EXERCISE	3. SIMPLE LINEAR, MULTIPLE LINEAR AND LOGISTIC REGRESSION
DATE	30.07.2024

AIM:
To perform linear, multiple linear and logistic regression for the given dataset.

DESCRIPTION:
Regression Analysis:
Regression is a statistical method for modeling relationships between dependent (target) and independent (predictor) variables to predict continuous values like salary or price. The goal is to minimize prediction error, with smaller errors indicating better model accuracy.

Key Concepts:
Dependent Variable: The outcome you want to predict.
Independent Variables: Predictors used to forecast the outcome.

Types of Regression:
1. Linear Regression:
Models the relationship using a straight line.
Goal: Minimize the squared differences between data points and the regression line.

2. Multiple Linear Regression (MLR):
Extends linear regression to multiple predictors.
Assumptions: Linear relationship, low multicollinearity, independent observations, normally distributed residuals.

3. Logistic Regression:
Used for classification with categorical outcomes.
Objective: Predict probabilities for categories, such as pass/fail.

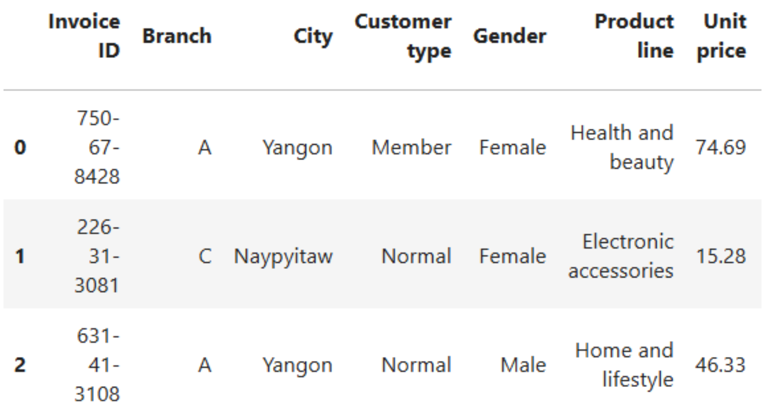
Performance Metrics:
Root Mean Squared Error (RMSE): Measures prediction error magnitude.
R-squared (r^2): Proportion of variance explained by the model.
Mean Absolute Error (MAE): Average absolute prediction error.
Mean Squared Error (MSE): Average squared differences between predictions and actual values.

QUESTIONS FOR LINEAR REGRESSION:

1. **Read CSV data into pandas dataframe object**

```
df = pd.read_csv('supermarket.csv')
df.head(3)
```

Output:



	Invoice ID	Branch	City	Customer type	Gender	Product line	Unit price
0	750-67-8428	A	Yangon	Member	Female	Health and beauty	74.69
1	226-31-3081	C	Naypyitaw	Normal	Female	Electronic accessories	15.28
2	631-41-3108	A	Yangon	Normal	Male	Home and lifestyle	46.33

2. **Do necessary preprocessing.**

```
df = df.dropna()    # Simple example: drop rows with missing values
```

3. **Choose independent variable (X) and dependent variable (Y) from given dataset.**

```
X = df[['Unit price']]
Y = df['Quantity']
```

4. **Find the bo and b1 values to get Ypredicted.**

```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
model = LinearRegression()
model.fit(X_train, Y_train)
Y_pred = model.predict(X_test)
```

Output:

```
▾ LinearRegression
LinearRegression()
```

5. **After getting Ypred, calculate the SSE (sum of squared error):**

```
SSE = np.sum((Y_test - Y_pred) ** 2)
```

6. **Calculate the RMSE (Root Mean Square Error) value.**

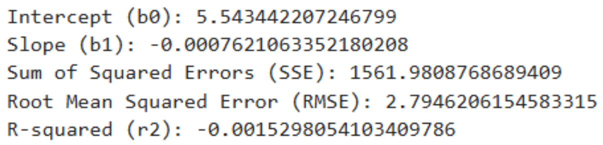
```
RMSE = np.sqrt(mean_squared_error(Y_test, Y_pred))
```

7. **Calculate the coefficient of determination (r2) r-square.**

```
r2 = r2_score(Y_test, Y_pred)
```

```
# Print the coefficients
print(f'Intercept (b0): {model.intercept_}')
print(f'Slope (b1): {model.coef_[0]}')
print(f'Sum of Squared Errors (SSE): {SSE}')
print(f'Root Mean Squared Error (RMSE): {RMSE}')
print(f'R-squared (r2): {r2}')
```

