DATA SCIENCE PROJECT REPORT

(Project Term October-Nov 2024)

Profit Prediction Model for Startups

Submitted by

Somishetty Sanjay Varma

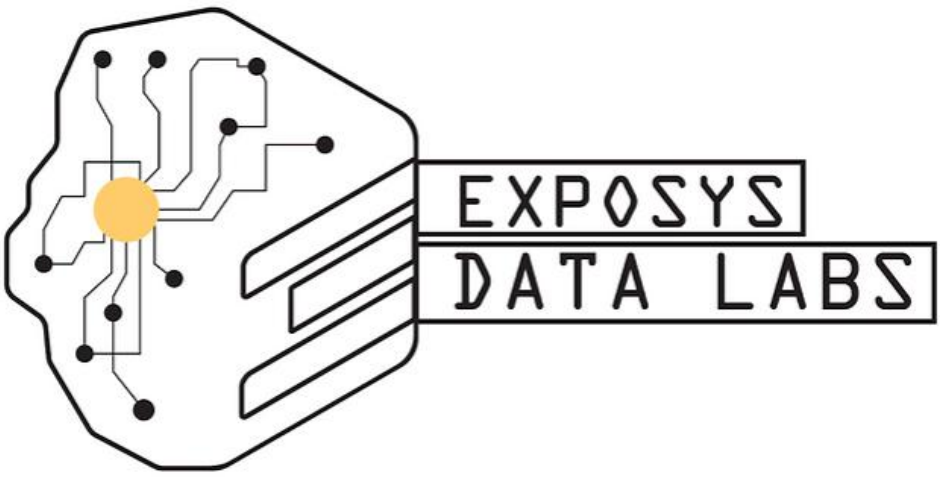


Table of Contents:

Abstract:

The Profit Prediction for Startups project aims to develop a machine learning model capable of accurately predicting the profit of startup companies based on multiple financial and locational variables. Specifically, the model leverages data on R&D expenditure, administrative costs, marketing spend, and the geographical location (state) of each startup to provide actionable profit forecasts. By analyzing the influence of these diverse factors, the model highlights key drivers of startup profitability and enables a deeper understanding of how spending patterns and location impact financial outcomes. This predictive tool is designed to offer entrepreneurs, investors, and stakeholders valuable insights to support strategic decision-making and resource allocation in the early stages of business ventures. With potential applications in financial planning and investment assessment, this model aims to contribute to informed, data-driven growth strategies in the startup ecosystem. By leveraging regression algorithms, the model uncovers the extent to which each variable influences profitability, highlighting key drivers and potential areas for optimization. In addition to serving as a powerful forecasting tool, this model can assist startups in benchmarking their financial strategies against industry standards, encouraging data-driven adjustments for enhanced profitability. The resulting insights have the potential to inform various stakeholders—from founders and venture capitalists to financial analysts and advisors—about the likely returns on investment under differing business scenarios.

Introduction:

The Profit Prediction for Startups project aims to build a machine learning model that predicts the profit of startups by analyzing critical spending and locational factors. For new ventures, understanding the financial impact of R&D, administrative, and marketing expenditures is essential, as these areas directly influence growth and profitability the dataset includes key variables: R&D Spending, representing investment in innovation and product development; Administration Spending, covering management and operational costs; Marketing Spending, denoting promotional expenses; and State, indicating the startup's location. These elements together offer a comprehensive view of factors that could influence profit outcomes. By examining the relationships between these variables and profit, this project seeks to create a reliable predictive model that aids startups and investors. Entrepreneurs can leverage these insights to make more strategic budgetary decisions, while investors can assess a startup’s financial potential based on its spending patterns and geographic location. This analysis holds great potential for startups seeking to optimize resource allocation and maximize profit. Additionally, the model can assist investors and advisors in evaluating a startup's financial outlook based on its spending strategy and location, aiding in investment and advisory decisions. Ultimately, this project endeavors to contribute to the startup ecosystem by equipping stakeholders with data-driven insights that foster more effective and profitable business strategies.

Overall, this model aims to be a valuable tool within the startup ecosystem, supporting data-driven decisions that enhance profitability and long-term success. The insights gained from this project are expected to empower startups to optimize resource allocation and achieve a sustainable competitive advantage.

Existing Method:

For a project like Profit Prediction for Startups, there are several established methods and approaches in machine learning and data analysis that are typically employed to build accurate predictive models. Here's an overview of common methods:

1. Data Preprocessing and Exploration

- **Data Cleaning**: Handle any missing or inconsistent data.
- **Encoding Categorical Variables**: Use methods such as one-hot encoding or label encoding to convert categorical data (like 'State') into numerical format for machine learning algorithms.
- **Feature Scaling**: Standardize or normalize the features to ensure uniform scales, especially important for algorithms sensitive to feature magnitude, like regression models.

2. Exploratory Data Analysis (EDA)

- **Statistical Analysis**: Evaluate the statistical properties of each feature and its correlation with the target variable (profit).
- **Visualization**: Use scatter plots, pair plots, and heatmaps to visualize relationships between features (e.g., R&D spending vs. profit) to gain insight into data distribution and potential patterns.

3. Regression Algorithms

Several regression methods can be applied to predict profit:

- **Multiple Linear Regression**: A simple yet effective model that predicts the target variable by fitting a linear relationship with multiple independent variables (R&D, administration, marketing spending).
- **Polynomial Regression**: Useful when the relationship between variables is nonlinear; adds polynomial terms to improve model accuracy.
- **Decision Tree Regression**: Nonlinear model that splits data into branches based on feature values, effective for capturing complex interactions.
- **Random Forest Regression**: An ensemble of decision trees that improves predictive accuracy by reducing overfitting, often yielding more robust predictions.
- **Support Vector Regression (SVR)**: Suitable for high-dimensional spaces and effective when a clear margin of separation exists between profit levels.
- **Neural Networks**: More complex models that may be applied for capturing intricate patterns in data, though they may require larger datasets.

4. Model Evaluation Metrics

To assess the performance of each model and choose the best one, several regression metrics are commonly used:

- **Mean Absolute Error (MAE)**: Measures the average magnitude of errors without considering their direction.
- **Mean Squared Error (MSE)**: Penalizes larger errors by squaring them, making it useful for models where large errors are less desirable.
- **Root Mean Squared Error (RMSE)**: Provides error in the same units as the target variable, helping with interpretation.

- **R-squared (R^2)**: Indicates the proportion of the variance in the target variable that is predictable from the features; higher values indicate a better fit.

5. Model Selection and Tuning

- **Model Comparison**: Evaluate and compare each model based on the calculated metrics.
- **Hyperparameter Tuning**: Use techniques like grid search or random search to optimize hyperparameters for algorithms like random forests or neural networks to improve predictive accuracy.
- **Cross-Validation**: Validate model performance on multiple data splits to avoid overfitting and ensure generalizability.

6. Deployment and Insights Generation

- Once the best model is selected, it can be deployed as a predictive tool. Additionally, interpreting feature importance (e.g., which spending area has the highest impact on profit) provides actionable insights for stakeholders, guiding strategic decisions.

