Multi-Linear-Regression

January 14, 2025

[ ]:
```
'''
    Multi Linear Regression -->

    Multiple linear regression is a statistical technique used to model
    the relationship between one dependent variable and two or more independent
    variables. It generalizes simple linear regression by allowing for multiple
    factors that could predict the outcome.

    Equation -->
    Y =  0 +  1X1 + 2X2 +   +  nXn +

    Where,
    Y = Predicted value (dependent variable)
    X1, X2,.. Xn = features (independent values)
     0 = Intercept
     1X1 + 2X2 +   +  nXn = Coefficients
       = Error term
'''
```

[ ]:
```
'''
    P Value -->

    The p-value is a statistical measure used to determine the significance
    of a result in hypothesis testing. In the context of regression models,
    it helps you understand whether the relationship between the dependent
    and independent variables is statistically significant.
'''
```

[ ]:
```
'''
    Methods for building model -->

    All in
    Backward Elimination
    Forward Selection
    Bi-Directional Elimination
    Score Comparision
'''
```

[ ]:
```
'''
    All In -->

    Definition: This method includes all the available features in the model
    without performing any feature selection. The idea is to fit the
    regression model with all independent variables (predictors).

    Use Case: Useful when you are confident that all features are relevant
    or when there is no clear idea of which features should be excluded.

    Pros: Simple and straightforward.

    Cons: May include irrelevant or redundant features, which can reduce
    model performance or cause overfitting.
'''
```

[ ]:
```
'''
    Backward Elimination -->

    Definition: Backward Elimination starts by fitting the model with all
    features and iteratively removes the least significant feature
    (based on p-values) one at a time. This process continues until all
    remaining features are statistically significant.

    Steps:
    Fit the model with all features.
    Remove the feature with the highest p-value (above a significance
    threshold, usually 0.05).
    Refit the model and repeat until all p-values are below the threshold.

    Pros: Helps in reducing the complexity of the model by removing
    non-significant features.

    Cons: May remove some features that, when combined with others,
    could improve the model.
'''
```

[ ]:
```
'''
    Forward Selection -->

    Definition: Forward Selection begins with an empty model (no features)
    and adds the most statistically significant feature at each step.
    The process continues until adding more features does not significantly
    improve the model.

    Steps:
    Start with no features.
```

```
    Add the feature with the lowest p-value (or highest R-squared increase).
    Repeat until adding further features does not significantly improve the␣
 ↪model.

    Pros: Builds the model step by step, allowing you to add only the most
    relevant features.

    Cons: Can miss combinations of features that, when included together,
    may be significant.
'''
```

[ ]:
```
'''
    Bi-Directional Elimination (Stepwise Regression) -->

    Definition: Bi-Directional Elimination combines the processes of
    Backward Elimination and Forward Selection. At each step, features
    are added or removed based on their significance, allowing for both
    directions of feature selection.

    Steps:
    Start with no features or a few selected ones.
    Perform Forward Selection by adding significant features.
    After adding a feature, perform Backward Elimination to see if any
    of the added features become insignificant and need removal.
    Continue this process until no further improvements can be made by
    adding or removing features.

    Pros: Flexible, as it allows for adding and removing features dynamically.

    Cons: Computationally intensive, especially for large datasets.
'''
```

[ ]:
```
'''
    Score Comparison (e.g., AIC/BIC) -->

    Definition: This method selects features based on a specific criterion,
    such as Akaike Information Criterion (AIC) or Bayesian Information Criterion
    (BIC). The model with the lowest AIC/BIC score is considered the best.
    It balances model complexity (the number of parameters) and the␣
 ↪goodness-of-fit.

    AIC/BIC Formula: These scores penalize models with more features,
    encouraging a balance between simplicity and accuracy:

    AIC = 2k - 2ln(L)
    BIC = ln(n)k - 2ln(L)
```

```
    Where,

    k = number of parameters
    L = likelihood of the model
    n = number of data points

    Pros: Provides a trade-off between model complexity and accuracy,
    avoiding overfitting.

    Cons: May not be intuitive to interpret compared to p-values and R-squared.
'''
```

