration_hours
     train_data["Duration_mins"] = duration_mins
```

```
[20] train_data.drop(["Duration"], axis = 1, inplace = True)

[21] train_data.head()
```

Handling Categorical Data:

- One can find many ways to handle categorical data. Some of them categorical data are,

- Nominal data --> data are not in any order --> OneHotEncoder is used in this case • Ordinal data --> data are in order --> LabelEncoder is used in this case

```
[22] train_data["Airline"].value_counts()
```

```
[23] # From graph we can see that Jet Airways Business have the highest Price.
     # Apart from the first Airline almost all are having similar median

     # Airline vs Price
     sns.catplot(y = "Price", x = "Airline", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 6, aspect = 3)
     plt.show()
```

```
[24] # As Airline is Nominal Categorical data we will perform OneHotEncoding

     Airline = train_data[["Airline"]]

     Airline = pd.get_dummies(Airline, drop_first= True)

     Airline.head()
```

```
[25] train_data["Source"].value_counts()
```

```
[26] # Source vs Price

     sns.catplot(y = "Price", x = "Source", data = train_data.sort_values("Price", ascending = False), kind="boxen", height = 4, aspect = 3)
     plt.show()
```

```
[27] # As Source is Nominal Categorical data we will perform OneHotEncoding

     Source = train_data[["Source"]]

     Source = pd.get_dummies(Source, drop_first= True)

     Source.head()
```

```
[28] train_data["Destination"].value_counts()
```

```
[29] # As Destination is Nominal Categorical data we will perform OneHotEncoding

     Destination = train_data[["Destination"]]

     Destination = pd.get_dummies(Destination, drop_first = True)

     Destination.head()
```

```
[30] train_data["Route"]
```

```
[31] # Additional_Info contains almost 80% no_info
     # Route and Total_Stops are related to each other

     train_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)
```

```
[32] train_data["Total_Stops"].value_counts()
```

```
[33] # As this is case of Ordinal Categorical type we perform LabelEncoder
     # Here Values are assigned with corresponding keys

     train_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

```
[34] train_data.head()
```

```
[35] # Concatenate dataframe --> train_data + Airline + Source + Destination

     data_train = pd.concat([train_data, Airline, Source, Destination], axis = 1)

[36] data_train.head()
```

```
[37] data_train.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

[38] data_train.head()
```

```
[39] data_train.shape
```

Test set:

```
[40] test_data = pd.read_excel(r"/content/Test_set.xlsx")

[41] test_data.head()
```

```
[42] # Preprocessing

     print("Test data Info")
     print("-"*75)
     print(test_data.info())

     print()
     print()

     print("Null values :")
     print("-"*75)
     test_data.dropna(inplace = True)
     print(test_data.isnull().sum())

     # EDA

     # Date_of_Journey
     test_data["Journey_day"] = pd.to_datetime(test_data.Date_of_Journey, format="%d/%m/%Y").dt.day
     test_data["Journey_month"] = pd.to_datetime(test_data["Date_of_Journey"], format = "%d/%m/%Y").dt.month
     test_data.drop(["Date_of_Journey"], axis = 1, inplace = True)

     # Dep_Time
     test_data["Dep_hour"] = pd.to_datetime(test_data["Dep_Time"]).dt.hour
     test_data["Dep_min"] = pd.to_datetime(test_data["Dep_Time"]).dt.minute
     test_data.drop(["Dep_Time"], axis = 1, inplace = True)
```

```
# Arrival_Time
test_data["Arrival_hour"] = pd.to_datetime(test_data.Arrival_Time).dt.hour
test_data["Arrival_min"] = pd.to_datetime(test_data.Arrival_Time).dt.minute
test_data.drop(["Arrival_Time"], axis = 1, inplace = True)

# Duration
duration = list(test_data["Duration"])

for i in range(len(duration)):
    if len(duration[i].split()) != 2:    # Check if duration contains only hour or mins
        if "h" in duration[i]:
            duration[i] = duration[i].strip() + " 0m"   # Adds 0 minute
        else:
            duration[i] = "0h " + duration[i]           # Adds 0 hour

duration_hours = []
duration_mins = []
for i in range(len(duration)):
    duration_hours.append(int(duration[i].split(sep = "h")[0]))    # Extract hours from duration
    duration_mins.append(int(duration[i].split(sep = "m")[0].split()[-1]))   # Extracts only minutes from duration

# Adding Duration column to test set
test_data["Duration_hours"] = duration_hours
test_data["Duration_mins"] = duration_mins
test_data.drop(["Duration"], axis = 1, inplace = True)
```

```
# Categorical data

print("Airline")
print("-"*75)
print(test_data["Airline"].value_counts())
Airline = pd.get_dummies(test_data["Airline"], drop_first= True)

print()

print("Source")
print("-"*75)
print(test_data["Source"].value_counts())
Source = pd.get_dummies(test_data["Source"], drop_first= True)

print()

print("Destination")
print("-"*75)
print(test_data["Destination"].value_counts())
Destination = pd.get_dummies(test_data["Destination"], drop_first = True)

# Additional_Info contains almost 80% no_info
# Route and Total_Stops are related to each other
test_data.drop(["Route", "Additional_Info"], axis = 1, inplace = True)

# Replacing Total_Stops
test_data.replace({"non-stop": 0, "1 stop": 1, "2 stops": 2, "3 stops": 3, "4 stops": 4}, inplace = True)
```

```
# Concatenate dataframe --> test_data + Airline + Source + Destination
data_test = pd.concat([test_data, Airline, Source, Destination], axis = 1)

data_test.drop(["Airline", "Source", "Destination"], axis = 1, inplace = True)

print()
print()

print("Shape of test data : ", data_test.shape)
```

```
[43] data_test.head()
```

