FinalFlightProject(1)

May 23, 2023

1 COMP 494 Final Project

Authors: Madeleine Woo, Maggie Betts, Kenny Collins

Date: 5/23/2023

1.0.1 Flight Prices

Dataset: https://www.kaggle.com/datasets/chidinmaokonta/flight-price-prediction-dataset-freshly-cleaned

1.1 Originally sourced by Shubham Bathwal

"The objective of the study is to analyse the flight booking dataset obtained from 'Ease My Trip' website and to conduct various statistical hypothesis tests in order to get meaningful information from it.

'Easemytrip' is an internet platform for booking flight tickets, and hence a platform that potential passengers use to buy tickets. A thorough study of the data will aid in the discovery of valuable insights that will be of enormous value to both business owners and passengers."

1.1.1 Final Project Requirements:

There are four sections of the final project. You are expected to perform the following tasks within each section to fulfill the project requirements. - **Data Importing and Pre-processing (50 Points)** - Import dataset and describe characteristics such as dimensions, data types, file types, and import methods used - Clean, wrangle, and handle missing data - Transform data appropriately using techniques such as aggregation, normalization, and feature construction - Reduce redundant data and perform need based discretization - **Data Analysis and Visualization (50 Points)** - Identify categorical, ordinal, and numerical variables within data - Provide measures of centrality and distribution with visualizations - Diagnose for correlations between variables and determine independent and dependent variables - Perform exploratory analysis in combination with visualization techniques to discover patterns and features of interest - **Data Analytics (50 Points)** - Determine the need for a supervised or unsupervised learning method and identify dependent and independent variables - Train, test, and provide accuracy and evaluation metrics for model results - **Presentation (50 Points)** - In a 5 to 10 minute presentation, briefly explain the project workflow from the code and results in your markdown notebook State your findings from the data and provide the interpretation of results from your analysis at each stage in the project

1.2 Table of Contents:

- Data Importing and Pre-processing
- Data Analysis and Visualization
- Data Analytics

1.3 Data Importing and Pre-processing

[ ]:
```
#import libraries needed
import pandas as pd
pd.set_option('display.max_columns', None)
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from scipy.stats import norm, skew, probplot
from scipy.special import boxcox1p
import warnings
warnings.filterwarnings('ignore')
from sklearn.linear_model import Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import KFold, cross_val_score
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
import xgboost as xgb
import lightgbm as lgb
import numpy as np
from sklearn.model_selection import train_test_split as tts
%matplotlib inline
```

[ ]:
```
#read in file
flights_df = pd.read_csv('Freshly_cleaned.csv')
```

[ ]:
```
#check number of rows and columns
flights_df.shape
```

[ ]: (300257, 25)

[ ]:
```
#count the number of categorical variables
cat_count = 0
for dtype in flights_df.dtypes:
    if dtype == 'object':
        cat_count = cat_count + 1
```

[ ]:
```
print('# of categorical variables:',cat_count)
print('# of continuous variables:',flights_df.shape[1] - cat_count - 1)
 ↪#subtract and extra column as 1 column is an ID column
```

# of categorical variables: 12
# of continuous variables: 12

[ ]:
```
flights_df.head()
```

[ ]:
```
   Unnamed: 0   airline   from       to  price class_category  class  day  \
0           0  SpiceJet  Delhi  Mumbai   5953        Economy      0   11
1           1  SpiceJet  Delhi  Mumbai   5953        Economy      0   11
2           2   AirAsia  Delhi  Mumbai   5956        Economy      0   11
3           3   Vistara  Delhi  Mumbai   5955        Economy      0   11
4           4   Vistara  Delhi  Mumbai   5955        Economy      0   11

   month flight_no         route  dep_hour  arr_hour     dep_period  \
0      2   SG-8709  Delhi-Mumbai        18        21      Afternoon
1      2   SG-8157  Delhi-Mumbai         6         8  Early_morning
2      2    I5-764  Delhi-Mumbai         4         6  Early_morning
3      2    UK-995  Delhi-Mumbai        10        12        Morning
4      2    UK-963  Delhi-Mumbai         8        11        Morning

      arr_period  airline_index  route_index  duration_in_min  stops  \
0          Night              4           14              130      0
1        Morning              4           14              140      0
2  Early_morning              1           14              130      0
3        Morning              7           14              135      0
4        Morning              7           14              140      0

  stops_category  arr_daytime arr_daytime_category  dep_daytime  \
0       Non-stop            0        Night Arrival            1
1       Non-stop            1      Daytime Arrival            1
2       Non-stop            1      Daytime Arrival            0
3       Non-stop            1      Daytime Arrival            1
4       Non-stop            1      Daytime Arrival            1

  dep_daytime_category month_category
0    Daytime Departure       February
1    Daytime Departure       February
2      Night Departure       February
3    Daytime Departure       February
4    Daytime Departure       February
```

[ ]:
```
#check the column names
flights_df.columns
```

[ ]:
```
Index(['Unnamed: 0', 'airline', 'from', 'to', 'price', 'class_category',
       'class', 'day', 'month', 'flight_no', 'route', 'dep_hour', 'arr_hour',
       'dep_period', 'arr_period', 'airline_index', 'route_index',
       'duration_in_min', 'stops', 'stops_category', 'arr_daytime',
       'arr_daytime_category', 'dep_daytime', 'dep_daytime_category',
```

```
        'month_category'],
       dtype='object')
```

[ ]:
```
#missing data
total = flights_df.isnull().sum().sort_values(ascending=False)
percent = (flights_df.isnull().sum()/flights_df.isnull().count()).
 ↪sort_values(ascending=False)
missing_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])
missing_data.head(20)
```

[ ]:

	Total	Percent
Unnamed: 0	0	0.0
dep_period	0	0.0
dep_daytime_category	0	0.0
dep_daytime	0	0.0
arr_daytime_category	0	0.0
arr_daytime	0	0.0
stops_category	0	0.0
stops	0	0.0
duration_in_min	0	0.0
route_index	0	0.0
airline_index	0	0.0
arr_period	0	0.0
arr_hour	0	0.0
airline	0	0.0
dep_hour	0	0.0
route	0	0.0
flight_no	0	0.0
month	0	0.0
day	0	0.0
class	0	0.0

