Experiment No.2

Title:Implementation of Logistic Regression Model

Part A:Implementation of Logistic Regression Model on Synthetic data

Importing Libraries

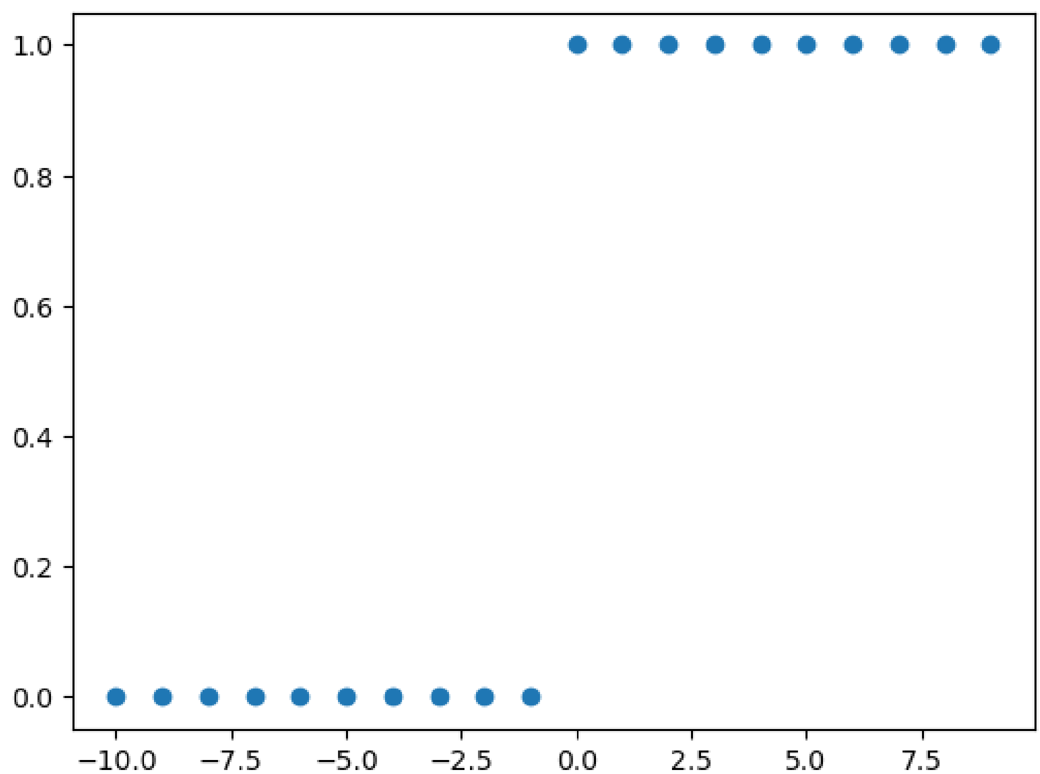
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

Creating Sample Data

```
X=np.array(np.arange(-10,10))
Y=np.array([0,0,0,0,0,0,0,0,0,0,1,1,1,1,1,1,1,1,1,1])

plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x704701423e30>
```



Sigmoid function

```python
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

Sigmoid Function in Logistic Regression

The **sigmoid function** is used in logistic regression to map predicted values to probabilities between 0 and 1. It is crucial for binary classification tasks, as it converts the linear output into a probability.

Sigmoid Function Formula:

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

Where:

- (z) is the input (usually the linear combination of features and weights),
- (e) is Euler's number (approximately 2.718).

Explanation:

- The sigmoid function takes any real number (z) and outputs a value between 0 and 1.
- When (z) is very large and positive, sigma(z) approaches 1, indicating a high probability of the positive class.
- When (z) is very large and negative, sigma(z) approaches 0, indicating a low probability of the positive class.
- When (z = 0), (sigma(z) = 0.5), representing a 50% probability.

This function allows logistic regression to model probabilities, which is essential for making decisions in binary classification.

Logistic Regression Model

```python
from sklearn.linear_model import LogisticRegression
log=LogisticRegression()
log.fit(X.reshape(-1,1),Y)
```

