COMPANY PROFIT PREDICTION CODE

Importing the packages:

```
import pandas as pd
import numpy as np
import seaborn as sns
import warnings
import matplotlib.pyplot as plt
pd.set_option ( "display.max_columns" , None)
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import GridSearchCV, train_test_split
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
```

Loading the Dataset

```
df = pd.read_csv("50_Startups.csv")
Start_Ups=df
```

Data Cleaning and Checking the null values

```
def Null_Values ( df ) :

    Total = dict( df.isnull ( ).sum ( ) [ df.isnull ( ).sum ( ) > 0 ] )

    if len ( Total ) > 0 :

        List = list ( dict(df.isnull ( ).sum ( ) [ df.isnull ( ).sum ( ) > 0 ] ) )

        Categorical = { i : df [ i ].isnull ( ).sum ( ) for i in list ( df.select_dtypes
( 'object' ) ) }
```

```
        Numeric = { i : df [ i ].isnull ( ).sum ( ) for i in list ( df.select_dtypes ( np.number ) ) }

        print ("The Categorical Null Values are : \n\n{}\n\nThe Numeric Null Values are :
\n\n{}\n\n".format ( Categorical , Numeric ) )

    else :

        print ( "This data set has no null values" )

Null_Values ( df )
```

```
This data set has no null values
```

```
df.head(10)
```

```
df.describe()
```

Non-numeric to Numeric

```
If any Non- numerics are found in the dataset convert it into the numeric values using encoding, here
dataset doesn't contain the non-numeric values
```

Random Sampling

```
def Sampling ( df , Y ) : # Y is the target variable , accepted in string format

    global Train , Test , Train_X , Test_X , Train_Y , Test_Y

    Train , Test = train_test_split( df , test_size = 0.2 , random_state = None ) # Random State =
None ( small dataset )

    Train_X = Train.drop ( columns = [ Y ] )

    Train_Y = Train [ Y ].reset_index ( drop = True  )

    Test_X = Test.drop ( columns = [ Y ] )

    Test_Y = Test [ Y ].reset_index ( drop = True  )

    print ( "The shapes of sampled data sets are (after standard sampling ratio = 0.2 test size ) :
\n" )

    print ( "Train Data = {}\n\nTest Data = {}\n\nTrain_X Data = {}\n\nTrain_Y Data = {}\n\nTest_X
Data = {}\n\nTest_Y Data = {}".format ( Train.shape , Test.shape , Train_X.shape , Train_Y.shape ,
Test_X.shape , Test_Y.shape ) )

Sampling ( df , 'Profit' )
```

Hyper Parameter Turning

```
Parameters = { 'fit_intercept': [ True , False ],

               'copy_X': [True , False ] }

LR = LinearRegression (  )

grid_search = GridSearchCV ( LR , Parameters )
```

```
grid_search.fit ( Train_X , Train_Y )
```

```
print( 'Best hyperparameters : ' , grid_search.best_params_ )
```

Describing the Data

```
def Plots ( DF ) :

    sns.set_palette( "pastel" )

    plt.figure ( figsize =  ( 10 , 10 ) )
```

```
    for i in list ( DF.select_dtypes ( np.number ).columns ) :

        display ( sns.histplot ( x = i , data = DF , bins = 40  )  )
```

```
        plt.show ( )
```

```
        print ( " " )

Plots(Start_Ups)
```

```
  def Statistics ( df ) :

    for i in list ( df.select_dtypes ( np.number ).columns ) :

        global Mean , Med , SD , Kur

        Mean = np.mean ( df [ i ] )
```

```
        Med = np.median ( df [ i ] )
```

```
        SD = np.std ( df [ i ] )

        Kur = df [ i ].kurt ( ) + 3

        print ( '\n\nColumn ----> ', i )

        print ( "\n\nMean = {}\n\nMedian = {}\n\nStandard Deviation = {}\n\nKurtosis = {}".format
( Mean , Med , SD , Kur ) )

        if Mean != Med :

            if Mean > Med :

                print ( "\nRight Skewed" )

            else :

                print ( "\nLeft Skewed" )

        if Kur > 3 :

            print ( "\nThe distribution is Leptokurtic" )

        else :

            print ( "\nThe distribution is Platykurtic\n" )

        print ( "----------------------------------------------------------------" )


Statistics ( Start_Ups )
```

```
  ASSUMPTIONS CHECK
#  MULTICOLLINEARITY
# Multicollinearity occurs when two or more independent variables in a linear regression model are
highly correlated with each other.

#    This can cause problems in the estimation of the model coefficients and the interpretation of the
results.
```