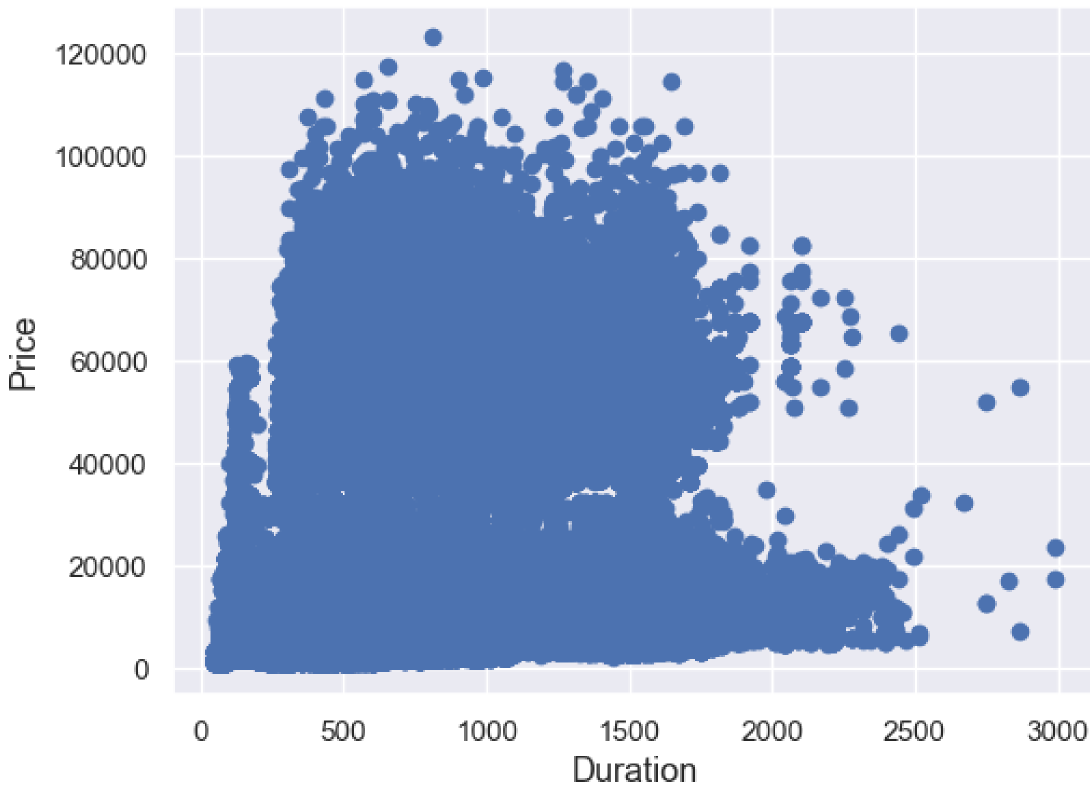
[ ]:
```
#Check remaining missing values if any
all_data_na = (flights_df.isnull().sum() / len(flights_df)) * 100
all_data_na = all_data_na.drop(all_data_na[all_data_na == 0].index).
 ↪sort_values(ascending=False)
missing_data = pd.DataFrame({'Missing Ratio' :all_data_na})
missing_data.head()
```

[ ]:
```
Empty DataFrame
Columns: [Missing Ratio]
Index: []
```

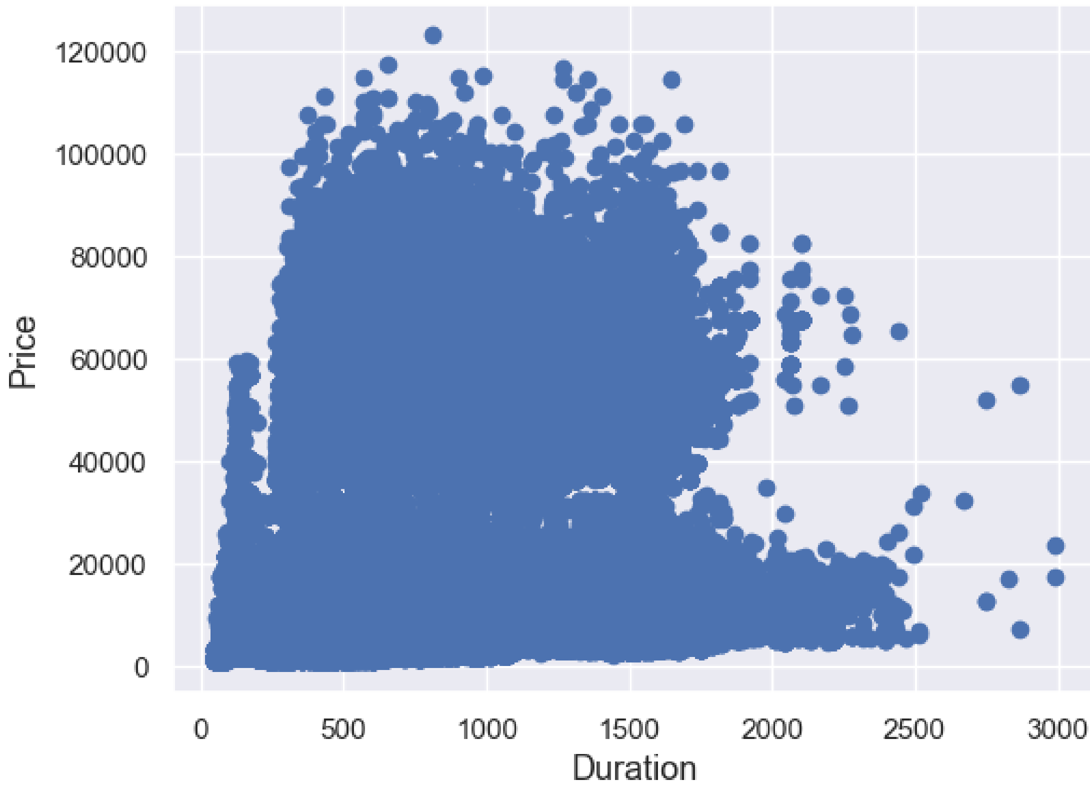
1.3.1 Handling Outliers

Target Variable

[ ]:
```
fig, ax = plt.subplots()
ax.scatter(x = flights_df['duration_in_min'], y = flights_df['price'])
plt.ylabel('Price', fontsize=13)
plt.xlabel('Duration', fontsize=13)
plt.show()
```



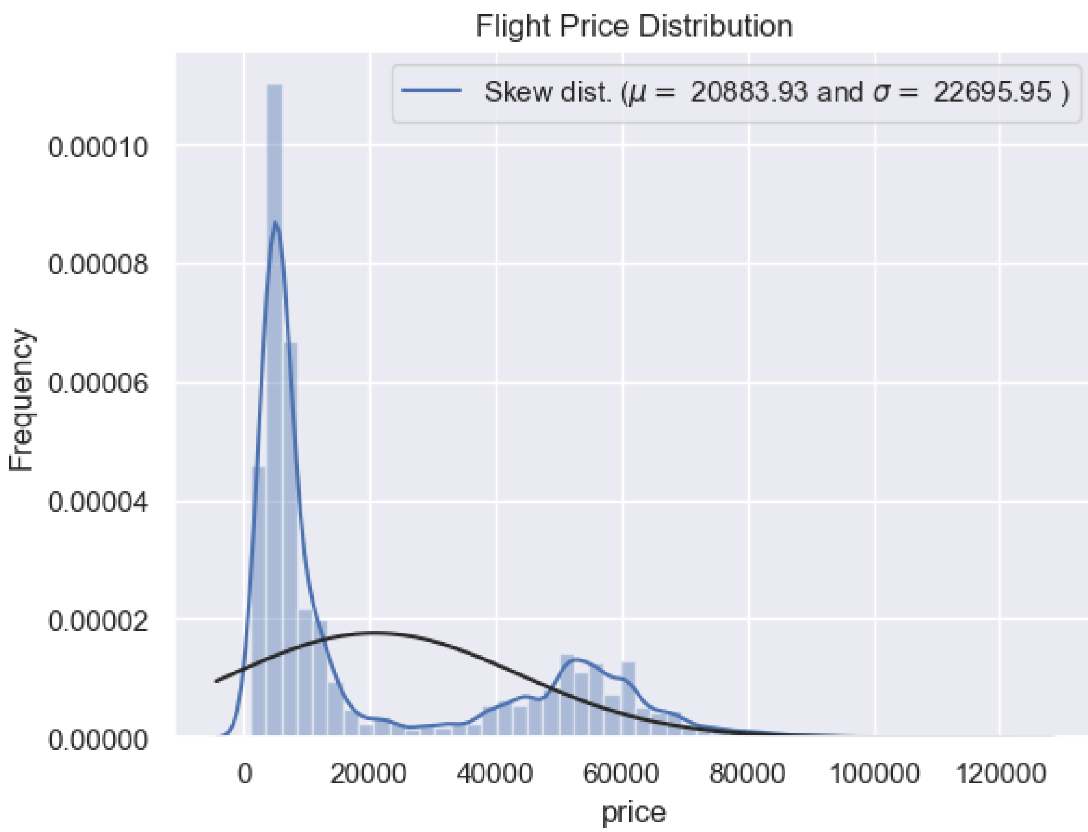
[ ]:
```
#Deleting outliers
flights_df = flights_df.drop(flights_df[(flights_df['price']>120000) &␣
↪(flights_df['duration_in_min']>2500)].index)

#Check the graphic again
fig, ax = plt.subplots()
ax.scatter(flights_df['duration_in_min'], flights_df['price'])
plt.ylabel('Price', fontsize=13)
plt.xlabel('Duration', fontsize=13)
plt.show()
```