Feature Selection:

Finding out the best feature which will contribute and have good relation with target variable. Following are some of the feature selection methods,

- heatmap
- feature_importance_
- SelectKBest

```
[44] data_train.shape
```

```
[45] data_train.columns
```

```
[46] X = data_train.loc[:, ['Total_Stops', 'Journey_day', 'Journey_month', 'Dep_hour',
           'Dep_min', 'Arrival_hour', 'Arrival_min', 'Duration_hours',
           'Duration_mins', 'Airline_Air India', 'Airline_GoAir', 'Airline_IndiGo',
           'Airline_Jet Airways', 'Airline_Jet Airways Business',
           'Airline_Multiple carriers',
           'Airline_Multiple carriers Premium economy', 'Airline_SpiceJet',
           'Airline_Trujet', 'Airline_Vistara', 'Airline_Vistara Premium economy',
           'Source_Chennai', 'Source_Delhi', 'Source_Kolkata', 'Source_Mumbai',
           'Destination_Cochin', 'Destination_Delhi', 'Destination_Hyderabad',
           'Destination_Kolkata', 'Destination_New Delhi']]
     X.head()
```

```
[47] y = data_train.iloc[:, 1]
     y.head()
```

```
[48] # Finds correlation between Independent and dependent attributes

     plt.figure(figsize = (18,18))
     sns.heatmap(train_data.corr(), annot = True, cmap = "RdYlGn")

     plt.show()
```

```
[49] # Important feature using ExtraTreesRegressor

     from sklearn.ensemble import ExtraTreesRegressor
     selection = ExtraTreesRegressor()
     selection.fit(X, y)
```

```
[50] print(selection.feature_importances_)
```

```
[51] #plot graph of feature importances for better visualization

     plt.figure(figsize = (12,8))
     feat_importances = pd.Series(selection.feature_importances_, index=X.columns)
     feat_importances.nlargest(20).plot(kind='barh')
     plt.show()
```

Fitting model using Random Forest:

- Split dataset into train and test set in order to prediction w.r.t X_test
- If needed do scaling of data
- Scaling is not done in Random forest
- Import model
- Fit the data
- Predict w.r.t X_test
- In regression check RSME Score
- Plot graph

```
[52] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)
```

```
[53] from sklearn.ensemble import RandomForestRegressor
     reg_rf = RandomForestRegressor()
     reg_rf.fit(X_train, y_train)
```

```
[54] y_pred = reg_rf.predict(X_test)
```

```
[55] reg_rf.score(X_train, y_train)
```

```
[57] sns.distplot(y_test-y_pred)
     plt.show()
```

```
[58]
     plt.scatter(y_test, y_pred, alpha = 0.5)
     plt.xlabel("y_test")
     plt.ylabel("y_pred")
     plt.show()
```

```
[59] from sklearn import metrics

[60] print('MAE:', metrics.mean_absolute_error(y_test, y_pred))
     print('MSE:', metrics.mean_squared_error(y_test, y_pred))
     print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, y_pred)))
```

```
[61] # RMSE/(max(DV)-min(DV))

     2090.5509/(max(y)-min(y))
```

```
[62] metrics.r2_score(y_test, y_pred)
```

Hyperparameter Tuning:

- Choose following method for hyperparameter tuning
- RandomizedSearchCV --> Fast
- GridSearchCV
- Assign hyperparameters in form of dictionary
- Fit the model
- Check best parameters and best score

```
[63] from sklearn.model_selection import RandomizedSearchCV

[64] #Randomized Search CV

     # Number of trees in random forest
     n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
     # Number of features to consider at every split
     max_features = ['auto', 'sqrt']
     # Maximum number of levels in tree
     max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
     # Minimum number of samples required to split a node
     min_samples_split = [2, 5, 10, 15, 100]
     # Minimum number of samples required at each leaf node
     min_samples_leaf = [1, 2, 5, 10]
```

```
[67] # Random search of parameters, using 5 fold cross validation,
     # search across 100 different combinations
     rf_random = RandomizedSearchCV(estimator = reg_rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter = 10, cv = 5, verbose=2, random_state=42, n_jobs = 1)
```

```
[68] rf_random.fit(X_train,y_train)
```

```
[69] rf_random.best_params_
```

```
[70] prediction = rf_random.predict(X_test)

[71] plt.figure(figsize = (8,8))
     sns.distplot(y_test-prediction)
     plt.show()
```

```
[72] plt.figure(figsize = (8,8))
     plt.scatter(y_test, prediction, alpha = 0.5)
     plt.xlabel("y_test")
     plt.ylabel("y_pred")
     plt.show()
```

```
[73] print('MAE:', metrics.mean_absolute_error(y_test, prediction))
     print('MSE:', metrics.mean_squared_error(y_test, prediction))
     print('RMSE:', np.sqrt(metrics.mean_squared_error(y_test, prediction))) #root-mean-square-deviation
```

Task To Do:

1. Execute all above code and see results.
2. Execute the code using second data set given in google drive named as task 2.