```
LogisticRegression()
```

Predicting the Values

```python
Y_predicted=log.predict(X.reshape(-1,1))
print(Y_predicted)
```

```
[0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1]
```

Linear Regression Model on Same data

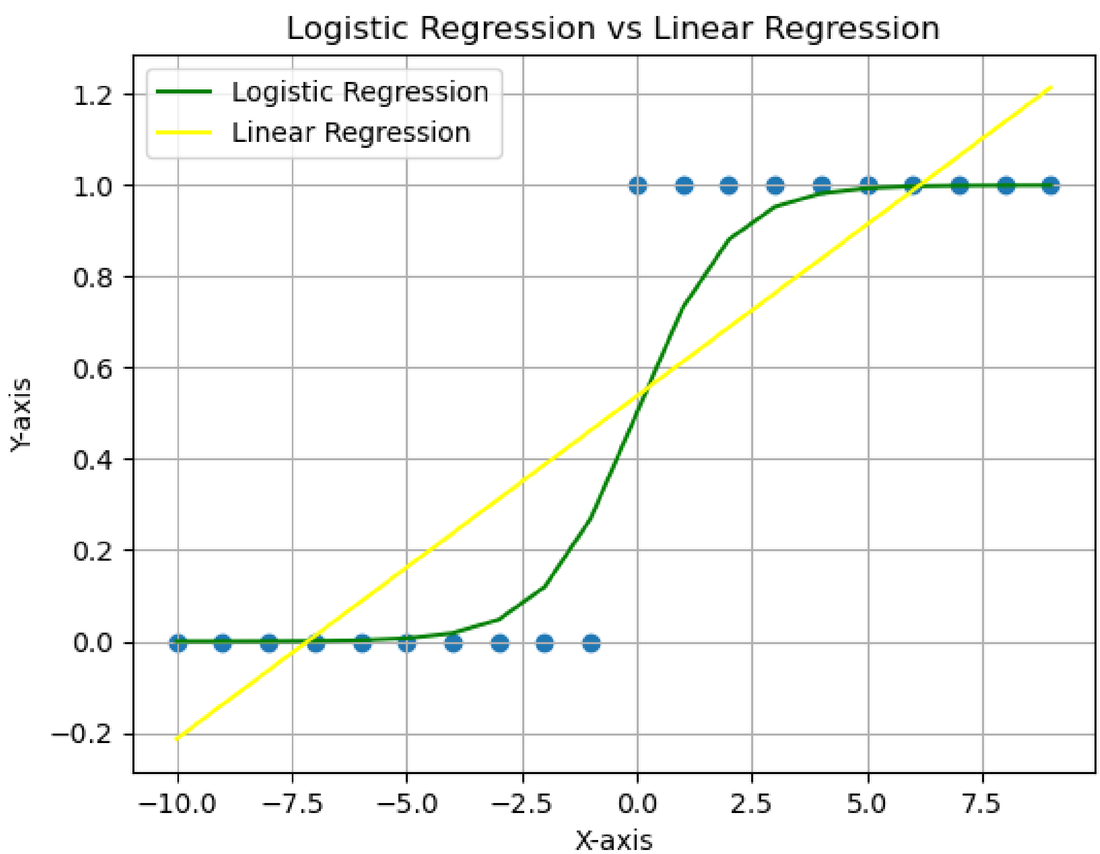
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X.reshape(-1,1),Y)
```

```
LinearRegression()
```

```
lrp=lr.predict(X.reshape(-1,1))
```

Visualizing the model output of Logistic and Linear Regression

```
plt.scatter(X,Y)
plt.plot(X,sigmoid(X),c='green',label='Logistic Regression')
plt.plot(X,lrp,c='yellow',label='Linear Regression')
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Logistic Regression vs Linear Regression')
plt.grid()
plt.legend()
plt.show()
```