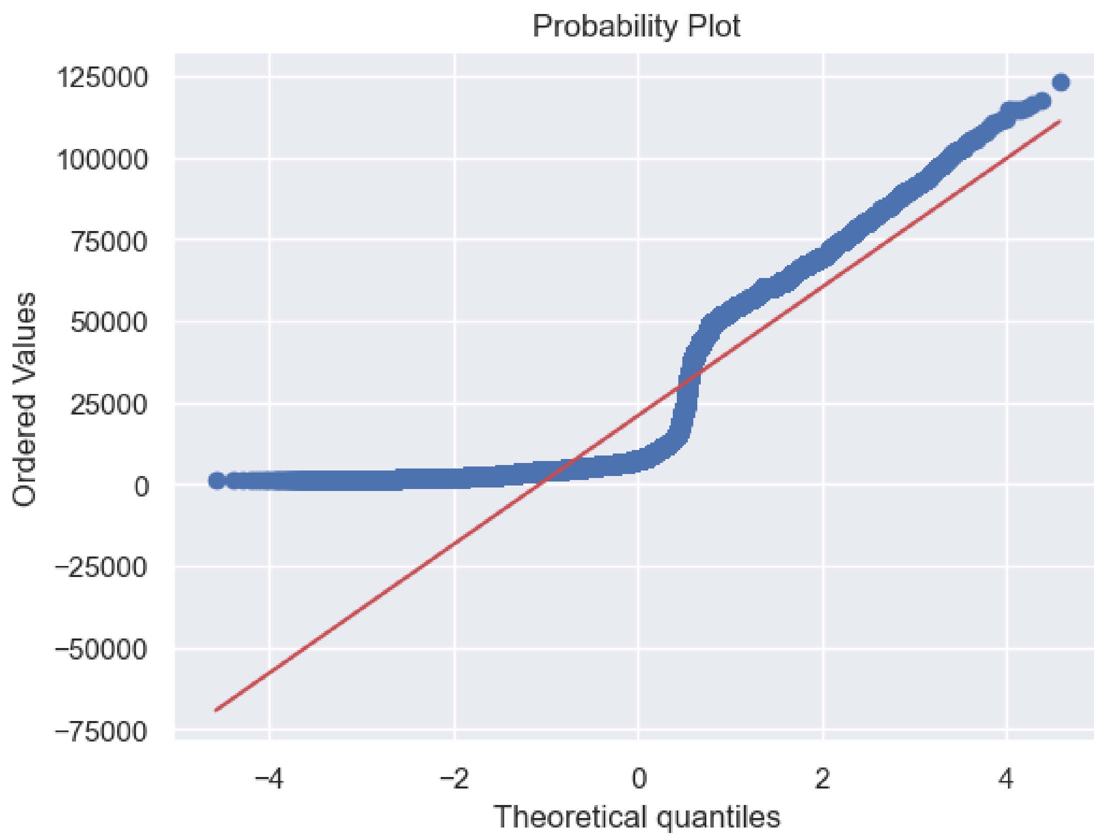


1.3.2 Normalize Target Variable

[ ]:
```
sns.distplot(flights_df['price'], fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(flights_df['price'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Skew dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,␣
↪sigma)],loc='best')
plt.ylabel('Frequency')
plt.title('Flight Price Distribution')

#Get also the QQ-plot
fig = plt.figure()
res = probplot(flights_df['price'], plot=plt)
plt.show()
```

```
mu = 20883.93 and sigma = 22695.95
```

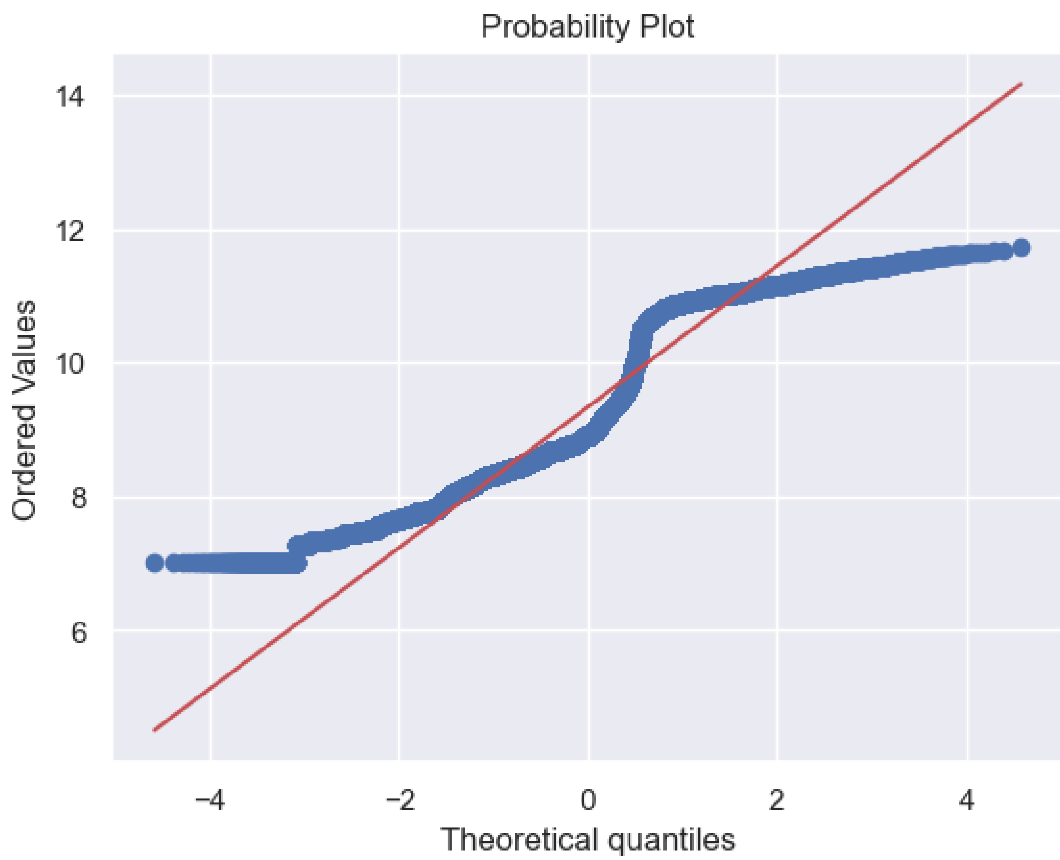

Flight Price Distribution
Skew dist. ($\mu =$ 20883.93 and $\sigma =$ 22695.95)
Frequency
0.00010
0.00008
0.00006
0.00004
0.00002
0.00000
0
20000
40000
60000
80000
100000
120000
price