Correlation Matrix

```
Correlation = df.corr ( )

Set_Corr = set ( Correlation.values [ ( Correlation > 0.5) | ( Correlation < -0.5)  ] )
```

```
Set_Corr.remove ( 1.0 )
```

```
sns.color_palette ( 'pastel' )
```

```
sns.heatmap ( df.corr ( ) , cmap = "BuPu" , annot = True )
```

Define Feature and Target Variable

```
X = df[['R&D Spend', 'Administration', 'Marketing Spend']]
y = df['Profit']
```

Split the data into Training and Testing Data

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Linear Regression Modelling

```
 Best hyperparameters :  {'copy_X': True, 'fit_intercept': True}
LR = LinearRegression ( copy_X = True , fit_intercept = True )

LR.fit ( Train_X , Train_Y )
```

```
Model = pd.DataFrame ( )

Model [ 'Predicted_Values' ] = LR.predict ( Train_X )
Model [ 'Actual_Values' ] = Train_Y
Model [ 'Error' ] = Train_Y - Model [ 'Predicted_Values' ]
Model [ 'Absolute_Error' ] = np.abs ( Model.Error )
Model [ 'Error_Percentage' ] = np.abs ( ( Model [ 'Error' ] * 100  ) / Train_Y )
Model = Model.sort_values ( 'Error_Percentage' , ascending = False )
Model
```

Co-Efficients of Variables

High Co-Efficients of Variables mean that the corresponding independent variable has a strong relationship with the dependent variable

```
Coefficients = pd.DataFrame ()

Columns = list(df.columns)
Columns.remove ( 'Profit' )
Coefficients [ 'Values' ]  = LR.coef_
Coefficients [ 'Columns' ] = Columns
Coefficients
```

Co-efficient of determinant

```
R2 = LR.score ( Train_X , Train_Y )

P = Train_X.shape [ 1 ]
N = Train_X.shape [ 0 ]
ADJ_R2 = 1 - ( ( ( 1 - R2) * ( N - 1 ) ) /  ( N - P - 1 ) )
print ( "The coefficient of determinant is : {} and the adjusted value is {}".format ( R2 * 100 ,
ADJ_R2 * 100 ) )
```

Checking for Assumptions

Mean of Error:

```
print ( "The mean of the error is : " , np.mean ( Model.Error ) )
```

```
print ( "The mean of absolute error is : " , np.mean ( Model.Absolute_Error ) )
```

Normal Distribution

The errors are violating the assumptions of Normal distribution

```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )
plt.ylabel ( 'Frequency' )
plt.grid ( )
plt.hist ( x = Model.Error );
```

Heteroscedasticity

```
sns.regplot ( x = Model.Actual_Values , y = Model.Predicted_Values , data = Model );
```

Metrics

MSE:

```
Predicted_Test = LR.predict ( Test_X )

Error_Test = Test_Y - Predicted_Test
Error_test_Percentage = abs(Error_Test) * 100 / Test_Y
MSE = np.mean ( np.square ( Error_Test ) )
print ( "The mean square value of error test is " , MSE )
print ( "\nThe mean square value of error train is " , np.mean ( np.square (Model.Error) ) )
```

RMSE:

```
RMSE = np.sqrt ( MSE )

print ( "The root mean square value of error test is " , RMSE )
print ( "\nroot mean square value of error train is " , np.sqrt ( np.mean ( np.square
( Model.Error) ) ) )
```

MAPE:

```
MAPE = np.mean ( Error_test_Percentage )

print ( "The mean absolute percentage error (of test) is " , MAPE )
print ( "\nThe mean absolute percentage error (of train) is " , np.mean ( Model.Error_Percentage ) )
```

Accuracy:

```
print ( "Accuracy on train data is : " , 100 - MAPE  )

print ( '\nAccuracy on test data is : ' , 100 - np.mean ( Model.Error_Percentage ) )
```