[9]:
```
#   Importing Libraries -->

import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

[10]:
```
#   Importing Dataset -->

data = pd.read_csv('Data/50_Startups.csv')
data
```

[10]:

	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
5	131876.90	99814.71	362861.36	New York	156991.12
6	134615.46	147198.87	127716.82	California	156122.51
7	130298.13	145530.06	323876.68	Florida	155752.60
8	120542.52	148718.95	311613.29	New York	152211.77
9	123334.88	108679.17	304981.62	California	149759.96
10	101913.08	110594.11	229160.95	Florida	146121.95
11	100671.96	91790.61	249744.55	California	144259.40
12	93863.75	127320.38	249839.44	Florida	141585.52
13	91992.39	135495.07	252664.93	California	134307.35
14	119943.24	156547.42	256512.92	Florida	132602.65
15	114523.61	122616.84	261776.23	New York	129917.04
16	78013.11	121597.55	264346.06	California	126992.93
17	94657.16	145077.58	282574.31	New York	125370.37

```
18   91749.16   114175.79   294919.57     Florida  124266.90
19   86419.70   153514.11        0.00    New York  122776.86
20   76253.86   113867.30   298664.47  California  118474.03
21   78389.47   153773.43   299737.29    New York  111313.02
22   73994.56   122782.75   303319.26     Florida  110352.25
23   67532.53   105751.03   304768.73     Florida  108733.99
24   77044.01    99281.34   140574.81    New York  108552.04
25   64664.71   139553.16   137962.62  California  107404.34
26   75328.87   144135.98   134050.07     Florida  105733.54
27   72107.60   127864.55   353183.81    New York  105008.31
28   66051.52   182645.56   118148.20     Florida  103282.38
29   65605.48   153032.06   107138.38    New York  101004.64
30   61994.48   115641.28    91131.24     Florida   99937.59
31   61136.38   152701.92    88218.23    New York   97483.56
32   63408.86   129219.61    46085.25  California   97427.84
33   55493.95   103057.49   214634.81     Florida   96778.92
34   46426.07   157693.92   210797.67  California   96712.80
35   46014.02    85047.44   205517.64    New York   96479.51
36   28663.76   127056.21   201126.82     Florida   90708.19
37   44069.95    51283.14   197029.42  California   89949.14
38   20229.59    65947.93   185265.10    New York   81229.06
39   38558.51    82982.09   174999.30  California   81005.76
40   28754.33   118546.05   172795.67  California   78239.91
41   27892.92    84710.77   164470.71     Florida   77798.83
42   23640.93    96189.63   148001.11  California   71498.49
43   15505.73   127382.30    35534.17    New York   69758.98
44   22177.74   154806.14    28334.72  California   65200.33
45    1000.23   124153.04     1903.93    New York   64926.08
46    1315.46   115816.21   297114.46     Florida   49490.75
47       0.00   135426.92        0.00  California   42559.73
48     542.05    51743.15        0.00    New York   35673.41
49       0.00   116983.80    45173.06  California   14681.40
```

[11]:
```python
#   Seperating Target From Features -->

X_data = data.iloc[:, :-1].values
y_data = data.iloc[:, -1].values
```

[12]:
```python
X_data
```

[12]:
```
array([[165349.2, 136897.8, 471784.1, 'New York'],
       [162597.7, 151377.59, 443898.53, 'California'],
       [153441.51, 101145.55, 407934.54, 'Florida'],
       [144372.41, 118671.85, 383199.62, 'New York'],
       [142107.34, 91391.77, 366168.42, 'Florida'],
       [131876.9, 99814.71, 362861.36, 'New York'],
       [134615.46, 147198.87, 127716.82, 'California'],
```