[ ]:
```
#We use the numpy fuction log1p which  applies log(1+x) to all elements of the
 ↪column
flights_df["price"] = np.log1p(flights_df["price"])

#Check the new distribution
sns.distplot(flights_df['price'] , fit=norm);

# Get the fitted parameters used by the function
(mu, sigma) = norm.fit(flights_df['price'])
print( '\n mu = {:.2f} and sigma = {:.2f}\n'.format(mu, sigma))

#Now plot the distribution
plt.legend(['Normal dist. ($\mu=$ {:.2f} and $\sigma=$ {:.2f} )'.format(mu,
 ↪sigma)],
            loc='best')
plt.ylabel('Frequency')
plt.title('Flight Price Distribution')

#Get also the QQ-plot
fig = plt.figure()
```

```python
res = probplot(flights_df['price'], plot=plt)
plt.show()
```

```
mu = 9.33 and sigma = 1.11
```



1.4 Data Analysis and Visualization

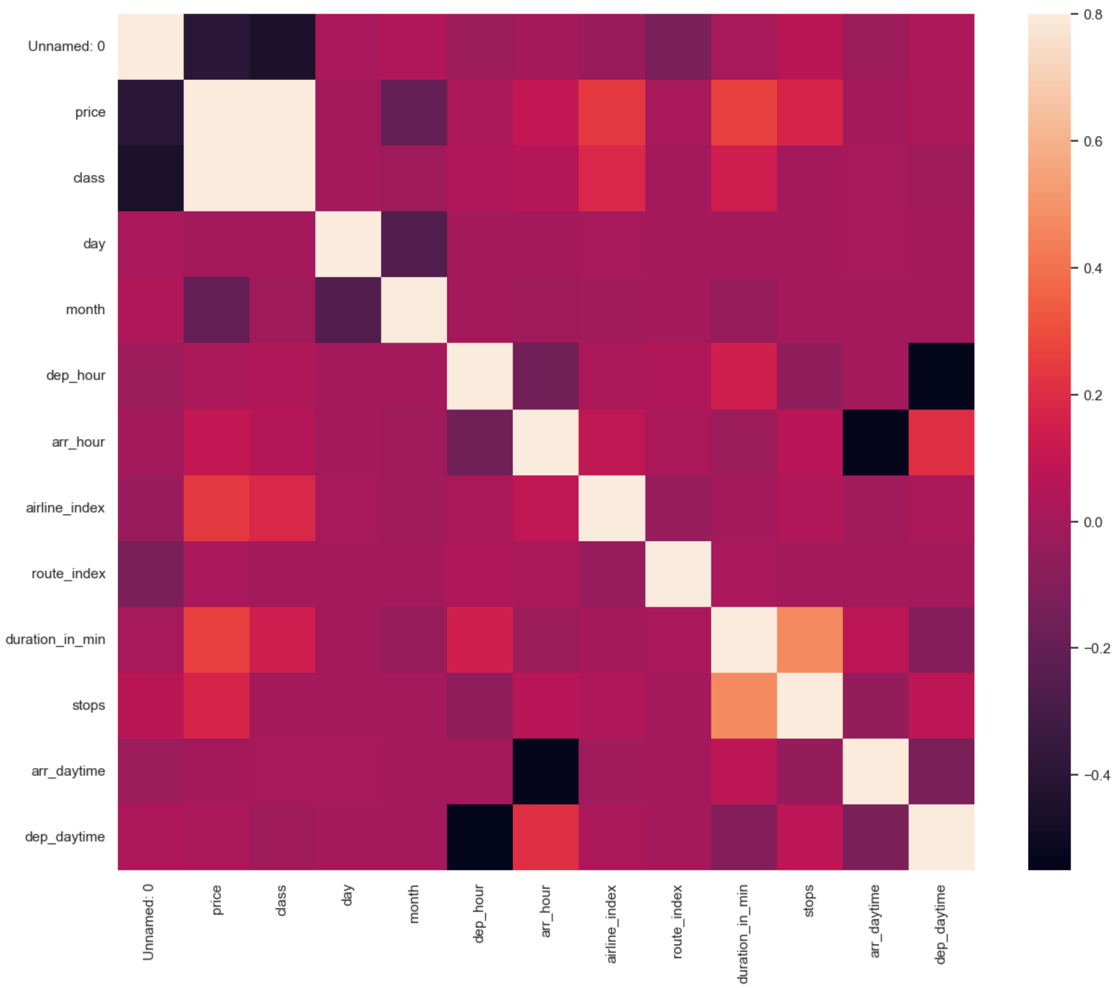
```
[ ]: from sklearn.preprocessing import LabelEncoder
```

Target Variable Scatterplots

```
[ ]: #scatterplot
sns.set()
cols = ['airline', 'from', 'to', 'price', 'class_category',
        'class', 'day', 'month', 'flight_no', 'route', 'dep_hour', 'arr_hour',
        'dep_period', 'arr_period', 'airline_index', 'route_index',
        'duration_in_min', 'stops', 'stops_category', 'arr_daytime',
        'arr_daytime_category', 'dep_daytime', 'dep_daytime_category',
        'month_category']
sns.pairplot(flights_df[cols], size = 2.5)
plt.show();
```

Correlation Matrix

```
[ ]: # Correlation map to see how flight characteristics are correlated with price
corrmat = flights_df.corr()
f, ax = plt.subplots(figsize=(15, 12))
sns.heatmap(corrmat, vmax=.8, square=True);
```



[ ]:
```
flights_df.head()
```

[ ]:
```
   Unnamed: 0   airline   from      to     price class_category  class  day  \
0           0  SpiceJet  Delhi  Mumbai  8.691819        Economy      0   11
1           1  SpiceJet  Delhi  Mumbai  8.691819        Economy      0   11
2           2   AirAsia  Delhi  Mumbai  8.692322        Economy      0   11
3           3   Vistara  Delhi  Mumbai  8.692154        Economy      0   11
4           4   Vistara  Delhi  Mumbai  8.692154        Economy      0   11

   month flight_no         route  dep_hour  arr_hour     dep_period  \
0      2   SG-8709  Delhi-Mumbai        18        21      Afternoon
1      2   SG-8157  Delhi-Mumbai         6         8  Early_morning
2      2    I5-764  Delhi-Mumbai         4         6  Early_morning
3      2    UK-995  Delhi-Mumbai        10        12        Morning
4      2    UK-963  Delhi-Mumbai         8        11        Morning
```

```
      arr_period  airline_index  route_index  duration_in_min  stops  \
0          Night              4           14              130      0
1        Morning              4           14              140      0
2  Early_morning              1           14              130      0
3        Morning              7           14              135      0
4        Morning              7           14              140      0

  stops_category  arr_daytime arr_daytime_category  dep_daytime  \
0       Non-stop            0        Night Arrival            1
1       Non-stop            1      Daytime Arrival            1
2       Non-stop            1      Daytime Arrival            0
3       Non-stop            1      Daytime Arrival            1
4       Non-stop            1      Daytime Arrival            1

  dep_daytime_category month_category
0    Daytime Departure       February
1    Daytime Departure       February
2      Night Departure       February
3    Daytime Departure       February
4    Daytime Departure       February
```