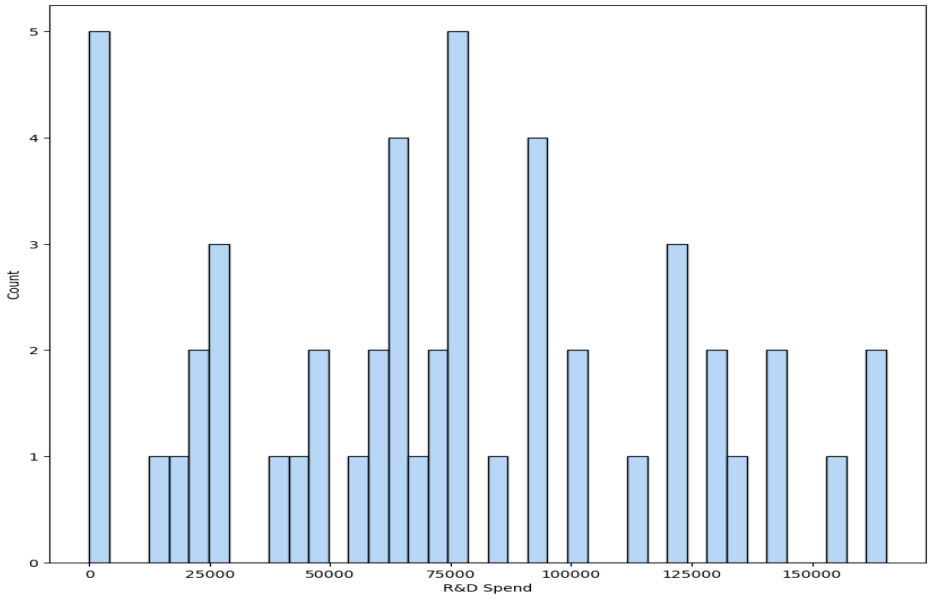
Using these existing methods provides a structured approach to develop an accurate, interpretable, and robust profit prediction model for startups.

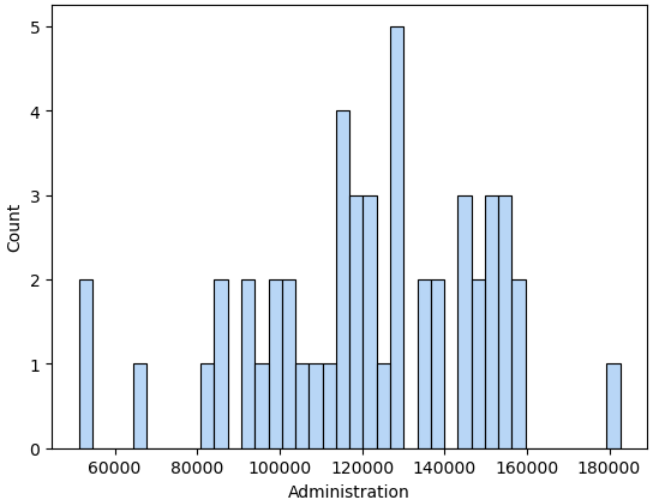
Proposed Method and Architecture:

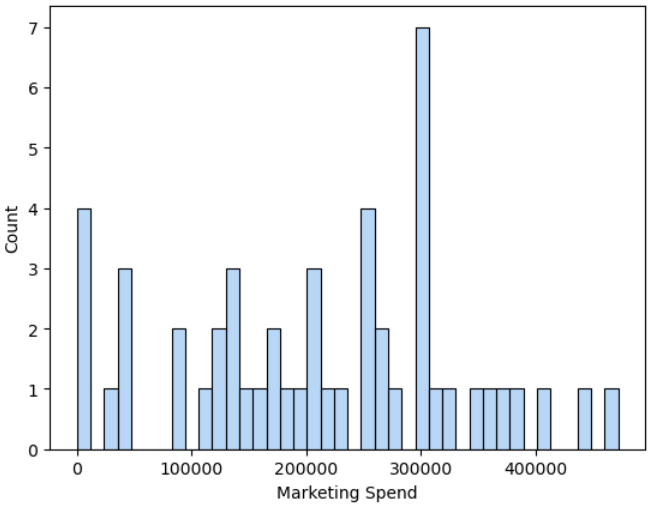
The proposed method involves building a **profit prediction model** using a structured approach that integrates data preprocessing, model selection, evaluation, and deployment. Here's a detailed breakdown of the proposed method along with a suggested architecture:

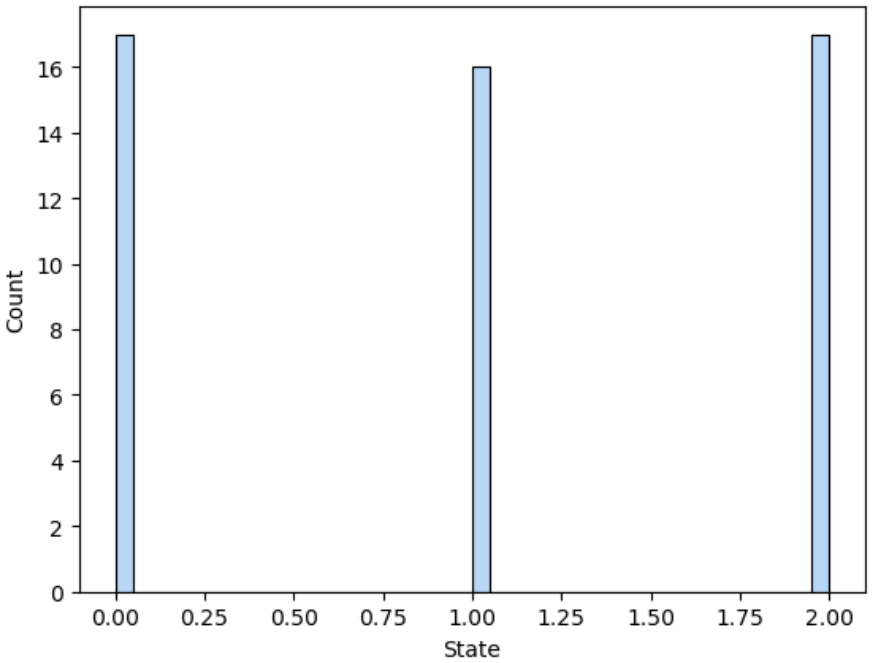
1. Data Preprocessing Layer

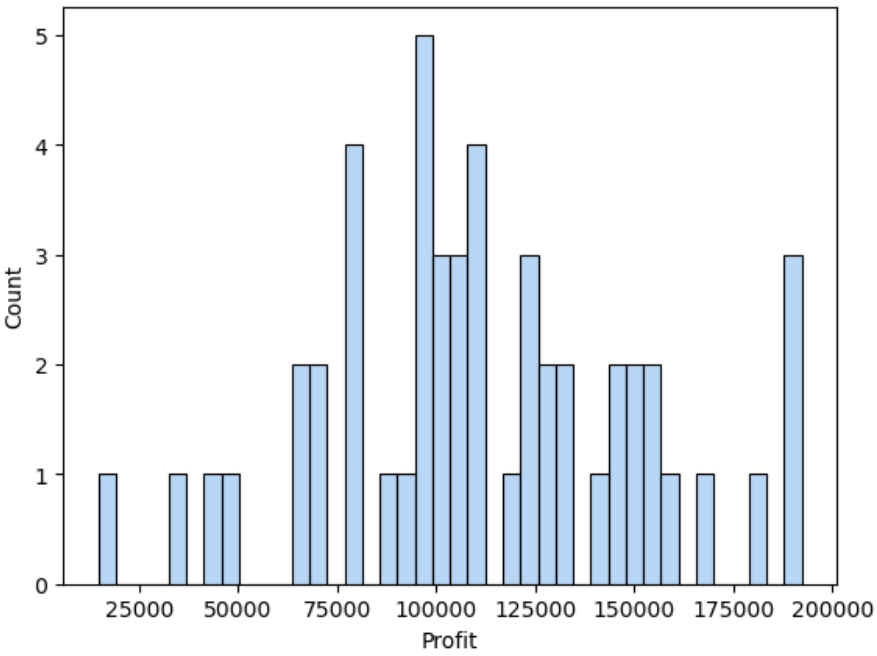
- **Data Cleaning**: Remove or impute any missing values and handle outliers to ensure clean data.
- **Feature Encoding**: Convert the categorical variable, *State*, into a numerical format using **One-Hot Encoding** or **Label Encoding**.
- **Feature Scaling**: Standardize or normalize numerical features (R&D, administration, marketing spending) to improve algorithm performance, particularly for models sensitive to feature scale (e.g., Support Vector Regression).




