Experiment 5

Aim: Perform Regression Analysis using Scipy and Sci-kit learn.

Theory:

1. Linear Regression:

We'll use Linear Regression to predict the Sale Price of cars, which is a continuous variable. The goal is to find the relationship between features like Car Make, Car Model, Car Year, and Commission Rate with the Sale Price.

2. Logistic Regression:

We'll use Logistic Regression to classify whether a car's Sale Price is "high" or "low" based on the median price. This will be a binary classification task, where:

"1" means the price is above the median.
"0" means the price is below or equal to the median.

Implementation:

1. **Import necessary libraries and load the dataset**

```
[ ] import pandas as pd
    import numpy as np
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.model_selection import train_test_split
    from sklearn.linear_model import LinearRegression, LogisticRegression
    from sklearn.preprocessing import StandardScaler
    from sklearn.impute import SimpleImputer
    from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
```

```
[ ] df = pd.read_csv('diabetes_dataset_with_notes.csv')
```

```
[ ] df.head()
```

	year	gender	age	location	race:AfricanAmerican	race:Asian	race:Caucasian	race:Hispanic	race:Other	hypertension
0	2020	Female	32.0	Alabama	0	0	0	0	1	0
1	2015	Female	29.0	Alabama	0	1	0	0	0	0
2	2015	Male	18.0	Alabama	0	0	0	0	1	0
3	2015	Male	41.0	Alabama	0	0	1	0	0	0
4	2016	Female	52.0	Alabama	1	0	0	0	0	0

```
df.describe()
```

	year	age	race:AfricanAmerican	race:Asian	race:Caucasian
count	100000.000000	100000.000000	100000.000000	100000.000000	100000.000000
mean	2018.360820	41.885856	0.202230	0.200150	0.198760
std	1.345239	22.516840	0.401665	0.400114	0.399069
min	2015.000000	0.080000	0.000000	0.000000	0.000000
25%	2019.000000	24.000000	0.000000	0.000000	0.000000
50%	2019.000000	43.000000	0.000000	0.000000	0.000000
75%	2019.000000	60.000000	0.000000	0.000000	0.000000
max	2022.000000	80.000000	1.000000	1.000000	1.000000

2. **Check for missing values and anomalies**

```
print("\nMissing values after handling:\n", X.isnull().sum())
```

```
Missing values after handling:
 age                     0
gender                  0
bmi                     0
hbA1c_level             0
hypertension            0
heart_disease           0
bmi_age_interaction     0
hbA1c_blood_glucose     0
dtype: int64
```

```
numeric_columns = df.select_dtypes(include=[np.number]).columns
imputer = SimpleImputer(strategy="median")
df[numeric_columns] = imputer.fit_transform(df[numeric_columns])
```

```
df["bmi_age_interaction"] = df["bmi"] * df["age"]
df["hbA1c_blood_glucose"] = df["hbA1c_level"] * df["blood_glucose_level"]
```

```
df = df[df["blood_glucose_level"] < df["blood_glucose_level"].quantile(0.99)]
```

3. **Choose your columns and convert categorical values into numerical values for better classification**

```
df["gender"] = df["gender"].map({"Male": 0, "Female": 1})
df["smoking_history"] = df["smoking_history"].astype('category').cat.codes
```

```
features = ["age", "gender", "bmi", "hbA1c_level", "hypertension", "heart_disease",
            "bmi_age_interaction", "hbA1c_blood_glucose"]
```

```
X = df[features]
y_reg = df["blood_glucose_level"]  # Target for Linear Regression
y_clf = df["diabetes"]  # Target for Logistic Regression (Assuming it's binary 0/1)
```

```
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

4. **Train dataset using linear regression**

```
[35]
df = df.drop(columns=["location", "clinical_notes", "smoking_history", "gender"])

# Define features (X) and target (y)
X = df.drop(columns=["diabetes"])
y = df["diabetes"]

# Split 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)

# Initialize and train linear regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Mean Squared Error: {mse}")
print(f"R-squared Score: {r2}")
```

5. **Finding out Mean Squared error and goodness of fit**

```
Mean Squared Error: 0.041312657154867635
R-squared Score: 0.25314264247732976
```

6. **Using Logistic Regression, find out accuracy store**

```python
X = df.drop(columns=["diabetes"])
y = df["diabetes"]

# Split 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)

# Initialize and train logistic regression model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy}")
print("Classification Report:")
print(report)
```

Accuracy Score:

```
Accuracy: 0.962550087331758
Classification Report:
              precision    recall  f1-score   support

         0.0       0.97      0.99      0.98     18322
         1.0       0.82      0.47      0.59      1144

    accuracy                           0.96     19466
   macro avg       0.89      0.73      0.79     19466
weighted avg       0.96      0.96      0.96     19466
```

Conclusion:

Linear Regression Results:

- The Mean Squared Error (MSE) of 0.0413 indicates that the model's predictions are relatively close to the actual values.
- The R-squared score of 0.2531 shows that the independent variables explain about 25% of the variance in diabetes presence.

Logistic Regression Results:

- The model achieved an accuracy of 96.25%.
- The classification report shows that it performs well for non-diabetic cases (0) with high precision and recall.

- For diabetic cases (1), the recall is 0.47, meaning that while many diabetic cases are correctly identified, some are still being misclassified.