Multi_Linear_Regression_Activity

March 26, 2024

1 Multi-Linear Regression Analysis

This notebook provides a comprehensive analysis of multiple linear regression on a dataset containing hours , prep exams and scores by ~ Kiaan Maharaj (ST10116983)

1.1 Import libraries

[2]:
```
import pandas as pd
```

1.2 Load and inspect the data

[5]:
```
data = pd.read_csv('../Multi_Linear_Regression/Dataset.csv',delimiter=';')

data.head()
```

[5]:

	hours	prep exams	score
0	1	1	76
1	2	3	78
2	2	3	85
3	4	5	88
4	2	2	72

Describe the data

[4]:
```
data.describe()
```

[4]:

	hours	prep exams	score
count	20.000000	20.000000	20.000000
mean	3.150000	2.450000	83.700000
std	1.598519	1.571958	9.841373
min	1.000000	0.000000	62.000000
25%	2.000000	1.000000	76.000000
50%	3.000000	2.500000	85.000000
75%	4.000000	4.000000	90.500000
max	6.000000	5.000000	99.000000

1.3 Model Coefficients and Y-intercept Calculation

[17]:
```
import statsmodels.api as sm

# Define independent variables (X) and dependent variable (y)
X = data[['hours', 'prep exams']]
y = data['score']

# Add a constant term to the independent variables to fit the intercept
X = sm.add_constant(X)

# Fit the multiple linear regression model
model = sm.OLS(y, X).fit()

# Get the coefficients (including the intercept)
coefficients = model.params

# Extract the y-intercept (beta_0)
intercept = coefficients['const']

print("Y-intercept (beta_0):", intercept)
print("Coefficients:")
print(coefficients)

# Define the formula using the coefficients and variables
formula = f"score = {intercept:.2f} + {coefficients['hours']:.2f} * hours +␣
↪{coefficients['prep exams']:.2f} * prep_exams"


print("\nMultiple Linear Regression Formula:")
print(formula)
```

```
Y-intercept (beta_0): 67.67352554133268
Coefficients:
const         67.673526
hours          5.555748
prep exams    -0.601687
dtype: float64

Multiple Linear Regression Formula:
score = 67.67 + 5.56 * hours + -0.60 * prep_exams
```

1.4 Calculate and Interpret the Correlation Coefficient

calculate and interpret the correlation coefficient between hours, number of exams prepared, and score.

[8]:
```
# Calculate the correlation matrix
correlation_matrix = data.corr()
```

```
# Extract the correlation coefficient between 'hours', 'exams', and 'score'
correlation_coefficients = correlation_matrix.loc[['hours', 'prep exams'],␣
↪'score']

# Print the correlation coefficients without displaying additional information
print(correlation_coefficients)
```

```
hours         0.852791
prep exams    0.369810
Name: score, dtype: float64
```

1.5 Estimated Regression Line Parameters

Write down estimated regression line parameters by performing linear regression analysis.

[9]:
```
import statsmodels.api as sm

# Define the independent variables (X) and the dependent variable (y)
X = data[['hours', 'prep exams']]
y = data['score']

# Add a constant term to the independent variables
X = sm.add_constant(X)

# Fit the linear regression model
model = sm.OLS(y, X).fit()

# Get the estimated parameters (intercept and slopes)
intercept = model.params['const']
slope_hours = model.params['hours']
slope_exams = model.params['prep exams']

print("Intercept:", intercept)
print("Slope for hours:", slope_hours)
print("Slope for exams:", slope_exams)
```

```
Intercept: 67.67352554133268
Slope for hours: 5.555748295250623
Slope for exams: -0.601686804641715
```

1.6 Estimate the Scores Value

Estimate the score value for an observation with 6 hours of preparation and 4 exams taken.

[10]:
```
# Given values
hours = 6
exams = 4
```

```
# Calculate the estimated score
estimated_score = intercept + (slope_hours * hours) + (slope_exams * exams)

print("Estimated score:", estimated_score)
```

```
Estimated score: 98.60126809426956
```

1.7 Calculate and Interpret the Coefficient of Determination

calculate and interpret the coefficient of determination of the model.

[11]:
```
# Get the coefficient of determination (R-squared)
r_squared = model.rsquared

print("Coefficient of determination (R-squared):", r_squared)
```

```
Coefficient of determination (R-squared): 0.7340272170388175
```

1.8 Test for Significance

Test the model for significance on a 5% level using an F-test.

[24]:
```
# Get the p-value for the F-test
f_pvalue = model.f_pvalue

print("p-value for F-test:", f_pvalue)

# Compare the p-value with the significance level (0.05)
if f_pvalue < 0.05:
    print(f_pvalue , "< 0.05")
else:
    print(f_pvalue , "> 0.05")
```

```
p-value for F-test: 1.2915647352305291e-05
1.2915647352305291e-05 < 0.05
```

1.9 Train Data

[18]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Assuming 'X' contains independent variables and 'y' contains the dependent␣
↪variable
X = data[['hours', 'prep exams']]
y = data['score']

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

```python
# Initialize the linear regression model
model = LinearRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Predict the scores for the testing data
y_pred = model.predict(X_test)

# Evaluate the model performance
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error:", mse)
```

```
Mean Squared Error: 17.699192001292992
```