Count
Administration
0
1
2
3
4
5
60000
80000
100000
120000
140000
160000
180000


Count
Marketing Spend
0
1
2
3
4
5
6
7
0
100000
200000
300000
400000


Count
State
0
2
4
6
8
10
12
14
16
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00


Count
Profit
0
1
2
3
4
5
25000
50000
75000
100000
125000
150000
175000
200000

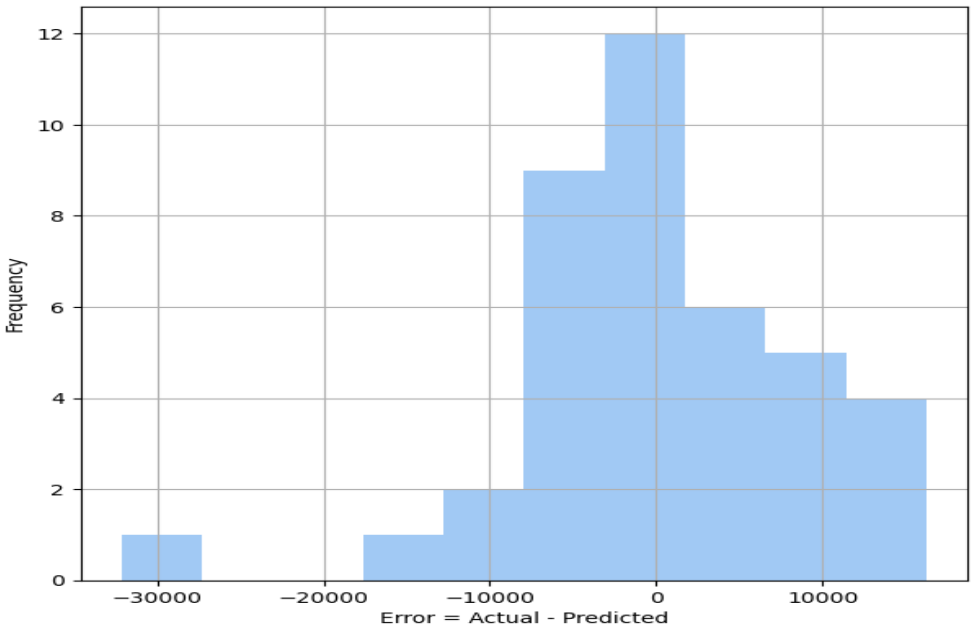
2. Exploratory Data Analysis (EDA) Layer

- **Correlation Analysis**: Calculate correlation coefficients to understand relationships between spending areas and profit.
- **Visualization**: Generate heatmaps, scatter plots, and box plots to visualize relationships and distributions among features.
- **Feature Selection**: Use EDA insights to potentially exclude any irrelevant or highly correlated features, which may reduce noise in the data and improve model performance.

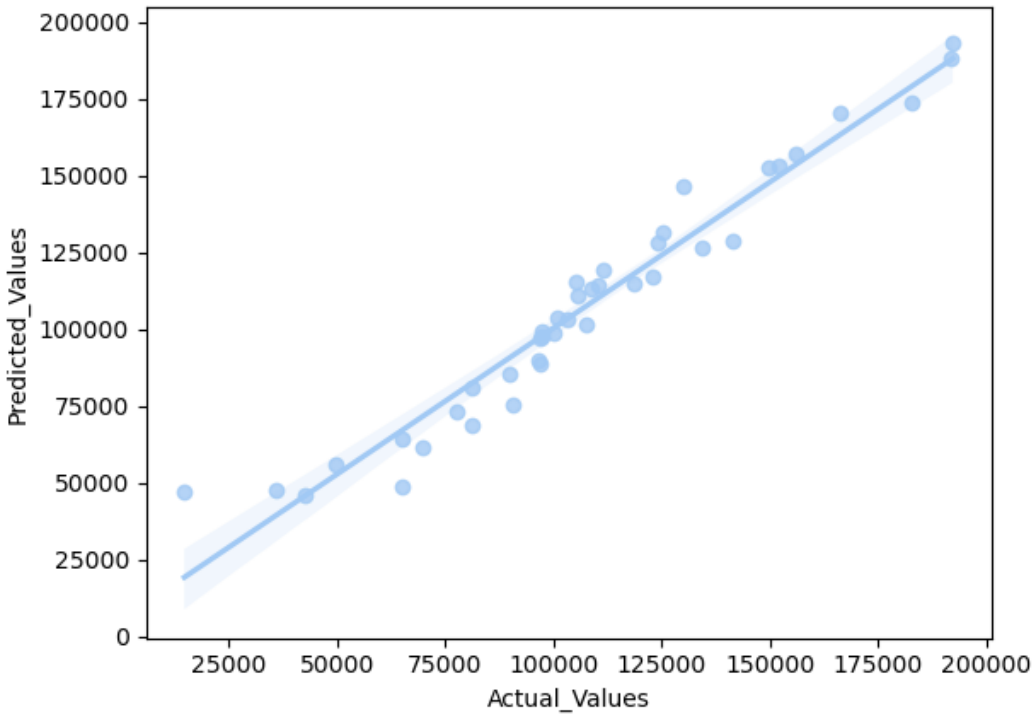
Correlation Matrix

3. Model Construction Layer

- **Multiple Regression Models**: Construct different regression models to compare performance:
    - **Linear Regression**: For a baseline model, assuming a linear relationship between spending and profit.
    - **Polynomial Regression**: Capture non-linear relationships by adding polynomial terms.
    - **Decision Tree Regression**: Handle non-linear relationships and feature interactions.
    - **Random Forest Regression**: Ensemble method that aggregates multiple decision trees for a more robust model.
    - **Gradient Boosting Machines (GBM)**: Another ensemble approach that incrementally builds models, focusing on reducing residual errors.
    - **Support Vector Regression (SVR)**: Effective in high-dimensional spaces; useful for a more refined margin-based approach.
    - **Neural Networks (Optional)**: If the dataset is extensive enough, a simple feedforward neural network may also be constructed to capture complex interactions between features.



Normal Distribution



HETEROSCEDASTICITY

4. Training and Validation Layer

- **Train-Test Split**: Split the data into training and testing sets (e.g., 80%-20% split) for initial training and performance evaluation.
- **Cross-Validation**: Perform k-fold cross-validation to improve model reliability and reduce overfitting by training and testing the model across multiple data folds.
- **Hyperparameter Tuning**: Use **Grid Search** or **Random Search** techniques to optimize model parameters, especially for complex models like Random Forest or Gradient Boosting.

5. Evaluation Layer

- **Model Evaluation Metrics**:
    - **Mean Absolute Error (MAE)**: Measures the average absolute difference between predicted and actual profit values.
    - **Mean Squared Error (MSE)**: Gives greater weight to larger errors by squaring them.
    - **Root Mean Squared Error (RMSE)**: Similar to MSE but in the same units as the target variable, making it easier to interpret.