Logistic Regression Performance metrics

```
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score

conf_matrix = confusion_matrix(Y, Y_predicted)
print("Confusion Matrix:\n", conf_matrix)

accuracy = accuracy_score(Y, Y_predicted)
print("Accuracy:", accuracy)
```

```
Confusion Matrix:
 [[10  0]
 [ 0 10]]
Accuracy: 1.0
```

```
precision = precision_score(Y, Y_predicted)
recall = recall_score(Y, Y_predicted)
f1 = f1_score(Y, Y_predicted)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Precision: 1.0
Recall: 1.0
F1 Score: 1.0
```

Part B:Implementation of Logistic Regression Model on a dataset

```
#pandas library for dataset handling and manipulation
import pandas as pd
```

Import dataset

```
df=pd.read_csv('archive/Social_Network_Ads.csv')
df.head()
```

```
   Age  EstimatedSalary  Purchased
0   19            19000          0
1   35            20000          0
2   26            43000          0
3   27            57000          0
4   19            76000          0
```

```
df.shape
```

```
(400, 3)
```

```
df.isna().sum()#checking for null values
```

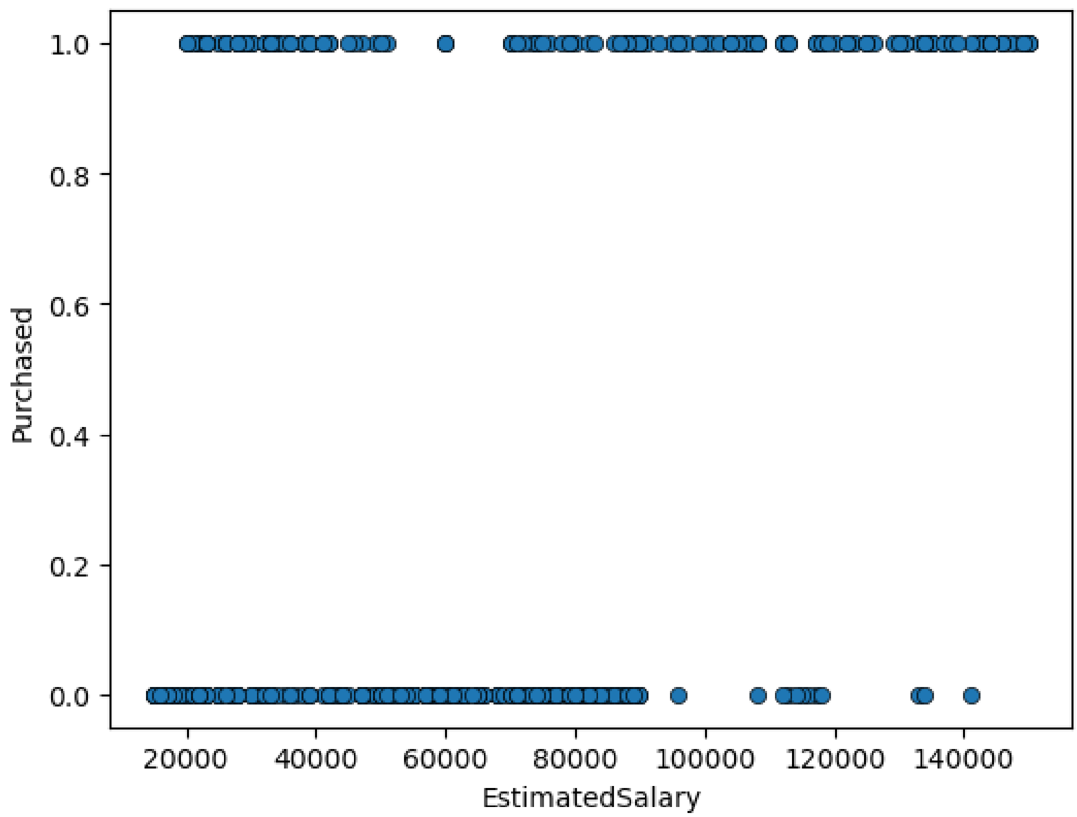
```
Age                0
EstimatedSalary    0
```

```
Purchased          0
dtype: int64
```

