Sprint - 2: Model - Random Forest Regressor,

- **Model - Random Forest Regressor:**

Visualize data

```
sns.pairplot(df1)
```

Visualize the correlation using heatmap

```
corrmat = df1.corr()
top_corr_features = corrmat.index
plt.figure(figsize = (25,25))
h = sns.heatmap(df1[top_corr_features].corr() , annot=True , cmap='Accent')
```

Divide data into dependent and independent features

```
X = df1.drop(columns='Selling_Price')
y = df1['Selling_Price']
```

Split data into training and testing

```
shuffle = StratifiedShuffleSplit(random_state=51 , test_size=0.2 , n_splits=1)

for train_index , test_index in shuffle.split(df1 , df1['Fuel_Type_Diesel'] ,
df1['Fuel_Type_Petrol']):
    X_train_shuffle = df1.iloc[train_index]
    X_test_shuffle =  df1.iloc[test_index]
X_train_shuffle.shape ,  X_test_shuffle.shape
X_train = X_train_shuffle.drop(columns = 'Selling_Price')
y_train = X_train_shuffle['Selling_Price']
X_test = X_test_shuffle.drop(columns = 'Selling_Price')
```

```
y_test = X_test_shuffle['Selling_Price']
X_train.shape , X_test.shape , y_train.shape , y_test.shape
```

Apply Hyperparameter Tuning on Random Forest Regressor

```
#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)]
# max_depth.append(None)
# 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]
# Create the random grid
random_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}

print(random_grid)
rfr = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
```

```
# search across 100 different combinations

rfr = RandomizedSearchCV(estimator = rfr, param_distributions =
random_grid,scoring='neg_mean_squared_error',

                    n_iter = 10, cv = 5, verbose=2, random_state=51, n_jobs = 1)

rfr.fit(X_train,y_train)
```

Predict data

```
pred = rfr.predict(X_test)
```

Check the r2_score

```
r2_score(pred , y_test)
```

Visualize the actual and predicted

```
plt.scatter(y_test,pred)
```

Display the actual and predicted

```
pd.DataFrame(np.c_[y_test , pred] , columns =['Actual' , 'Predicted'])
```

Save model

```
import pickle

file = open('car_price_prediction_model.pkl','wb')

pickle.dump(rfr,file)
```