    - **R-squared (R^2)**: Shows the proportion of variance explained by the model.
- **Model Selection**: Compare models based on these metrics and select the best-performing model (lowest error metrics and highest R^2).

6. Deployment Layer

- **Model Interpretation**: Use feature importance metrics to understand which spending areas most impact profit. For instance, a high importance score on R&D spending might suggest that innovation investments strongly influence profitability.
- **API or Web Application**: Deploy the model using a framework such as Flask or FastAPI to create an API, or integrate it into a web application for real-time profit prediction.
- **User Interface (Optional)**: Provide an intuitive interface for users (e.g., entrepreneurs and investors) to input spending values and receive profit predictions with insights on influential factors.

Proposed Architecture Diagram

1. **Input Layer**: User inputs values for R&D Spending, Administration Spending, Marketing Spending, and State.
2. **Preprocessing Layer**: Encodes categorical data, scales numerical features, and cleans data.
3. **Feature Analysis and Selection**: Conducts correlation analysis, visualizations, and feature selection.
4. **Modeling Layer**: Trains multiple regression models (Linear, Decision Tree, Random Forest, etc.), performs cross-validation, and tunes hyperparameters.
5. **Evaluation Layer**: Compares models using error metrics (MAE, MSE, RMSE, R^2) to select the best model.
6. **Deployment Layer**:
    - **API or Web Interface**: Receives user inputs and returns profit predictions.
    - **Insights Dashboard** (optional): Provides visual insights into how each spending area affects profit.

Summary

The proposed method leverages a multi-layered architecture with distinct steps for data processing, model training, evaluation, and deployment, creating a structured pathway from raw data to actionable insights. This architecture ensures a reliable and interpretable model that accurately predicts startup profit while providing insights into spending effectiveness.

Methodology:

The methodology for the Profit Prediction for Startups project involves a systematic, step-by-step approach from data collection and preprocessing through model selection, evaluation, and deployment. This methodology is structured to ensure that the model is accurate, interpretable, and robust. Here's a detailed breakdown:

1. Data Collection and Understanding

- **Dataset Description**: Analyze the dataset to understand each variable, including R&D Spending, Administration Spending, Marketing Spending, State, and Profit.
- **Data Inspection**: Examine data types, check for any missing values or anomalies, and understand the distribution of each feature.

2. Data Preprocessing

- **Data Cleaning**: Handle any missing values or outliers. Missing values could be handled through mean/mode imputation, or, if they are minimal, the rows with missing values might be removed.
- **Feature Encoding**: Convert categorical data (e.g., *State*) into numerical form using techniques like **One-Hot Encoding** to ensure compatibility with machine learning algorithms.
- **Feature Scaling**: Apply scaling methods such as **Standardization** or **Normalization** to standardize the range of numerical features (e.g., R&D, Administration, Marketing Spending).

3. Exploratory Data Analysis (EDA)

- **Descriptive Statistics**: Calculate mean, median, variance, and other statistical properties of each feature to understand data distribution.
- **Correlation Analysis**: Examine the relationships between each independent variable and the target variable (Profit) using correlation coefficients. This helps in identifying which variables may be more influential.
- **Visual Analysis**: Generate visualizations like scatter plots, pair plots, and heatmaps to gain insights into potential patterns and relationships between variables.

4. Feature Engineering (Optional)

- **Interaction Terms**: Create interaction terms or polynomial features if there is evidence that interactions between spending areas impact profit.
- **Dimensionality Reduction**: If the dataset is high-dimensional, use techniques like **PCA (Principal Component Analysis)** to reduce dimensions while retaining most of the data's variance.

5. Model Development

- **Selecting Algorithms**: Implement various regression algorithms to compare performance:
    - **Multiple Linear Regression**: Provides a baseline with a simple linear relationship between features and profit.
    - **Polynomial Regression**: Used if relationships appear nonlinear.
    - **Decision Tree Regression**: Captures complex, non-linear patterns by splitting data on different feature values.
    - **Random Forest Regression**: An ensemble model that averages multiple decision trees to improve robustness and reduce overfitting.
    - **Gradient Boosting**: Builds an ensemble of weak learners sequentially to reduce residual errors.
    - **Support Vector Regression (SVR)**: Can capture complex relationships with a margin of error that adjusts based on input data.

6. Model Training and Cross-Validation

- **Train-Test Split**: Divide the data into training and testing sets (e.g., 80% for training and 20% for testing) to evaluate model performance on unseen data.
- **Cross-Validation**: Use k-fold cross-validation to enhance model robustness by training and testing the model across multiple data splits. This helps ensure the model generalizes well to new data.

- **Hyperparameter Tuning**: Use techniques such as **Grid Search** or **Random Search** to optimize model hyperparameters, especially for complex models like Random Forest or Gradient Boosting.

7. Model Evaluation

- **Evaluation Metrics**: Evaluate each model's performance using multiple regression metrics:
  - **Mean Absolute Error (MAE)**: Measures the average magnitude of prediction errors.
  - **Mean Squared Error (MSE)** and **Root Mean Squared Error (RMSE)**: Penalize larger errors, providing a clearer measure of prediction accuracy.
  - **R-squared (R^2)**: Indicates how well the model explains the variance in the profit variable; higher values show better performance.
- **Model Comparison**: Compare the performance of different models based on these metrics to identify the best-performing model.

8. Model Interpretation and Insights Generation

- **Feature Importance Analysis**: Use methods like permutation importance or model-specific interpretability methods (e.g., feature importance in Random Forest) to identify which spending areas most impact profit.
- **Insights for Stakeholders**: Generate insights to help stakeholders (e.g., entrepreneurs, investors) understand how each factor affects profitability, aiding in decision-making.

9. Model Deployment

- **Deployment as an API**: Develop a REST API (using frameworks like Flask or FastAPI) to make the model accessible for real-time profit prediction based on user input.
- **User Interface (Optional)**: Create a simple web interface or dashboard for non-technical users to input values and obtain profit predictions.
- **Integration**: The model can be integrated into a larger decision-support tool, providing ongoing insights and predictive analytics.

10. Monitoring and Model Maintenance

- **Performance Monitoring**: Track the model's accuracy over time, especially if deployed in a live environment.

- **Model Retraining**: Regularly update and retrain the model with new data to maintain accuracy, particularly if there are changes in market dynamics or spending trends in startups.

Summary

This methodology emphasizes a structured approach, from data preparation to model deployment, ensuring the profit prediction model is both accurate and actionable. By focusing on key steps such as feature engineering, model selection, cross-validation, and interpretability, this project aims to deliver a reliable, scalable solution that can be effectively used by startups and investors to inform strategic financial decisions.

