Implementation of Linear regression

29 September 2022 06:44

Dataset : "Details of the 50 start-ups"

The Following describes the dataset columns:

- **R&D Spend** - Amount of money that spend on Research and Development.
- **Administration -** Amount of money that spend on Administration.
- **Marketing Spend -** Amount of money that spend on Marketing
- **State -** The State where the start-ups operate.
- **Profit -** Amount of profit earned by the start-ups.

Objective

- To predict the profit made by a start-up on the basis of expenses incurred and the state where they operate.

STEP 01: "Importing Libraries"

- General libraries:
  Pandas, NumPy,
  Math from NumPy.

```
import numpy as np
import pandas as pd
from numpy import math
```

- **Scikit-learn** is a free software machine learning library for the Python programming language.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

- For performing evaluation metrics,

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
```

- For Data-visualization,

```
import matplotlib.pyplot as plt
```

STEP 02: "Importing the Drive and Data"

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
# Importing the dataset
dataset = pd.read_csv('/content/drive/MyDrive/Almabetter/Module 04 ML/50_Startups.csv')
```

STEP 03: "Understanding the Data"

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

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

```
dataset.shape
```

```
(50, 5)
```

```
[7] dataset.info()
```

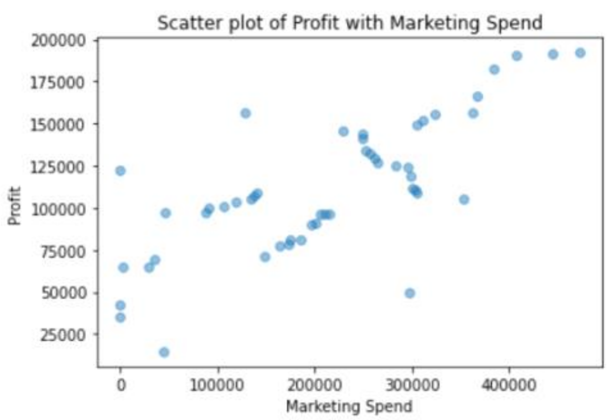
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   R&D Spend        50 non-null     float64
 1   Administration   50 non-null     float64
 2   Marketing Spend  50 non-null     float64
 3   State            50 non-null     object
 4   Profit           50 non-null     float64
dtypes: float64(4), object(1)
memory usage: 2.1+ KB
```

- We noticed that, all columns is in float datatype
- Except, column "State" which is categorical.
- As the column profit will play vital role in this dataset, we can use plotting option to compare the data's and visualize it.

STEP 04: "Scatter plot on Profit vs ______R & D spend/ Administration spend / Marketing spend"

Marketing Spend Vs Profit:

```
plt.scatter(dataset['Marketing Spend'], dataset['Profit'], alpha=0.5)
plt.title('Scatter plot of Profit with Marketing Spend')
plt.xlabel('Marketing Spend')
plt.ylabel('Profit')
plt.show()
```

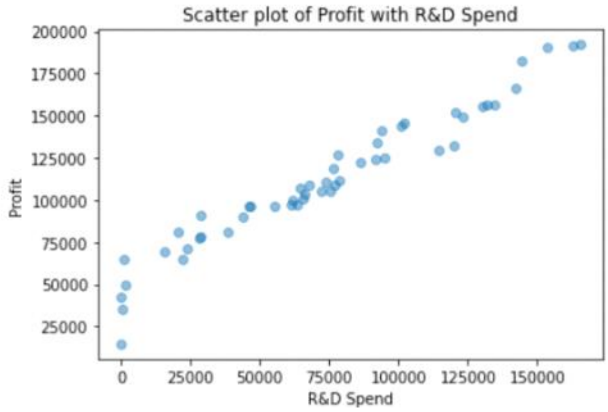


Note: What is alpha in matplotlib?

Matplotlib allows you to regulate the transparency of a graph plot using the alpha attribute. By default, alpha=1. If you would like to form the graph plot more transparent, then you'll make alpha but 1, such as 0.5 or 0.25.

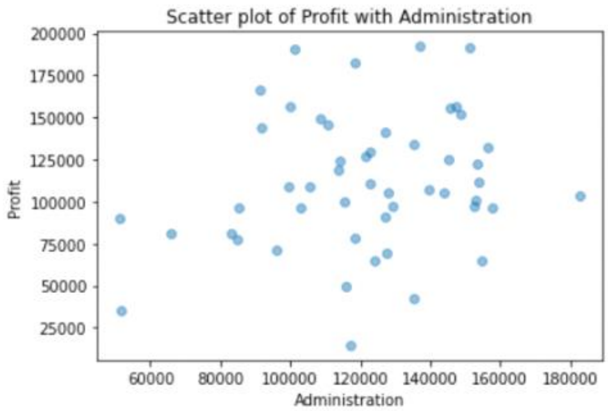
R & D spend vs Profit:

```
plt.scatter(dataset['R&D Spend'], dataset['Profit'], alpha=0.5)
plt.title('Scatter plot of Profit with R&D Spend')
plt.xlabel('R&D Spend')
plt.ylabel('Profit')
plt.show()
```



Administration vs Profit:

```
plt.scatter(dataset['Administration'], dataset['Profit'], alpha=0.5)
plt.title('Scatter plot of Profit with Administration')
plt.xlabel('Administration')
plt.ylabel('Profit')
plt.show()
```



Observation from the above scatter plot:

- Both Marketing spend and R&D with profit has some linear relationship[.
- While the administration and profit has been scattered throughout the data.

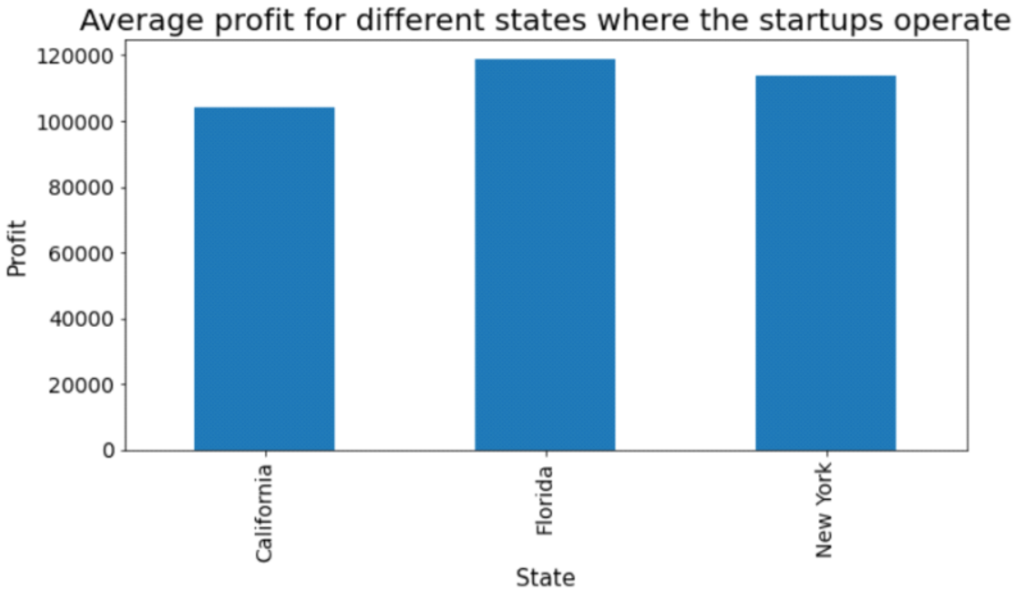
STEP 05: "Visualize the Average profit vs each state" - Using Bar Plot.

```
avg_profit_each_state = dataset.groupby(['State'])['Profit'].mean().plot.bar(
    figsize = (10,5),
    fontsize = 14
)

# Set the title
avg_profit_each_state.set_title("Average profit for different states where the startups operate", fontsize = 20)

# Set x and y-labels
avg_profit_each_state.set_xlabel("State", fontsize = 15)
avg_profit_each_state.set_ylabel("Profit", fontsize = 15)
```

- Using Group by option, "states" column has been grouped with respect to mean of the profit.



STEP 06: "Create dummy variables for the categorical variable State"

- From the dataset, we noticed the categorical column - "State"

	R&D Spend	Administration	Marketing Spend	State	Profit
0	165349.20	136897.80	471784.10	New York	192261.83
1	162597.70	151377.59	443898.53	California	191792.06
2	153441.51	101145.55	407934.54	Florida	191050.39
3	144372.41	118671.85	383199.62	New York	182901.99
4	142107.34	91391.77	366168.42	Florida	166187.94

Categorical Column

- To get idea about the counts of each category using (**dataset.State.value_counts()**)

```
New York      17
California    17
Florida       16
Name: State, dtype: int64
```

- Hence, we transforming the categorical column into the dummy variables as below,

```
dataset['NewYork_State'] = np.where(dataset['State']=='New York', 1, 0)
dataset['California_State'] = np.where(dataset['State']=='California', 1, 0)
dataset['Florida_State'] = np.where(dataset['State']=='Florida', 1, 0)
```

- After creating the new dummy variables with each column, we can drop the existing categorical column.