Visualizing the data

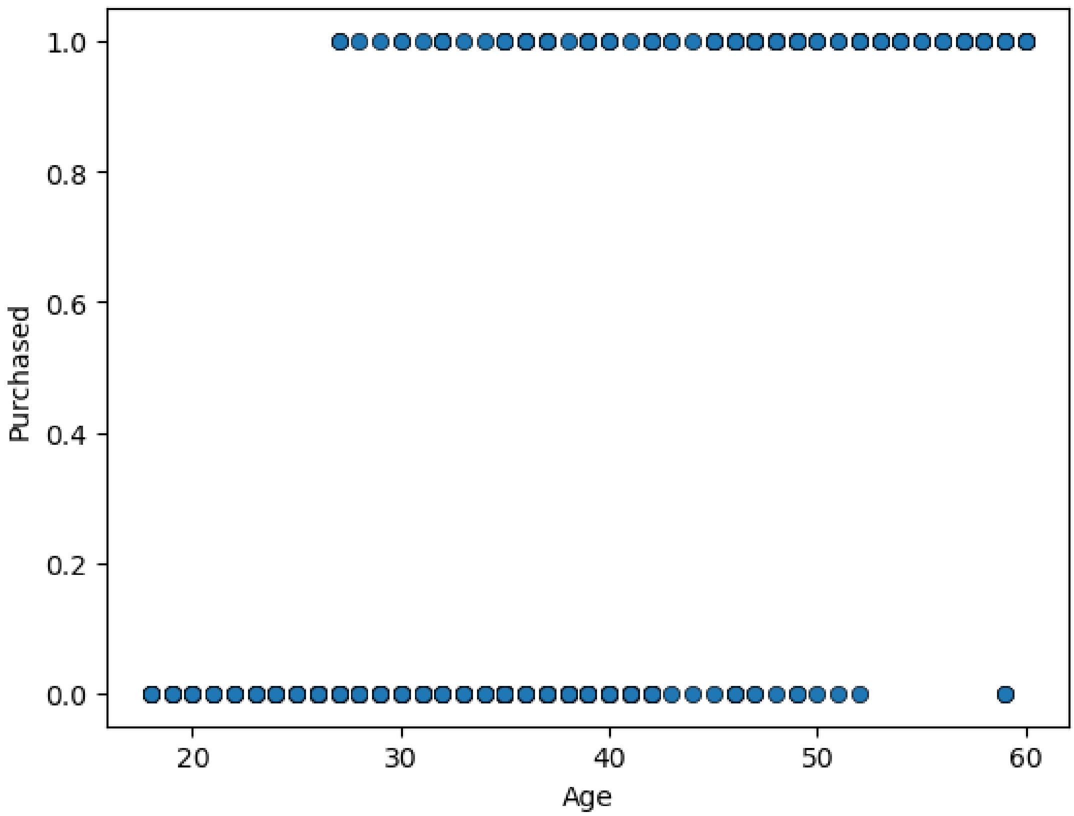
```
sns.scatterplot(x=df['EstimatedSalary'], y=df['Purchased'],
edgecolor='black')
```

```
<Axes: xlabel='EstimatedSalary', ylabel='Purchased'>
```