[ ]:
```python
flights_df['class'] = flights_df['class'].apply(str)

# make a new Date column

# Cast 'month' column to string values
flights_df['month'] = flights_df['month'].apply(str)
flights_df['day'] = flights_df['day'].apply(str)


# Round down the 'day' column to integers
#flights_df['day'] = np.floor(flights_df['day']).astype(int)

# Create the 'date' column by concatenating 'month' and 'day' with fixed year␣
↪(2022)
flights_df['date'] = pd.to_datetime('2022-' + flights_df['month'].str.zfill(2)␣
↪+ '-' + flights_df['day'].str.zfill(2), errors='coerce')
flights_df['date'] = flights_df['date'].dt.date
flights_df['date'] = flights_df['date'].apply(str)
flights_df.head()
```

[ ]:
```
   Unnamed: 0   airline   from      to     price class_category class day  \
0           0  SpiceJet  Delhi  Mumbai  8.691819        Economy     0  11
1           1  SpiceJet  Delhi  Mumbai  8.691819        Economy     0  11
2           2   AirAsia  Delhi  Mumbai  8.692322        Economy     0  11
3           3   Vistara  Delhi  Mumbai  8.692154        Economy     0  11
4           4   Vistara  Delhi  Mumbai  8.692154        Economy     0  11
```

```
  month flight_no         route  dep_hour  arr_hour     dep_period  \
0     2   SG-8709  Delhi-Mumbai        18        21      Afternoon
1     2   SG-8157  Delhi-Mumbai         6         8  Early_morning
2     2    I5-764  Delhi-Mumbai         4         6  Early_morning
3     2    UK-995  Delhi-Mumbai        10        12        Morning
4     2    UK-963  Delhi-Mumbai         8        11        Morning

      arr_period  airline_index  route_index  duration_in_min  stops  \
0          Night              4           14              130      0
1        Morning              4           14              140      0
2  Early_morning              1           14              130      0
3        Morning              7           14              135      0
4        Morning              7           14              140      0

  stops_category  arr_daytime arr_daytime_category  dep_daytime  \
0       Non-stop            0        Night Arrival            1
1       Non-stop            1      Daytime Arrival            1
2       Non-stop            1      Daytime Arrival            0
3       Non-stop            1      Daytime Arrival            1
4       Non-stop            1      Daytime Arrival            1

  dep_daytime_category month_category        date
0    Daytime Departure       February  2022-02-11
1    Daytime Departure       February  2022-02-11
2      Night Departure       February  2022-02-11
3    Daytime Departure       February  2022-02-11
4    Daytime Departure       February  2022-02-11
```

1.4.1 Label encode categorical variables

[ ]:
```python
cols = ('airline', 'from', 'to', 'class_category',
       'class', 'flight_no', 'route',
       'dep_period', 'arr_period', 'stops_category',
       'arr_daytime_category', 'dep_daytime_category',
       'month_category', 'date')

# process columns, apply LabelEncoder to categorical features
for c in cols:
    lbl = LabelEncoder()
    lbl.fit(list(flights_df[c].values))
    flights_df[c] = lbl.transform(list(flights_df[c].values))

# shape
print('Shape flights_df: {}'.format(flights_df.shape))
```

```
Shape flights_df: (300257, 26)
```

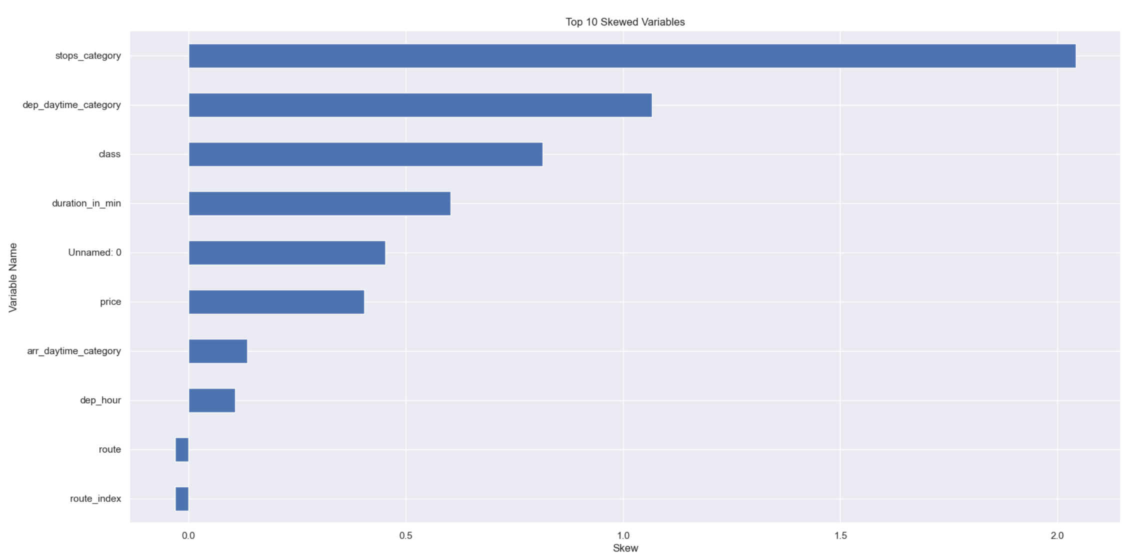
[ ]:
```
numeric_feats = flights_df.dtypes[flights_df.dtypes != "object"].index

# Check the skew of all numerical features
skewed_feats = flights_df[numeric_feats].apply(lambda x: skew(x.dropna())).
    ↪sort_values(ascending=False)
print("\nSkew in numerical features: \n")
skewness = pd.DataFrame({'Skew' :skewed_feats})
skewness.head(10)
```

```
Skew in numerical features:
```

[ ]:

	Skew
stops_category	2.041899
dep_daytime_category	1.066973
class	0.814789
duration_in_min	0.602977
Unnamed: 0	0.453130
price	0.404798
arr_daytime_category	0.134823
dep_hour	0.106441
route	-0.031934
route_index	-0.031934

[ ]:
```
skewness['Skew'].head(10).plot(kind='barh', figsize = (20,10)).invert_yaxis()␣
    ↪#top 10 missing columns
plt.xlabel("Skew")
plt.ylabel("Variable Name")
plt.title("Top 10 Skewed Variables")
plt.show()
```



[ ]:
```
skewness = skewness[abs(skewness) > 0.75]
print("There are {} skewed numerical features to Box Cox transform (normalize)".
    ↪format(skewness.shape[0]))

skewed_features = skewness.index
lam = 0.15
for feat in skewed_features:
    #all_data[feat] += 1
    flights_df[feat] = boxcox1p(flights_df[feat], lam)

#all_data[skewed_features] = np.log1p(all_data[skewed_features])
```

```
There are 24 skewed numerical features to Box Cox transform (normalize)
```