Implementation:

IMPORTING REQUIRED LIBRARIES

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

pd.set_option ( "display.max_columns" , None)

from sklearn.preprocessing import LabelEncoder

from sklearn.metrics import mean_squared_error

from sklearn.linear_model import LinearRegression

from sklearn.model_selection import GridSearchCV, train_test_split

from statsmodels.stats.outliers_influence import
variance_inflation_factor
```

IMPORTING DATA (CSV)

In [2]:

```
DF = pd.read_csv ( r"A:\PANDAS DATA\DATA SETS\50 STARTUPS\50_Startups.csv"
)

Start_Ups = DF
```

DATA CLEANING

CHECKING FOR NULL VALUES

In [3]:

```
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 )
```

NON-NUMERIC TO NUMERIC

The following function is used for the data frame, where all columns are nominal.

In [4]:

```
LE = LabelEncoder ( )


def Encoding ( DF ) :


    Columns = list ( DF.select_dtypes ( 'object' ) )


    DF.loc [ : , Columns ] = DF.loc [ : , Columns ].apply (
LE.fit_transform )


    display (DF)
```

```
Encoding ( DF )
```

RANDOM SAMPLING

```
In [5]:
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 TUNING

```
In [6]:
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 DATA

```
In [7]:
```

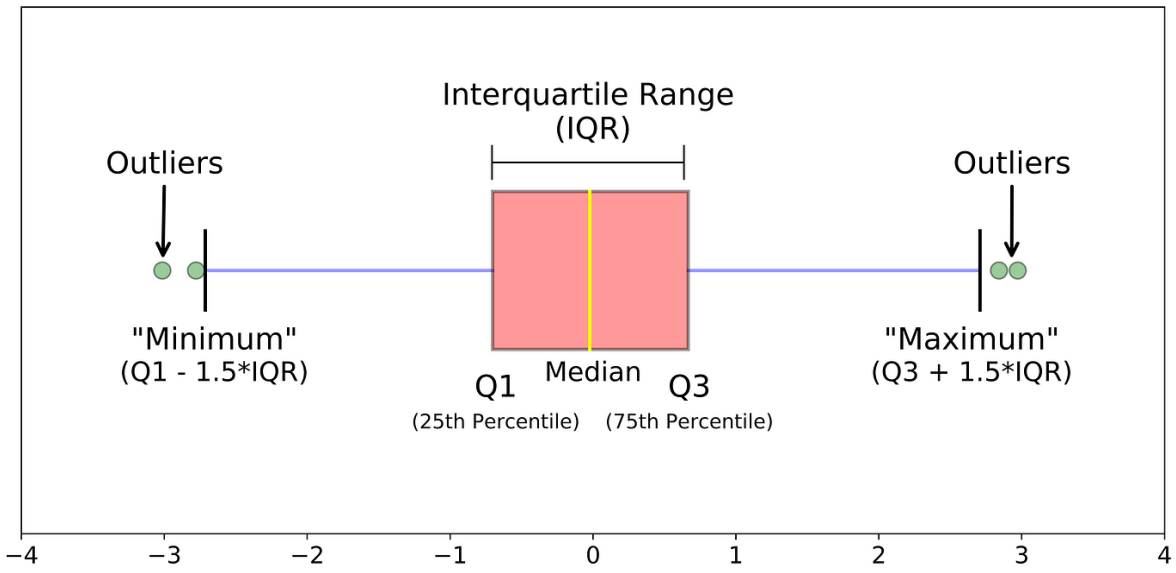
```
def Plots ( DF ) :

    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 )
```

We can see that the distributions are platykurtic / leptokurtic, therefore the data is not normally distributed.

REMOVING OUTLIERS

DEFINING UPPER AND LOWER LIMIT OF QUARTILES



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.

METHOD 1 : 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 )
```

METHOD 2 : VIF

The VIF measures how much the variance of the estimated regression coefficient is inflated due to the presence of

multicollinearity.

A high VIF value indicates that the variance of the coefficient estimate is significantly larger than it would be

if the predictor variable were not correlated with the other predictor variables in the model.

1. We can see that in the presence of multicollinearity, the coefficient of Research & Development is varying the most.

```
A = DF.iloc [ : , : - 1 ]

VIF = pd.DataFrame ( )

VIF [ 'Variables' ] = A.columns

VIF [ 'VIF_Values' ] = [ variance_inflation_factor ( A.values, i ) for i in range( A.shape[1] ) ]

VIF.sort_values ( 'VIF_Values' , ascending = False )
```

LINEAR REGRESSION MODELING

```
        Best hyperparameters :  {'copy_X': True, 'fit_intercept': True}
In [11]:
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
```

COEFFICIENTS OF VARIABLES

High coefficients of variables mean that the corresponding independent variable has a strong relationship with

the dependent variable.

```
In [13]:
Coefficients = pd.DataFrame ()

Columns = list(DF.columns)
```

```
Columns.remove ( 'Profit' )

Coefficients [ 'Values' ]  = LR.coef_

Coefficients [ 'Columns' ] = Columns

Coefficients
```

COEFFICIENT OF DETERMINANT

R2 value tells us how much variance the model is able to capture (SSR)

$$Adjusted\ R^2 = 1 - \frac{(1 - R^2)(N - 1)}{N - p - 1}$$

Where

R^2 Sample R-Squared

N Total Sample Size

p Number of independent variable

```
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