```
dataset.drop(columns=['State'],axis=1,inplace=True)
```

Restructured dataframe as,

	R&D Spend	Administration	Marketing Spend	Profit	NewYork_State	California_State	Florida_State
0	165349.20	136897.80	471784.10	192261.83	1	0	0
1	162597.70	151377.59	443898.53	191792.06	0	1	0
2	153441.51	101145.55	407934.54	191050.39	0	0	1
3	144372.41	118671.85	383199.62	182901.99	1	0	0
4	142107.34	91391.77	366168.42	166187.94	0	0	1

STEP 07: "Separating the Independent variable and dependent variable"

For Dependent variables,

- We already know, column "profit" is the only dependent variable
- Hence,

```
dependent_variable = 'Profit'
```

For Independent Variables,

- Just creating a list of variables from the dataset which should be excluded from "dependent variable"

```
independent_variables = list(set(dataset.columns.tolist()) - {dependent_variable})
```

```
independent_variables
```

```
['California_State',
 'Marketing Spend',
 'NewYork_State',
 'R&D Spend',
 'Administration',
 'Florida_State']
```

- And finally creating a new variable, **"X"** - independent variable and "**Y" -**Dependent variable

```
# Create the data of independent variables
X = dataset[independent_variables].values

# Create the dependent variable data
y = dataset[dependent_variable].values
```

STEP 08: "Start training the dataset"- using "Train_test_split"

- It has been done by using the method.
- What is train test split?

  The train-test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms/Applications.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

- Test size ?
  - test_size: this is a float value whose value ranges between 0.0 and 1.0. it represents the proportion of our test size. its default value is none.
  - Here, we taking 20% of the dataset as the test dataset.
- Random state?
  - The random state hyperparameter in the train_test_split() function controls the **shuffling process**.
  - With **random_state=None** , we get different train and test sets across different executions and the shuffling process is out of control. With **random_state=0** , we get the same train and test sets across different executions.

STEP 09: "Transform the Data into required scaling"

- On general understanding of the dataset we splited, we execute some code to view the data splited.
- We noticed the values are ranging widely hence it better to scale the values in the dataset by using the **Transform features**.

```
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

STEP 10: "Fitting the Linear Regression to our training dataset"

- **LinearRegression()**
  Ordinary least squares Linear Regression.

  LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

- The intercept (sometimes called the "constant") in a regression model represents the mean value of the response variable when all of the predictor variables in the model are equal to zero

```
regressor.intercept_
```

```
44153.954667848506
```

- To get the co-efficient of the features of the dataset,

```
regressor.coef_
```

```
array([ 8.66383692e+01,  1.72720281e+04,  7.86007422e+02,  1.27892182e+05,
        3.49927567e+03, -8.72645791e+02])
```

STEP 11: "Predicting the "Y" for "X_test"

- Using the values of the X-test that we separated initially using train test split method, we predicting the values of the "y" and declaring it as a new variable name called "**y_pred**"

```
y_pred = regressor.predict(X_test)
```

STEP 12: "Performing the Evaluation Metrics"

- We already know the actual values of "Y-test" for "X-test" from the separated data.
- Now, we predicted the value of y from the "Y-pred" for "X-pred".
- Evaluation metrics will be performed for checking the accuracy.

```
mean_squared_error(y_test, y_pred)
```

```
83502864.0325773
```

```python
math.sqrt(mean_squared_error(y_test, y_pred))
```

```
9137.99015279494
```

```python
r2_score(y_test, y_pred)
```

```
0.9347068473282425
```

```python
from sklearn.metrics import mean_absolute_error
mae = mean_absolute_error(y_pred,y_test)
mae
```

```
7514.293659640595
```

So, the mean absolute error is 7514.293659640595. Therefore our predicted value can be 7514.293659640595 units more or less than the actual value.