[ ]:
```
flights_df.head()
```

[ ]:
```
   Unnamed: 0   airline      from        to     price  class_category  class  \
0    0.000000  1.820334  1.194318  2.055642  2.706137        0.730463    0.0
1    0.730463  1.820334  1.194318  2.055642  2.706137        0.730463    0.0
2    1.194318  0.730463  1.194318  2.055642  2.706210        0.730463    0.0
3    1.540963  2.440268  1.194318  2.055642  2.706186        0.730463    0.0
4    1.820334  2.440268  1.194318  2.055642  2.706186        0.730463    0.0

  day month  flight_no    route  dep_hour  arr_hour  dep_period  arr_period  \
0  11     2  13.131010  3.34076  3.701973  3.932510    0.000000    1.540963
1  11     2  13.086719  3.34076  2.259674  2.602594    0.730463    1.194318
2  11     2  12.686669  3.34076  1.820334  2.259674    0.730463    0.730463
3  11     2  13.434009  3.34076  2.885846  3.128239    1.194318    1.194318
4  11     2  13.414727  3.34076  2.602594  3.011340    1.194318    1.194318

   airline_index  route_index  duration_in_min  stops  stops_category  \
0       1.820334      3.34076         7.184917    0.0        1.194318
1       1.820334      3.34076         7.338607    0.0        1.194318
2       0.730463      3.34076         7.184917    0.0        1.194318
3       2.440268      3.34076         7.262963    0.0        1.194318
4       2.440268      3.34076         7.338607    0.0        1.194318

   arr_daytime  arr_daytime_category  dep_daytime  dep_daytime_category  \
0     0.000000              0.730463     0.730463              0.000000
1     0.730463              0.000000     0.730463              0.000000
2     0.730463              0.000000     0.000000              0.730463
3     0.730463              0.000000     0.730463              0.000000
4     0.730463              0.000000     0.730463              0.000000

   month_category  date
```

```
0              0.0    0.0
1              0.0    0.0
2              0.0    0.0
3              0.0    0.0
4              0.0    0.0
```

1.4.2 Data Analytics

[ ]:
```
# Step 1: Prepare the Data
# Prepare X_train, y_train, X_test, y_test
X = flights_df[['airline', 'from', 'to', 'class_category',
       'class', 'flight_no', 'route',
       'dep_period', 'arr_period', 'stops_category',
       'arr_daytime_category', 'dep_daytime_category',
       'month_category']]
X = pd.get_dummies(X)  # One-hot encode categorical columns
Y = flights_df['price']

# Create the 'date' column by concatenating 'month' and 'day' with fixed year␣
↪(2022)
flights_df['date'] = pd.to_datetime('2022-' + flights_df['month'].astype(str) +␣
↪'-' + flights_df['day'].astype(str), errors='coerce')

X_train, X_test, y_train, y_test = tts(X, Y, test_size=0.2, random_state=42)

# Step 2: Split the Data
split_date = pd.to_datetime('2022-03-18')  # Convert split_date to pandas␣
↪datetime format
split_index = (flights_df['date'] >= split_date).idxmax()

X_train_split, y_train_split = X_train.iloc[:split_index], y_train.iloc[:
↪split_index]
X_test_split, y_test_split = X_train.iloc[split_index:], y_train.
↪iloc[split_index:]

# Step 4: Prepare Data for XGBoost
dtrain = xgb.DMatrix(X_train_split, label=y_train_split)
dtest = xgb.DMatrix(X_test_split, label=y_test_split)

# Step 5: Define XGBoost Parameters
params = {
    'objective': 'reg:squarederror',
    'eval_metric': 'rmse',
    'max_depth': 6,
    'eta': 0.1,
    'subsample': 0.8,
    'colsample_bytree': 0.8
```

```
}

# Step 6: Train the XGBoost Model
num_rounds = 100
model = xgb.train(params, dtrain, num_rounds)

# Step 7: Make Predictions
y_pred = model.predict(dtest)

# Step 8: Evaluate the Model
xgb_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
xgb_stdev = np.std(y_test_split)
print(f"XGB RMSE: {xgb_rmse:.4f}, std deviation: {xgb_stdev:.4f}")
```

```
XGB RMSE: 0.0365, std deviation: 0.1509
```

[ ]:
```
# Step 5: Define LightGBM Parameters
params = {
    'objective': 'regression',
    'metric': 'rmse',
    'max_depth': 6,
    'learning_rate': 0.1,
    'subsample': 0.8,
    'colsample_bytree': 0.8
}

# Step 6: Train the LightGBM Model
num_rounds = 100
lgb_train = lgb.Dataset(X_train_split, label=y_train_split)
lgb_test = lgb.Dataset(X_test_split, label=y_test_split)
model = lgb.train(params, lgb_train, num_rounds)

# Step 7: Make Predictions
y_pred = model.predict(X_test_split)

# Step 8: Evaluate the Model
lgb_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
lgb_std_dev = np.std(y_test_split)
print(f"LGBM RMSE: {lgb_rmse:.4f}, std deviation: {lgb_std_dev:.4f}")
```

```
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] Accuracy may be bad since you didn't explicitly set
num_leaves OR 2^max_depth > num_leaves. (num_leaves=31).
[LightGBM] [Warning] Auto-choosing row-wise multi-threading, the overhead of
testing was 0.000872 seconds.
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
```

```
[LightGBM] [Info] Total Bins 316
[LightGBM] [Info] Number of data points in the train set: 7029, number of used
features: 13
[LightGBM] [Info] Start training from score 2.789862
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
LGBM RMSE: 0.0368, std deviation: 0.1509
```

[ ]:

```python
# Step 5: Train the Linear Regression Model
model = LinearRegression()
model.fit(X_train_split, y_train_split)

# Step 6: Make Predictions
y_pred = model.predict(X_test_split)

# Step 7: Evaluate the Model
lr_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
lr_std_dev = np.std(y_test_split)
print(f"Linear Regression RMSE: {lr_rmse:.4f}, std deviation: {lr_std_dev:.4f}")
```

```
Linear Regression RMSE: 0.0540, std deviation: 0.1509
```

[ ]:

```python
# Step 5: Train the Linear Regression Model without intercept
model = LinearRegression(fit_intercept=False)
model.fit(X_train_split, y_train_split)

# Step 6: Make Predictions
y_pred = model.predict(X_test_split)

# Step 7: Evaluate the Model
lr_no_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
lr_no_std_dev = np.std(y_test_split)
print(f"Linear Regression (No Intercept) RMSE: {lr_no_rmse:.4f}, std deviation:␣
 ↪{lr_no_std_dev:.4f}")
```

```
Linear Regression (No Intercept) RMSE: 0.0540, std deviation: 0.1509
```

[ ]:
```
# Step 5: Train the Decision Tree Regressor
model = DecisionTreeRegressor()
model.fit(X_train_split, y_train_split)

# Step 6: Make Predictions
y_pred = model.predict(X_test_split)

# Step 7: Evaluate the Model
dt_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
dt_std_dev = np.std(y_test_split)
print(f"Decision Tree Regressor RMSE: {dt_rmse:.4f}, std deviation: {dt_std_dev:
↪.4f}")
```

```
Decision Tree Regressor RMSE: 0.0462, std deviation: 0.1509
```