Output:

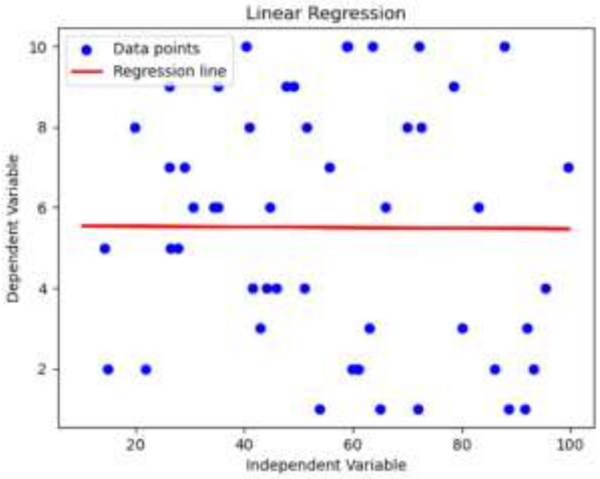


```
Intercept (b0): 5.543442207246799
Slope (b1): -0.0007621063352180208
Sum of Squared Errors (SSE): 1561.9808768689409
Root Mean Squared Error (RMSE): 2.7946206154583315
R-squared (r2): -0.0015298054103409786
```

8. **Plot regression line along with the given data points.**

```
subset_size = 50  # Number of data points to sample
subset_df = df.sample(n=subset_size, random_state=42)
# Prepare the subset data
X_subset = subset_df[['Unit price']]
Y_subset = subset_df['Quantity']
# Plot the regression line with the subset data
plt.scatter(X_subset, Y_subset, color='blue', label='Data points')
plt.plot(X_test, Y_pred, color='red', linewidth=2, label='Regression line')
plt.xlabel('Independent Variable')
plt.ylabel('Dependent Variable')
plt.title('Linear Regression')
plt.legend()
plt.show()
```

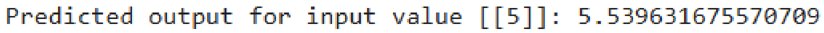
Output:



9. **Predict the output for a given input value.**

```
input_value = np.array([[5]])  # Replace with your test input value
predicted_output = model.predict(input_value)
print(f'Predicted output for input value {input_value}: {predicted_output[0]}')
```

Output:



```
Predicted output for input value [[5]]: 5.539631675570709
```

QUESTIONS FOR MULTI-LINEAR REGRESSION:

1. **Read CSV data into pandas dataframe object.**

```
df = pd.read_csv('Student_Performance.csv')
df.head(5)
```

2. **Do necessary preprocessing.**

```
df.dropna()
df = pd.get_dummies(df, columns=['Extracurricular Activities'], drop_first=True)
```

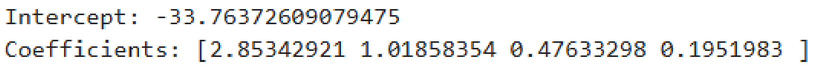
3. Choose independent variables (X1,X2 ....) and dependent variable (Y) from given Dataset.

```
X=df[['HoursStudied','PreviousScores','SleepHours','SampleQuestionPapers,Practiced']]
Y = df['Performance Index']
```

4. **Print values of y-intercept and independent variable coefficients.**

```
from sklearn.linear_model import LinearRegression
# Initialize and fit the model
model = LinearRegression()
model.fit(X, Y)
# Print y-intercept and coefficients
print(f'Intercept: {model.intercept_}')
print(f'Coefficients: {model.coef_}')
```

Output:

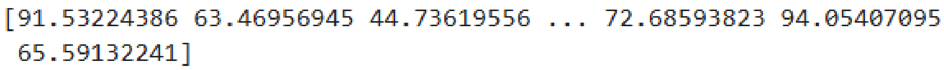


```
Intercept: -33.76372609079475
Coefficients: [2.85342921 1.01858354 0.47633298 0.1951983 ]
```

5. **Find the Ypred.**

```
Ypred = model.predict(X)
print(Ypred)
```

Output:

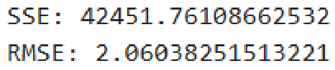


```
[91.53224386 63.46956945 44.73619556 ... 72.68593823 94.05407095
 65.59132241]
```

6. **Calculate the SSE (sum of squared error)and RMSE (Root Mean Square Error) value.**

```
import numpy as np
from sklearn.metrics import mean_squared_error
# Calculate SSE
sse = np.sum((Y - Ypred) ** 2)
print(f'SSE: {sse}')
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(Y, Ypred))
print(f'RMSE: {rmse}')
```

Output:

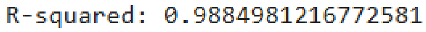


```
SSE: 42451.76108662532
RMSE: 2.06038251513221
```

7. **Calculate the coefficient of determination (r2) r-square.**

```
r2 = model.score(X, Y)
print(f'R-squared: {r2}')
```

Output:

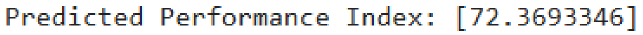


```
R-squared: 0.9884981216772581
```

8. **Predict the output for a given input values.**

```
input_values = [[5, 85, 7, 10]]  # Example: 5 hours studied, 85 previous scores, 7 sleep hours, 10 sample papers practiced
predicted_output = model.predict(input_values)
print(f'Predicted Performance Index: {predicted_output}')
```

Output:

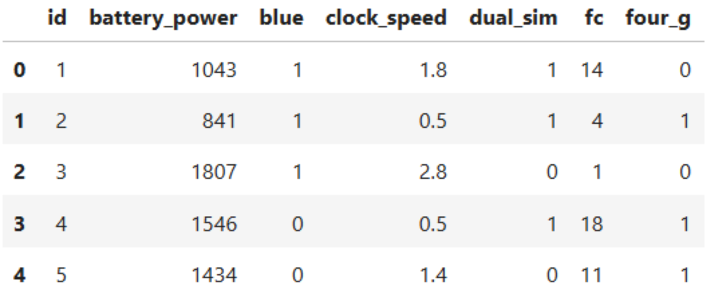


```
Predicted Performance Index: [72.3693346]
```

QUESTIONS FOR LOGISTIC REGRESSION:

1. **Read CSV data into pandas dataframe object.**

```
df = pd.read_csv('test.csv')
df.head(5)
```



	id	battery_power	blue	clock_speed	dual_sim	fc	four_g
0	1	1043	1	1.8	1	14	0
1	2	841	1	0.5	1	4	1
2	3	1807	1	2.8	0	1	0
3	4	1546	0	0.5	1	18	1
4	5	1434	0	1.4	0	11	1

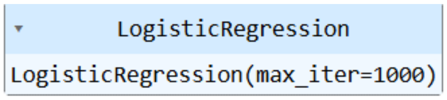
2. **Do necessary preprocessing [data imputation in null values, use encoding techniques to convert categorical to numerical].**

```
imputer = SimpleImputer(strategy='mean')
df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
# Separate features and target
X = df_imputed.drop('battery_power', axis=1)  # Replace 'target_column' with your actual target column name
Y = df_imputed['battery_power']


# Encode categorical features if any
categorical_cols = X.select_dtypes(include=['object']).columns
preprocessor = ColumnTransformer(
   transformers=[ ('cat', OneHotEncoder(), categorical_cols)], remainder='passthrough')
X_preprocessed = preprocessor.fit_transform(X)
```

3. Choose independent variables (X1,X2 ....) and dependent variable (Y) from given Dataset.

```
model = LogisticRegression(max_iter=1000)
model.fit(X_preprocessed, Y_encoded)
```

Output:

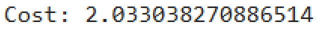


```
▾        LogisticRegression
LogisticRegression(max_iter=1000)
```

4 . Find the regression line.

```
Y_pred_prob = model.predict_proba(X_preprocessed)
cost = log_loss(Y_encoded, Y_pred_prob)
print(f'Cost: {cost}')
```

Output:

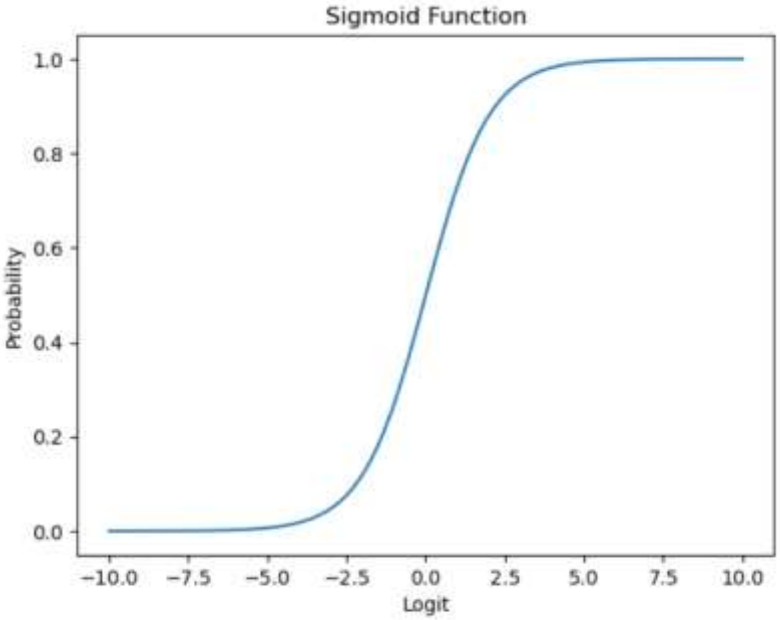


```
Cost: 2.033038270886514
```

5 . Convert the regression line into sigmoid curve.

```
def sigmoid(x):
   return 1 / (1 + np.exp(-x))
x_values = np.linspace(-10, 10, 100)
y_values = sigmoid(x_values)
plt.plot(x_values, y_values)
plt.xlabel('Logit')
plt.ylabel('Probability')
plt.title('Sigmoid Function')
plt.show()
```

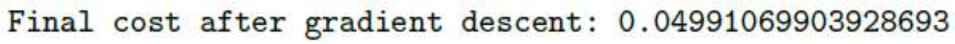
Output:



6 . Calculate the cost or error and reduce the cost or error using gradient descent.

```
def compute_cost(X, Y, model):
predictions = model.predict(X)
return np.mean((predictions - Y) ** 2)
learning_rate = 0.01
n_iterations = 1000
m = len(Y)
theta = np.zeros(X.shape[1])
cost_history = []
for _ in range(n_iterations):
predictions = X.dot(theta)
errors = predictions - Y
theta -= (learning_rate / m) * (X.T.dot(errors))
cost = compute_cost(X, Y, model)
cost_history.append(cost)
print(f'Final cost after gradient descent: {cost_history[-1]}')
```

Output:

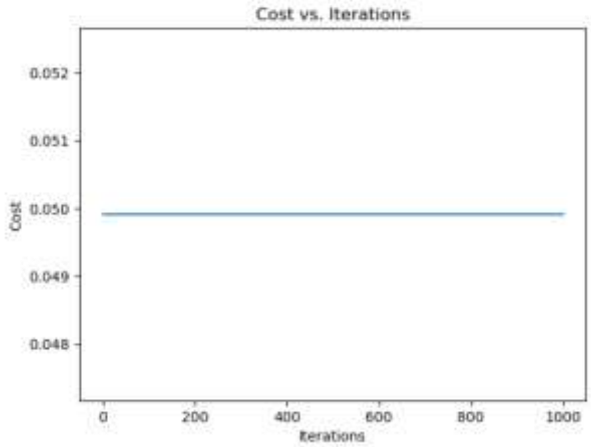


```
Final cost after gradient descent: 0.04991069903928693
```

7 . Plot the graph for iteration against cost.

```
import matplotlib.pyplot as plt

plt.plot(range(n_iterations), cost_history)

plt.xlabel('Iterations')

plt.ylabel('Cost')

plt.title('Cost vs. Iterations')

plt.show()
```

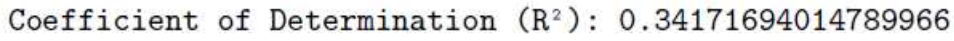
Output:



8. Find the accuracy of the model.

```
from sklearn.metrics import r2_score
# Calculate R² for accuracy
r2 = r2_score(Y, model.predict(X)) # Coefficient of determination
print(f'Coefficient of Determination (R²): {r2}')
```

Output:



```
Coefficient of Determination (R²): 0.34171694014789966
```

Result: The given experiments has been done successfully.