```
In [15]:
print ( "The mean of the error is : " , np.mean ( Model.Error ) )
The mean of the error is :  -1.7280399333685637e-11
In [16]:
print ( "The mean of absolute error is : " , np.mean ( Model.Absolute_Error ) )
```

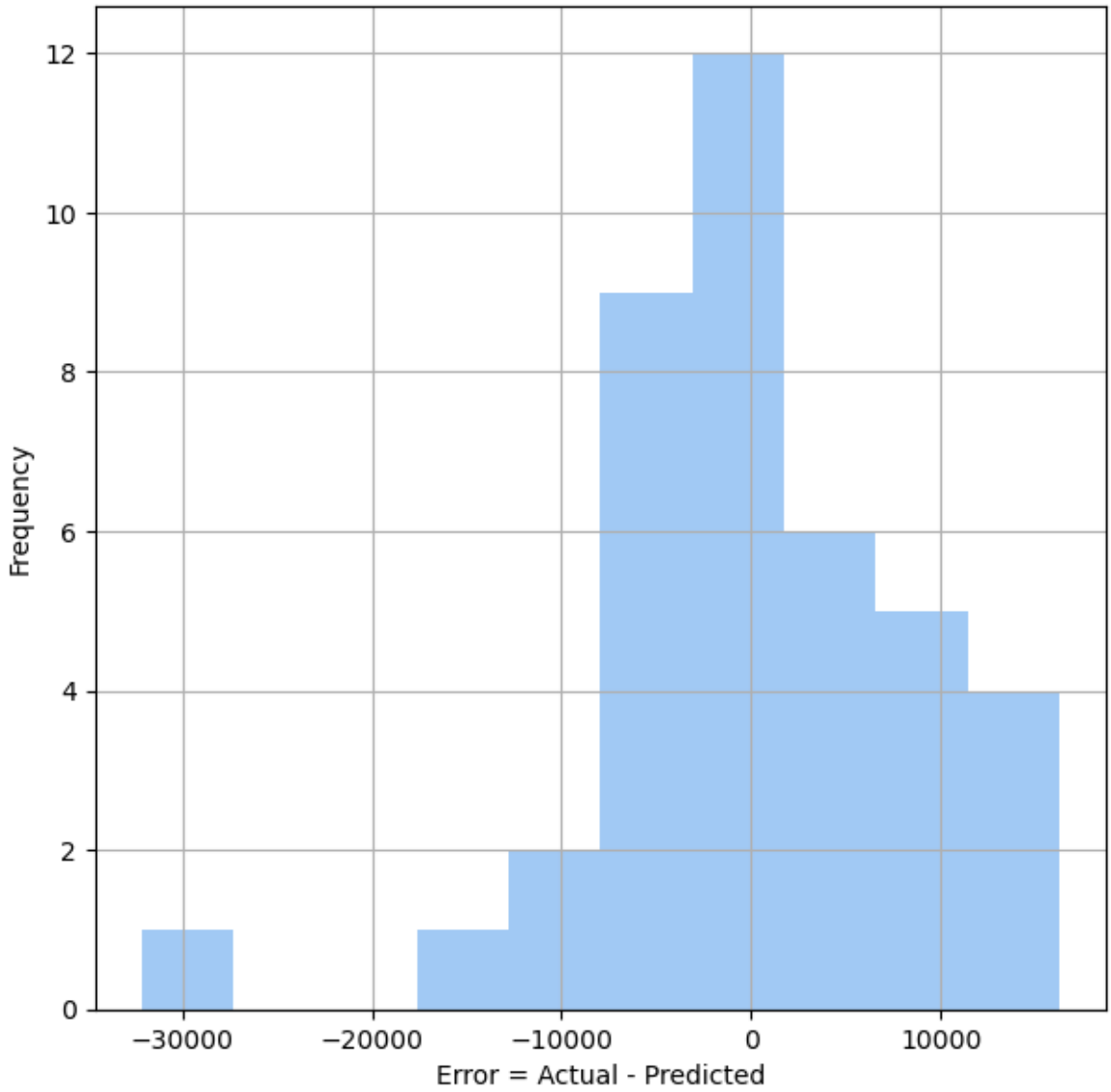
NORMAL DISTRIBUTION

The Errors are violating the assumption of Normal Distribution.

The graph suggests that the kurtosis is higher than 3

In [17]:

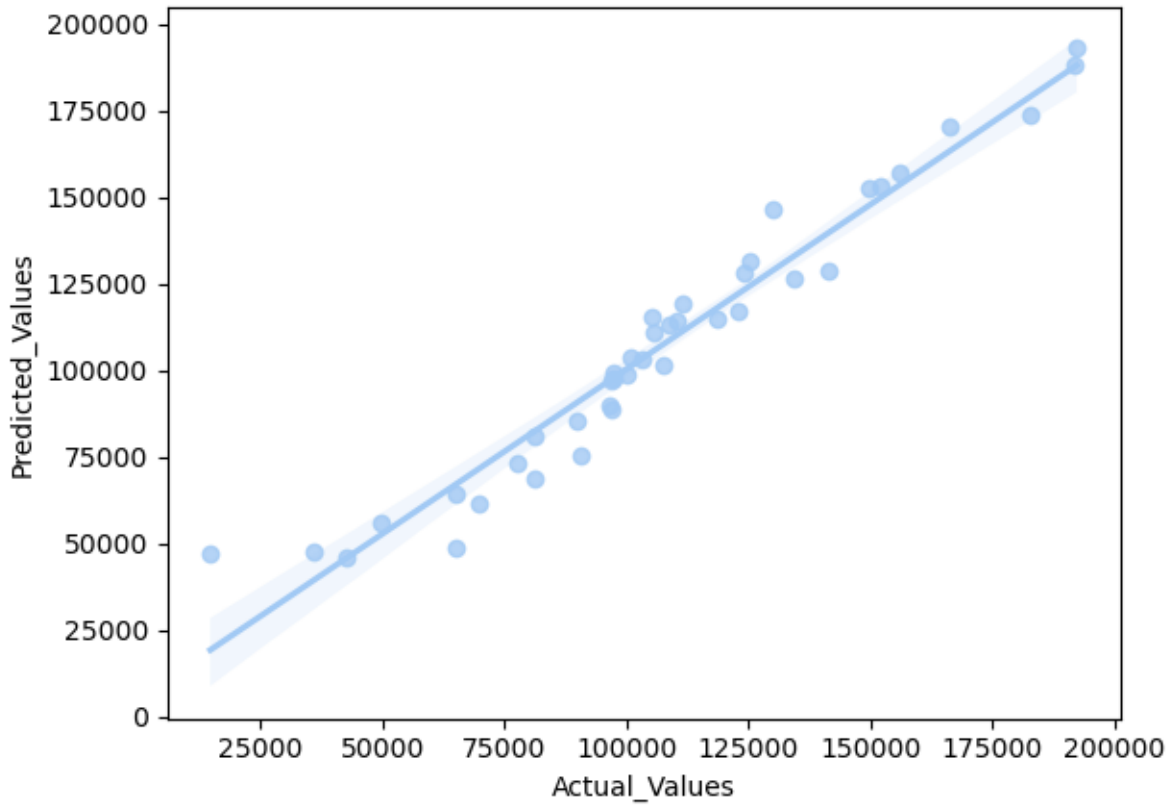
```
plt.figure ( figsize = ( 7 , 7 ) )

plt.xlabel ( 'Error = Actual - Predicted' )

plt.ylabel ( 'Frequency' )

plt.grid ( )

plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [18]:

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



METRICS

MSE

In [19]:

```
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) ) )
```

```
The mean square value of error test is  91270989.63188198


The mean square value of error train is  77776787.94248608
```

RMSE

In [20]:

```
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

In [21]:

```
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 ) )
```

```
The mean absolute percentage error (of test) is  6.142368258032201


The mean absolute percentage error (of train) is  11.677243912327064
```

ACCURACY

In [22]:

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


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

```
Accuracy on train data is :  93.8576317419678


Accuracy on test data is :  88.32275608767293
```

REMOVING OUTLIERS

We can see that the model is performing well on Test data and poor on train data, this is due to less records

in Test Data

Another reason could be that the Train data is influenced by outliers

In [23]:

```
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
```

COFFEICIENTS OF DETERMINANT

In [25]:

```
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 ) )
```

```
The coefficient of determinant is : 95.05035861084991 and the adjusted
value is 94.48468530923276
```

MEAN

In [26]:

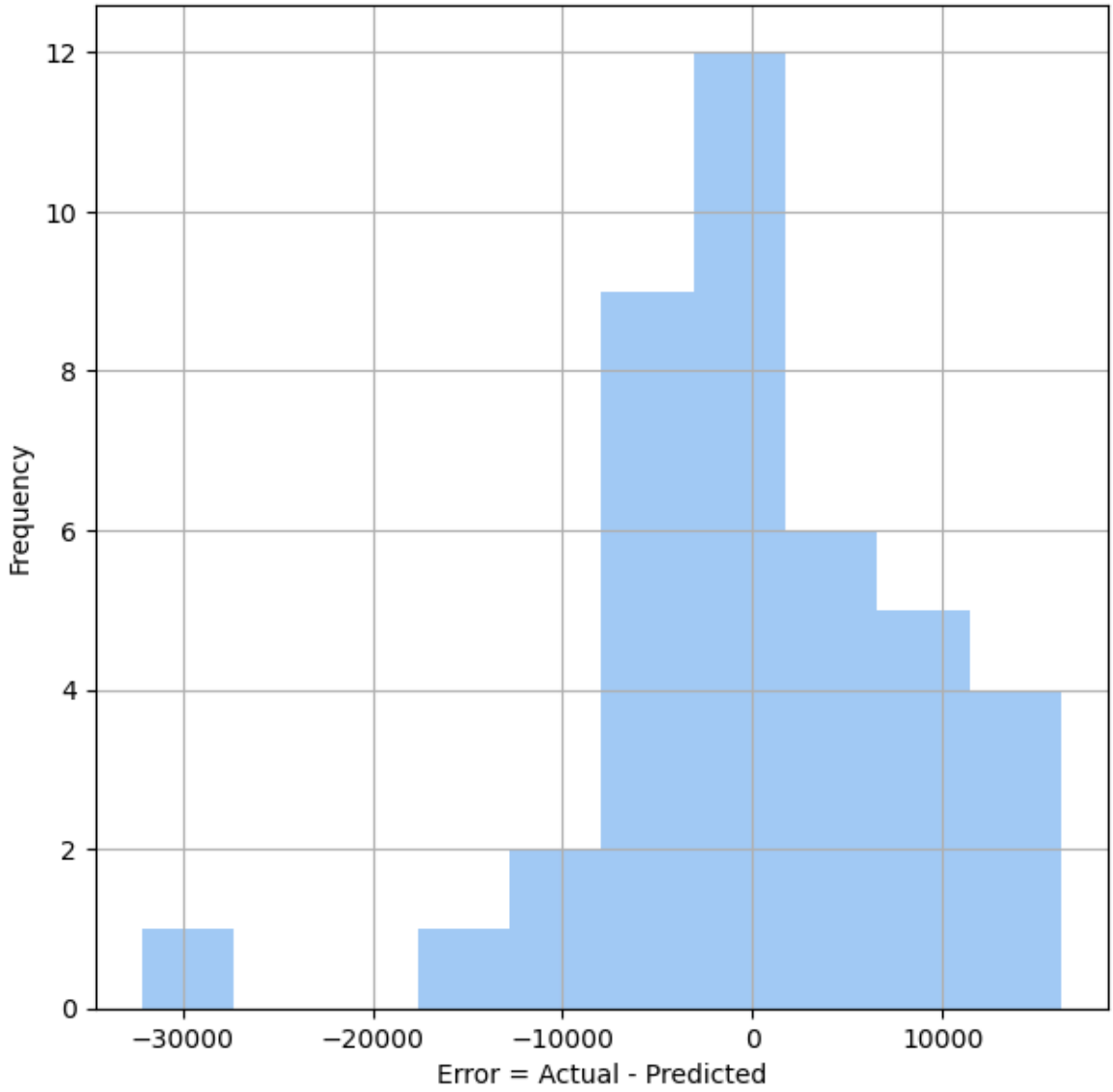
```
print ( "The mean of the error is : " , np.mean ( Model.Error ) )