[ ]:
```
# Step 5: Train the KNeighborsRegressor
model = KNeighborsRegressor()
model.fit(X_train_split, y_train_split)

# Step 6: Make Predictions
y_pred = model.predict(X_test_split)

# Step 7: Evaluate the Model
kn_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
kn_std_dev = np.std(y_test_split)
print(f"KNeighborsRegressor RMSE: {kn_rmse:.4f}, std deviation: {kn_std_dev:.
↪4f}")
```

```
KNeighborsRegressor RMSE: 0.0484, std deviation: 0.1509
```

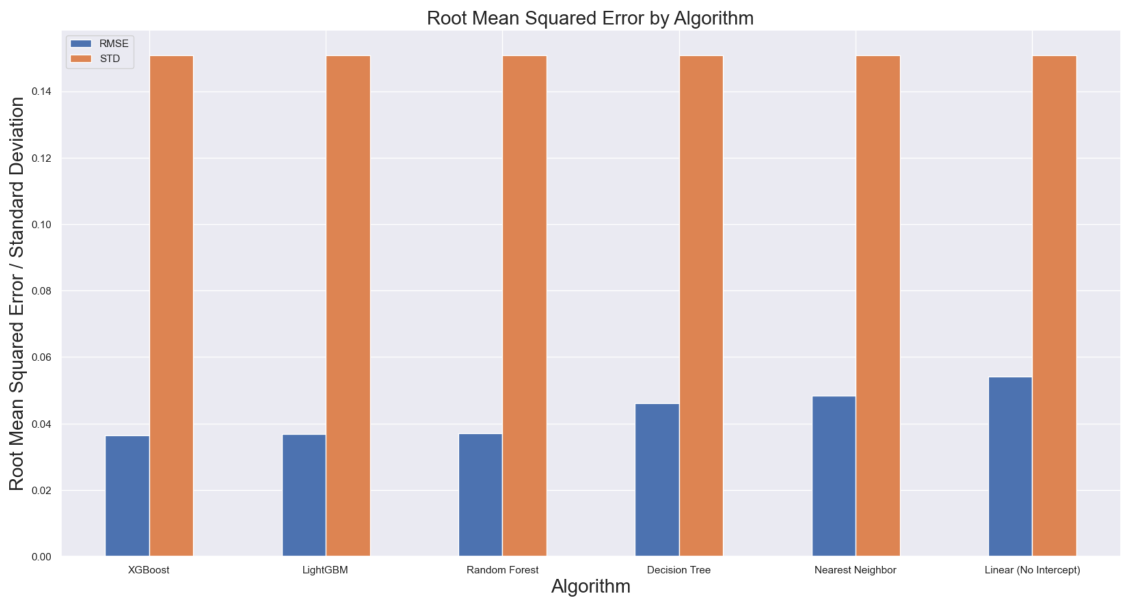
[ ]:
```
# Step 5: Train the Random Forest Regressor
model = RandomForestRegressor()
model.fit(X_train_split, y_train_split)

# Step 6: Make Predictions
y_pred = model.predict(X_test_split)

# Step 7: Evaluate the Model
rf_rmse = np.sqrt(mean_squared_error(y_test_split, y_pred))
rf_std_dev = np.std(y_test_split)
print(f"Random Forest RMSE: {rf_rmse:.4f}, std deviation: {rf_std_dev:.4f}")
```

```
Random Forest RMSE: 0.0371, std deviation: 0.1509
```

[ ]:
```
#plot RMSE and STD for each Algorithm
data = {'Linear (No Intercept)':[lr_no_rmse.mean(),lr_no_std_dev], 'XGBoost':
↪[xgb_rmse.mean(),xgb_stdev], 'Random Forest': [rf_rmse.mean(),rf_std_dev]
       , 'LightGBM': [lgb_rmse.mean(),lgb_std_dev], 'Decision Tree': [dt_rmse.
↪mean(),dt_std_dev],'Nearest Neighbor': [kn_rmse.mean(),kn_std_dev]}
```

```
data_df = pd.DataFrame(data=data).T.reset_index().sort_values(by =␣
↪[0],ascending = True)
data_df.columns = ['Algorithm','RMSE','STD']
```

[ ]:
```
# creating the bar plot
data_df.plot(kind='bar',x = 'Algorithm', y = ['RMSE', 'STD'], figsize =␣
↪(20,10), rot=0)
plt.xlabel("Algorithm",fontsize=20)
plt.ylabel("Root Mean Squared Error / Standard Deviation",fontsize=20)
plt.title("Root Mean Squared Error by Algorithm",fontsize=20)
plt.show()
```



[ ]:
```
flights_df = pd.get_dummies(flights_df)
flights_df.head()
```

[ ]:
```
train_df = flights_df[flights_df.columns.difference(['Unnamed: 0', 'price'])]
```

[ ]:
```
#Validation function
n_folds = 5

def rmse_cv(model,n_folds):
    kf=KFold(n_splits=n_folds)
    rmse = np.sqrt(-cross_val_score(model, train_df, flights_df.price,␣
↪scoring="neg_mean_squared_error", cv = kf))
    return rmse
```

[ ]:
```
lr_w_int = LinearRegression()
lr_no_int = LinearRegression(fit_intercept=False)
```

[ ]:
```
neigh = KNeighborsRegressor(n_neighbors=10)
```

[ ]:
```
rf = RandomForestRegressor(n_estimators=100)
```

[ ]:
```
dt = DecisionTreeRegressor(max_depth = 10)
```

[ ]:
```
model_xgb = xgb.XGBRegressor(max_depth=5, n_estimators=1000, learning_rate=0.01)
```

[ ]:
```
model_lgb = lgb.LGBMRegressor(learning_rate=0.01, max_depth=5,␣
↪n_estimators=1000)
```

Algorithm Results on a 5 Fold Cross Validation

[ ]:
```
score_linear = rmse_cv(lr_w_int,n_folds)
print("Linear Regression (w/ Intercept) score: {:.4f} ({:.4f})\n".
↪format(score_linear.mean(), score_linear.std()))
```

```
Linear Regression (w/ Intercept) score: 0.0517 (0.0048)
```

[ ]:
```
score_linear_no_int = rmse_cv(lr_no_int,n_folds)
print("Linear Regression (No Intercept) score: {:.4f} ({:.4f})\n".
↪format(score_linear_no_int.mean(), score_linear_no_int.std()))
```

```
Linear Regression (No Intercept) score: 0.0517 (0.0047)
```

[ ]:
```
score_neigh = rmse_cv(neigh,n_folds)
print("Nearest Neighbor (13) score: {:.4f} ({:.4f})\n".format(score_neigh.
↪mean(), score_neigh.std()))
```

```
Nearest Neighbor (13) score: 0.1283 (0.0461)
```

[ ]:
```
score_dt = rmse_cv(dt,n_folds)
print("Decision Tree Regression score: {:.4f} ({:.4f})\n".format(score_dt.
↪mean(), score_dt.std()))
```

```
Decision Tree Regression score: 0.0429 (0.0064)
```

[ ]:
```
score_rf = rmse_cv(rf,n_folds)
print("Random Forest Regression score: {:.4f} ({:.4f})\n".format(score_rf.
↪mean(), score_rf.std()))
```

```
Random Forest Regression score: 0.0422 (0.0082)
```