Removing Outliers

We can see the model is performing well on test data and poor on the train data, this is due to the less records on the test data. Another reason could be that the train is influenced by the outliers

```
def Quartiles ( df , Y , K ) :

    global Lower , Upper , Lower_Outliers , Upper_Outliers

    Lower = df [ Y ].quantile ( q = 0.25 )

    Upper = df [ Y ].quantile ( q = 0.75 )
    IQR = Upper - Lower
    Lower_Outliers =  Lower - K * IQR
    Upper_Outliers =  Upper + K * IQR
    df.drop ( labels = list ( df [ df [ Y ] >= Upper_Outliers ].index ) , inplace = True )
    df.drop ( labels = list ( df [ df [ Y ] <= Lower_Outliers ].index ) , inplace = True )

    df.reset_index ( drop = True , inplace = True )
Quartiles ( df , 'Profit' , 0.75 ) ## K = 1 becasue the data is not vast
```

```
LR = LinearRegression ( copy_X = True , fit_intercept = True )

LR.fit ( Train_X , Train_Y )
Model = pd.DataFrame ( )
Model [ 'Predicted_Values' ] = LR.predict ( Train_X )
Model [ 'Actual_Values' ] = Train_Y
Model [ 'Error' ] = Train_Y - Model [ 'Predicted_Values' ]
Model [ 'Absolute_Error' ] = np.abs ( Model.Error )
Model [ 'Error_Percentage' ] = np.abs ( ( Model [ 'Error' ] * 100  ) / Train_Y )
Model = Model.sort_values ( 'Error_Percentage' , ascending = False )
Model
```

Co-Efficients of Determinant

```
R2 = LR.score ( Train_X , Train_Y )

P = Train_X.shape [ 1 ]
N = Train_X.shape [ 0 ]
ADJ_R2 = 1 - ( ( ( 1 - R2) * ( N - 1 ) ) /  ( N - P - 1 ) )
print ( "The coefficient of determinant is : {} and the adjusted value is {}".format ( R2 * 100 ,
ADJ_R2 * 100 ) )
```

Mean

```
R2 = LR.score ( Train_X , Train_Y )

P = Train_X.shape [ 1 ]
```

```
N = Train_X.shape [ 0 ]
ADJ_R2 = 1 - ( ( ( 1 - R2) * ( N - 1 ) ) /  ( N - P - 1 ) )
print ( "The coefficient of determinant is : {} and the adjusted value is {}".format ( R2 * 100 ,
ADJ_R2 * 100 ) )
```

Normal Distribution

```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )
plt.ylabel ( 'Frequency' )
plt.grid ( )
plt.hist ( x = Model.Error );
```

Metrics

```
Predicted_Test = LR.predict ( Test_X )

Error_Test = Test_Y - Predicted_Test
Error_test_Percentage = abs(Error_Test) * 100 / Test_Y
MSE = np.mean ( np.square ( Error_Test ) )
print ( "\nThe mean square value of error test is " , MSE )
print ( "\nThe mean square value of error train is " , np.mean ( np.square (Model.Error) ) )
#### RMSE
RMSE = np.sqrt ( MSE )
print ( "\nThe root mean square value of error test is " , RMSE )
print ( "\nThe root mean square value of error train is " , np.sqrt ( np.mean ( np.square
( Model.Error) ) ) )
#### MAPE
MAPE = np.mean ( Error_test_Percentage )
print ( "\nThe mean absolute percentage error (of test) is " , MAPE )
print ( "\nThe mean absolute percentage error (of train) is " , np.mean ( Model.Error_Percentage ) )
#### ACCURACY
print ( "\nAccuracy on train data is : " , 100 - MAPE  )
print ( '\nAccuracy on test data is : ' , 100 - np.mean ( Model.Error_Percentage ) )
```

Define Models

```
models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=1.0),
    "Random Forest Regressor": RandomForestRegressor(n_estimators=100, random_state=42)
}
```

Train and Evaluate Models

```
results = {}
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    results[name] = {
        "R2 Score": r2_score(y_test, y_pred),
        "MAE": mean_absolute_error(y_test, y_pred),
        "MSE": mean_squared_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test, y_pred))
    }
```

Print Results

```
        for model, metrics in results.items():
    print(f"{model}:")
    for metric, value in metrics.items():
        print(f"  {metric}: {value:.4f}")
    print()
```

Random forest Regressor achieved the highest R^2 score (0.9103) and the lowest error matrics so making it into the best model for predicting company profit

Train Model

```
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

Function to predict the profit using the best model (Random forest Regressor)

```
def predict_profit(rnd_spend, admin, marketing):
    best_model = models["Random Forest Regressor"]
    input_data = np.array([[rnd_spend, admin, marketing]])
    prediction = best_model.predict(input_data)
    return prediction[0]
```

Predicting the profit

```
rnd_spend = float(input("Enter R&D Spend: "))
admin = float(input("Enter Administration Cost: "))
marketing = float(input("Enter Marketing Spend: "))
```

```
predicted_profit = predict_profit(rnd_spend, admin, marketing)
print(f"Predicted Profit: {predicted_profit:.2f}")
```