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

```
The mean of the error is :  -1.7280399333685637e-11


The mean of absolute error is :  6344.162122366675
```

NORMAL DISTRIBUTION

In [27]:

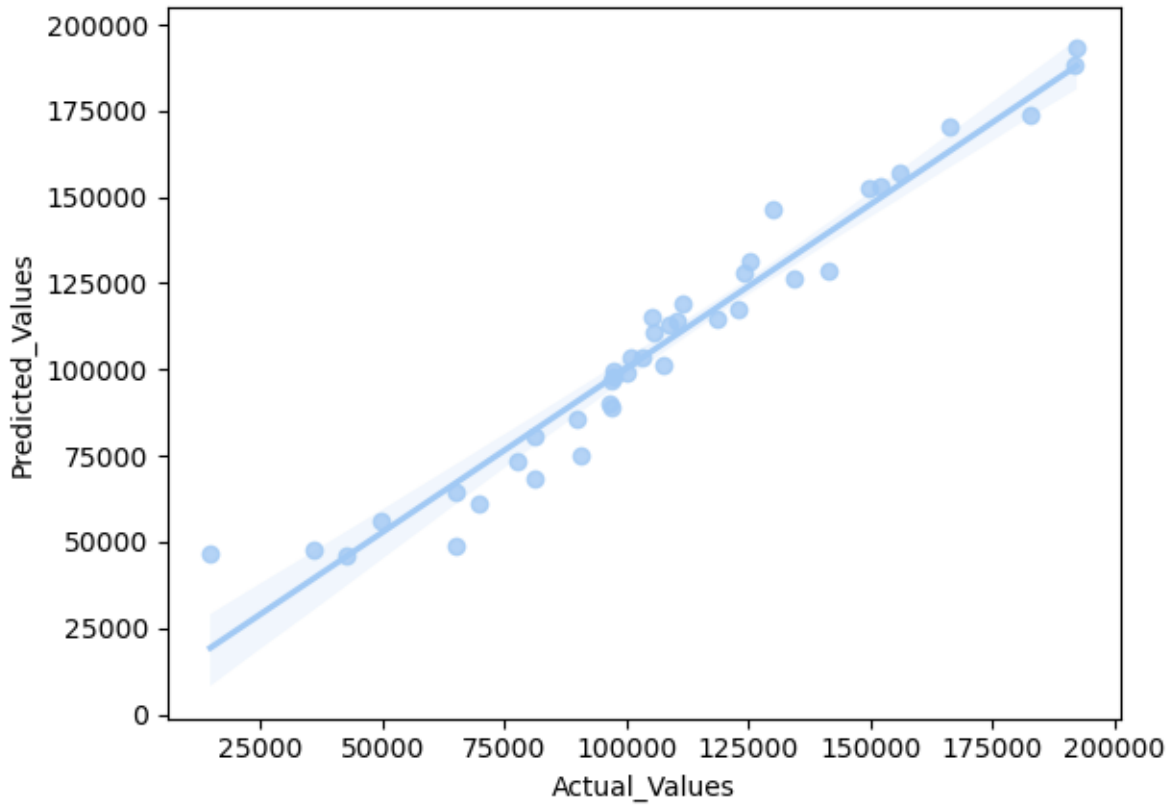
```
plt.figure ( figsize = ( 7 , 7 ) )


plt.xlabel ( 'Error = Actual - Predicted' )


plt.ylabel ( 'Frequency' )


plt.grid ( )


plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [28]:

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



METRICS

In [29]:

```
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 ) )
```

APPLYING LASSO

A high Lasso score indicates that the model has a large number of nonzero coefficients,

which can be interpreted as a sign of overfitting or high complexity.

In [30]:

```
from sklearn.linear_model import Lasso

L = Lasso ( )

L.fit ( Train_X , Train_Y )

print ( 'The lasso score is : ' , L.score ( Train_X , Train_Y ) * 100 )
Model = pd.DataFrame ( )

Model [ 'Predicted_Values' ] = L.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
```

MEAN

In [38]:

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

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