```
[130298.13, 145530.06, 323876.68, 'Florida'],
[120542.52, 148718.95, 311613.29, 'New York'],
[123334.88, 108679.17, 304981.62, 'California'],
[101913.08, 110594.11, 229160.95, 'Florida'],
[100671.96, 91790.61, 249744.55, 'California'],
[93863.75, 127320.38, 249839.44, 'Florida'],
[91992.39, 135495.07, 252664.93, 'California'],
[119943.24, 156547.42, 256512.92, 'Florida'],
[114523.61, 122616.84, 261776.23, 'New York'],
[78013.11, 121597.55, 264346.06, 'California'],
[94657.16, 145077.58, 282574.31, 'New York'],
[91749.16, 114175.79, 294919.57, 'Florida'],
[86419.7, 153514.11, 0.0, 'New York'],
[76253.86, 113867.3, 298664.47, 'California'],
[78389.47, 153773.43, 299737.29, 'New York'],
[73994.56, 122782.75, 303319.26, 'Florida'],
[67532.53, 105751.03, 304768.73, 'Florida'],
[77044.01, 99281.34, 140574.81, 'New York'],
[64664.71, 139553.16, 137962.62, 'California'],
[75328.87, 144135.98, 134050.07, 'Florida'],
[72107.6, 127864.55, 353183.81, 'New York'],
[66051.52, 182645.56, 118148.2, 'Florida'],
[65605.48, 153032.06, 107138.38, 'New York'],
[61994.48, 115641.28, 91131.24, 'Florida'],
[61136.38, 152701.92, 88218.23, 'New York'],
[63408.86, 129219.61, 46085.25, 'California'],
[55493.95, 103057.49, 214634.81, 'Florida'],
[46426.07, 157693.92, 210797.67, 'California'],
[46014.02, 85047.44, 205517.64, 'New York'],
[28663.76, 127056.21, 201126.82, 'Florida'],
[44069.95, 51283.14, 197029.42, 'California'],
[20229.59, 65947.93, 185265.1, 'New York'],
[38558.51, 82982.09, 174999.3, 'California'],
[28754.33, 118546.05, 172795.67, 'California'],
[27892.92, 84710.77, 164470.71, 'Florida'],
[23640.93, 96189.63, 148001.11, 'California'],
[15505.73, 127382.3, 35534.17, 'New York'],
[22177.74, 154806.14, 28334.72, 'California'],
[1000.23, 124153.04, 1903.93, 'New York'],
[1315.46, 115816.21, 297114.46, 'Florida'],
[0.0, 135426.92, 0.0, 'California'],
[542.05, 51743.15, 0.0, 'New York'],
[0.0, 116983.8, 45173.06, 'California']], dtype=object)
```

[13]:
```
#   Encoding Categorical Data -->
```