[ ]:
```
score_xg = rmse_cv(model_xgb,n_folds)
print("Xgboost score: {:.4f} ({:.4f})\n".format(score_xg.mean(), score_xg.
 ↪std()))
```

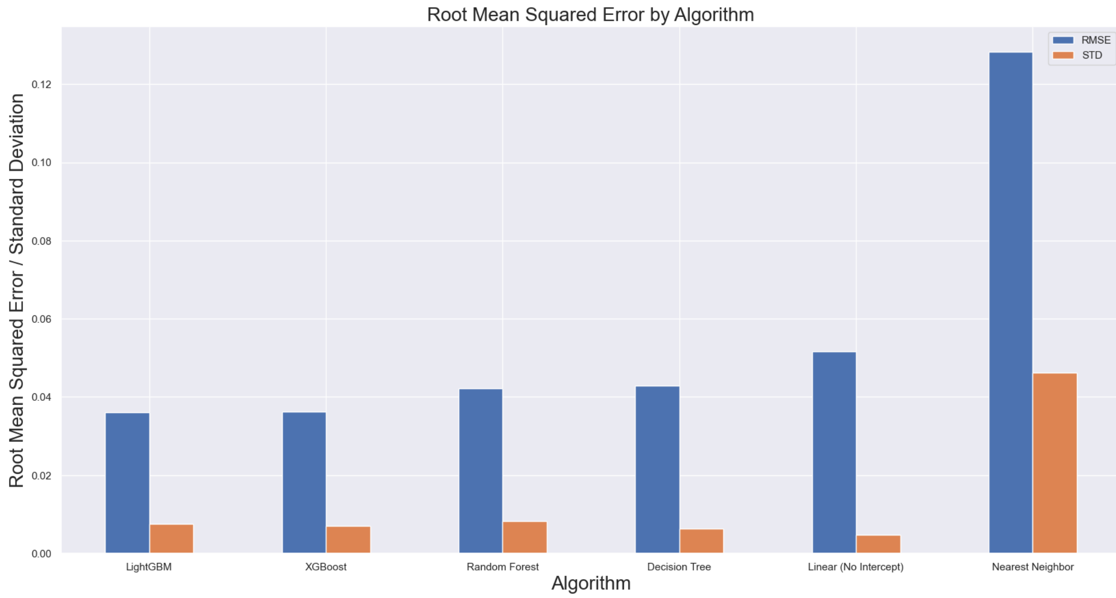
```
Xgboost score: 0.0362 (0.0071)
```

[ ]:
```
score_lgbm = rmse_cv(model_lgb,n_folds)
print("LGBM score: {:.4f} ({:.4f})\n" .format(score_lgbm.mean(), score_lgbm.
 ↪std()))
```

```
LGBM score: 0.0360 (0.0076)
```

[ ]:
```
#plot RMSE and STD for each Algorithm
data = {'Linear (No Intercept)':[score_linear_no_int.mean(),score_linear_no_int.
 ↪std()], 'XGBoost':[score_xg.mean(),score_xg.std()], 'Random Forest':␣
 ↪[score_rf.mean(),score_rf.std()]
        , 'LightGBM': [score_lgbm.mean(),score_lgbm.std()], 'Decision Tree':␣
 ↪[score_dt.mean(),score_dt.std()],'Nearest Neighbor': [score_neigh.
 ↪mean(),score_neigh.std()]}
data_df = pd.DataFrame(data=data).T.reset_index().sort_values(by =␣
 ↪[0],ascending = True)
data_df.columns = ['Algorithm','RMSE','STD']
```

[ ]:
```
# creating the bar plot
data_df.plot(kind='bar',x = 'Algorithm', y = ['RMSE', 'STD'], figsize =␣
 ↪(20,10), rot=0)
plt.xlabel("Algorithm",fontsize=20)
plt.ylabel("Root Mean Squared Error / Standard Deviation",fontsize=20)
plt.title("Root Mean Squared Error by Algorithm",fontsize=20)
plt.show()
```

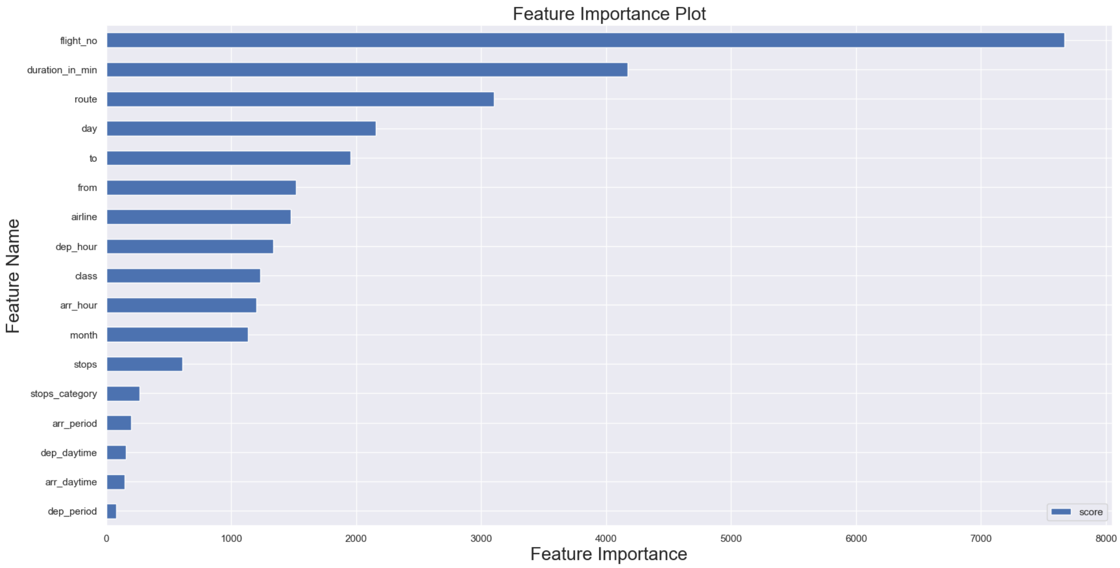


We see that XGBoost has the lowest rmse, but linear (no intercept) has the lowest standard deviation.

1.4.3 Variable Importance Plot

Only applies to tree based models (Decision Trees, Random Forest, GBMs)

[ ]:
```
model = model_xgb.fit(train_df, flights_df.price) #fit model on entire dataset␣
↪to get variable importance since we fit it on each fold
feature_important = model.get_booster().get_score(importance_type='weight')

keys = list(feature_important.keys())
values = list(feature_important.values())

data = pd.DataFrame(data=values, index=keys, columns=["score"]).sort_values(by␣
↪= "score", ascending=False)
data[:20].plot(kind='barh', figsize = (20,10)).invert_yaxis(); ## plot top 20␣
↪features
plt.xlabel("Feature Importance",fontsize=20)
plt.ylabel("Feature Name",fontsize=20)
plt.title("Feature Importance Plot",fontsize=20)
plt.show()
```


Feature Importance Plot
flight_no
duration_in_min
route
day
to
from
airline
dep_hour
class
arr_hour
month
stops
stops_category
arr_period
dep_daytime
arr_daytime
dep_period
Feature Name
0
1000
2000
3000
4000
5000
6000
7000
8000
Feature Importance
score