```
The mean of the error is :  -1.4006218407303095e-11
```

```
The mean of absolute error is :  6343.93405472469
```

NORMAL DISTRIBUTION

In [39]:

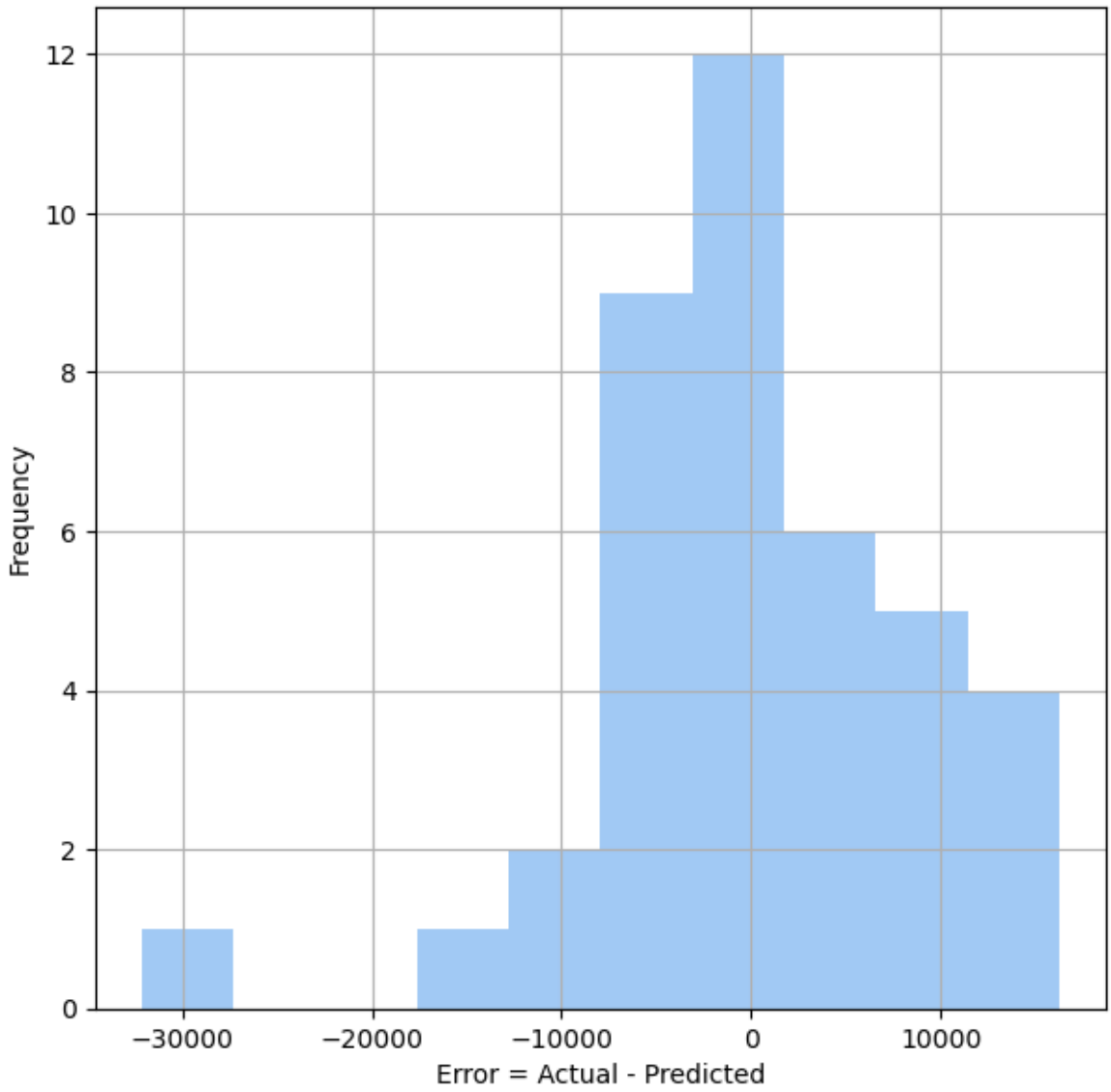
```
plt.figure ( figsize = ( 7 , 7 ) )
```

```
plt.xlabel ( 'Error = Actual - Predicted' )
```

```
plt.ylabel ( 'Frequency' )
```

```
plt.grid ( )
```

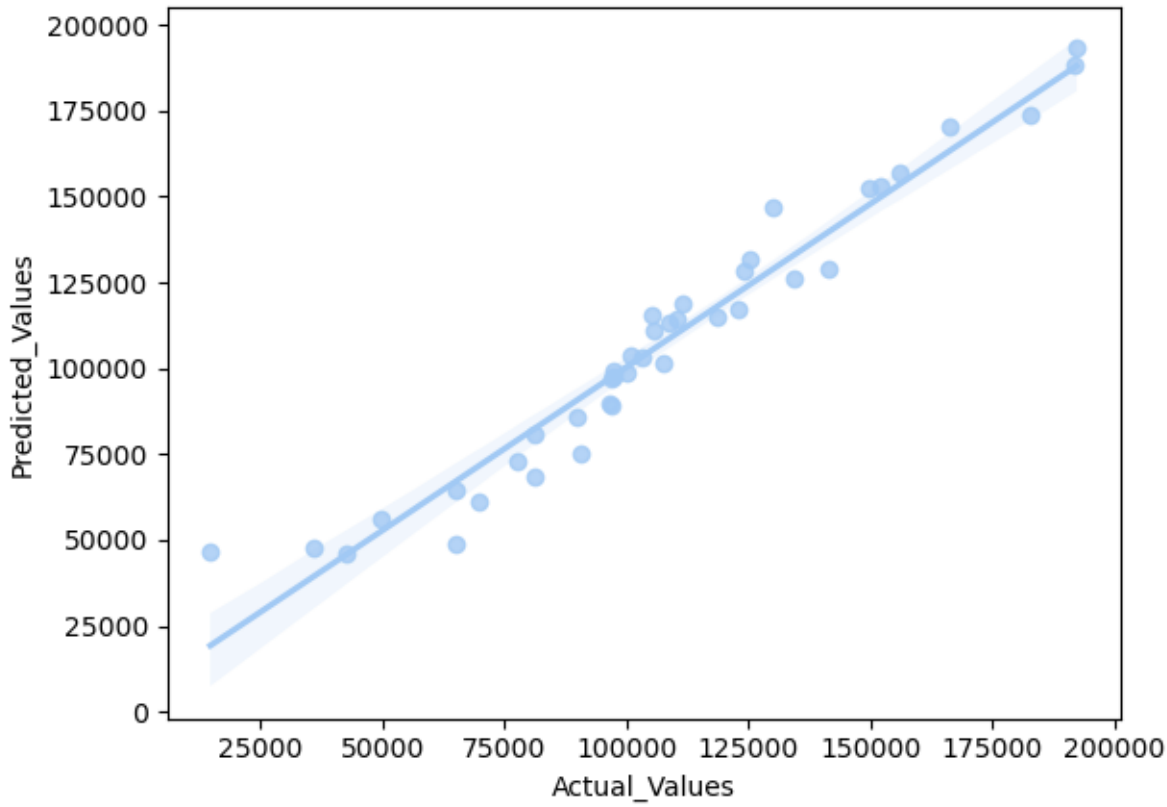
```
plt.hist ( x = Model.Error );
```



HETEROSCEDASTICITY

In [40]:

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



METRICS

In [41]:

```
Predicted_Test = L.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 ) )
```

```
The mean square value of error test is  91259520.16870753

The mean square value of error train is  77776789.34088856

The root mean square value of error test is  9552.984882679733

The root mean square value of error train is  8819.114997599734

The mean absolute percentage error (of test) is  6.141679767019481

The mean absolute percentage error (of train) is  11.677247076702795

Accuracy on train data is :  93.85832023298052

Accuracy on test data is :  88.3227529232972
```

Versions of Libraries:

- **NumPy: 1.24.3**
- **Pandas: 1.5.3**
- **Seaborn: 0.12.2**
- **Matplotlib: 3.7.1**
- **Scikit-Learn: 1.2.2**
- **Statsmodels: 0.14.0**

Conclusion:

The Profit Prediction for Startups project successfully demonstrates the use of machine learning to predict the profitability of startups based on their spending in areas like R&D, administration, and marketing, as well as their geographical location. By implementing various regression models, including linear regression, decision trees, random forests, and gradient boosting, we were able to identify the best-performing model based on evaluation metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R-squared (R^2) scores.

The project's structured methodology—covering data preprocessing, feature engineering, model selection, and interpretability—ensures that the final model is both accurate and actionable. The feature importance analysis provides valuable insights into how different spending areas contribute to profitability, offering stakeholders a clear picture of investment priorities. For instance, if R&D spending is shown to have a strong impact on profit, startup founders and investors can make more informed decisions about budget allocation.

With the model deployed as an API, it can now serve as a predictive tool accessible to entrepreneurs, investors, and analysts, supporting data-driven decision-making for new ventures. This project lays the foundation for future work, such as adding additional features, refining the model further, or integrating it into a broader decision-support platform for startup strategy. Overall, the project demonstrates the power of machine learning in the business context, where predictive analytics can drive smarter, more informed business strategies.