```
clt = ColumnTransformer(transformers=[('encoder', OneHotEncoder(), [3])],␣
 ↪remainder='passthrough')
X_data = np.array(clt.fit_transform(X_data))
```

[14]:
```
X_data
```

[14]:
```
array([[0.0, 0.0, 1.0, 165349.2, 136897.8, 471784.1],
       [1.0, 0.0, 0.0, 162597.7, 151377.59, 443898.53],
       [0.0, 1.0, 0.0, 153441.51, 101145.55, 407934.54],
       [0.0, 0.0, 1.0, 144372.41, 118671.85, 383199.62],
       [0.0, 1.0, 0.0, 142107.34, 91391.77, 366168.42],
       [0.0, 0.0, 1.0, 131876.9, 99814.71, 362861.36],
       [1.0, 0.0, 0.0, 134615.46, 147198.87, 127716.82],
       [0.0, 1.0, 0.0, 130298.13, 145530.06, 323876.68],
       [0.0, 0.0, 1.0, 120542.52, 148718.95, 311613.29],
       [1.0, 0.0, 0.0, 123334.88, 108679.17, 304981.62],
       [0.0, 1.0, 0.0, 101913.08, 110594.11, 229160.95],
       [1.0, 0.0, 0.0, 100671.96, 91790.61, 249744.55],
       [0.0, 1.0, 0.0, 93863.75, 127320.38, 249839.44],
       [1.0, 0.0, 0.0, 91992.39, 135495.07, 252664.93],
       [0.0, 1.0, 0.0, 119943.24, 156547.42, 256512.92],
       [0.0, 0.0, 1.0, 114523.61, 122616.84, 261776.23],
       [1.0, 0.0, 0.0, 78013.11, 121597.55, 264346.06],
       [0.0, 0.0, 1.0, 94657.16, 145077.58, 282574.31],
       [0.0, 1.0, 0.0, 91749.16, 114175.79, 294919.57],
       [0.0, 0.0, 1.0, 86419.7, 153514.11, 0.0],
       [1.0, 0.0, 0.0, 76253.86, 113867.3, 298664.47],
       [0.0, 0.0, 1.0, 78389.47, 153773.43, 299737.29],
       [0.0, 1.0, 0.0, 73994.56, 122782.75, 303319.26],
       [0.0, 1.0, 0.0, 67532.53, 105751.03, 304768.73],
       [0.0, 0.0, 1.0, 77044.01, 99281.34, 140574.81],
       [1.0, 0.0, 0.0, 64664.71, 139553.16, 137962.62],
       [0.0, 1.0, 0.0, 75328.87, 144135.98, 134050.07],
       [0.0, 0.0, 1.0, 72107.6, 127864.55, 353183.81],
       [0.0, 1.0, 0.0, 66051.52, 182645.56, 118148.2],
       [0.0, 0.0, 1.0, 65605.48, 153032.06, 107138.38],
       [0.0, 1.0, 0.0, 61994.48, 115641.28, 91131.24],
       [0.0, 0.0, 1.0, 61136.38, 152701.92, 88218.23],
       [1.0, 0.0, 0.0, 63408.86, 129219.61, 46085.25],
       [0.0, 1.0, 0.0, 55493.95, 103057.49, 214634.81],
       [1.0, 0.0, 0.0, 46426.07, 157693.92, 210797.67],
       [0.0, 0.0, 1.0, 46014.02, 85047.44, 205517.64],
       [0.0, 1.0, 0.0, 28663.76, 127056.21, 201126.82],
       [1.0, 0.0, 0.0, 44069.95, 51283.14, 197029.42],
       [0.0, 0.0, 1.0, 20229.59, 65947.93, 185265.1],
       [1.0, 0.0, 0.0, 38558.51, 82982.09, 174999.3],
       [1.0, 0.0, 0.0, 28754.33, 118546.05, 172795.67],
```

```
[0.0, 1.0, 0.0, 27892.92, 84710.77, 164470.71],
[1.0, 0.0, 0.0, 23640.93, 96189.63, 148001.11],
[0.0, 0.0, 1.0, 15505.73, 127382.3, 35534.17],
[1.0, 0.0, 0.0, 22177.74, 154806.14, 28334.72],
[0.0, 0.0, 1.0, 1000.23, 124153.04, 1903.93],
[0.0, 1.0, 0.0, 1315.46, 115816.21, 297114.46],
[1.0, 0.0, 0.0, 0.0, 135426.92, 0.0],
[0.0, 0.0, 1.0, 542.05, 51743.15, 0.0],
[1.0, 0.0, 0.0, 0.0, 116983.8, 45173.06]], dtype=object)
```

[15]:
```
#   Splitting The Data -->

X_train, X_test, y_train, y_test = train_test_split(X_data, y_data, test_size=0.
↪2, random_state=42)
```

[16]:
```
#   Building Model -->

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

[16]: LinearRegression()

[17]:
```
#   Predicting Values -->

y_pred = model.predict(X_test)
np.set_printoptions(precision=2) #  To display numeric values upto 2 decimal␣
↪points
print(np.concatenate((y_pred.reshape(len(y_pred),1),y_test.
↪reshape(len(y_test),1)),1))
```

```
[[126362.88 134307.35]
 [ 84608.45  81005.76]
 [ 99677.49  99937.59]
 [ 46357.46  64926.08]
 [128750.48 125370.37]
 [ 50912.42  35673.41]
 [109741.35 105733.54]
 [100643.24 107404.34]
 [ 97599.28  97427.84]
 [113097.43 122776.86]]
```

[18]:
```
# Plotting the actual vs predicted values

plt.figure(figsize=(8,6))
sns.scatterplot(x=y_test, y=y_pred)  # Scatter plot of actual vs predicted
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',␣
↪lw=2)  # Identity line (perfect prediction)
plt.title("Actual vs Predicted Profit")
```

```
plt.xlabel("Actual Profit")
plt.ylabel("Predicted Profit")
plt.grid(True)
plt.show()
```