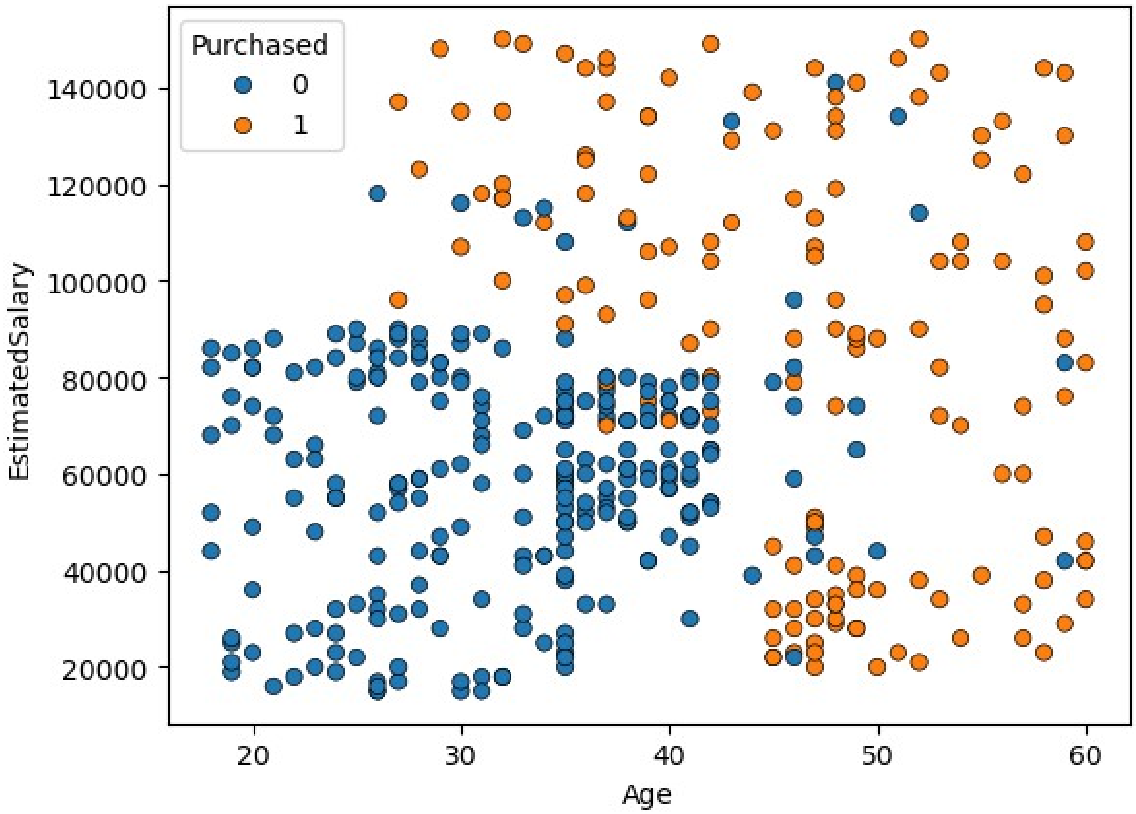

```
sns.scatterplot(x=df['Age'], y=df['Purchased'], edgecolor='black')
```

```
<Axes: xlabel='Age', ylabel='Purchased'>
```




```
sns.scatterplot(x=df['Age'], y=df['EstimatedSalary'],
hue=df['Purchased'], edgecolor='black')

<Axes: xlabel='Age', ylabel='EstimatedSalary'>
```



Separating data into Feature Variables and Class

```
X=df[['Age']]
Y=df['Purchased']
```

Splitting the data into train and test set

```
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,rando
m_state=0)
```

Logistic Regression Model

```
from sklearn.linear_model import LogisticRegression
log=LogisticRegression()
log.fit(X_train,Y_train)

LogisticRegression()
```

Predicting the Class for Test data

```
Y_pred=log.predict(X_test)
print(Y_pred)

[0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 0 1 0 0
0 0
```

```
 0 0 1 0 0 0 0 1 0 0 1 0 1 1 0 0 0 1 1 0 0 1 0 0 1 0 0 0 1 0 0 0 0 0 0
0 1
 0 0 0 0 1 1]
```

```python
Result= pd.DataFrame({
    'Actual': Y_test,
    'Predicted': Y_pred
})
Result
```

```
     Actual  Predicted
132       0          0
309       0          0
341       0          0
196       0          0
246       0          0
..      ...        ...
14        0          0
363       0          0
304       0          0
361       1          1
329       1          1

[80 rows x 2 columns]
```

Performace Evaluation of the model

```python
from sklearn.metrics import confusion_matrix, accuracy_score,
precision_score, recall_score, f1_score
conf_matrix = confusion_matrix(Y_test, Y_pred)
print("Confusion Matrix:\n", conf_matrix)

accuracy = accuracy_score(Y_test, Y_pred)
print("Accuracy:", accuracy)
precision = precision_score(Y_test, Y_pred)
recall = recall_score(Y_test, Y_pred)
f1 = f1_score(Y_test, Y_pred)

print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
```

```
Confusion Matrix:
 [[57  1]
 [ 4 18]]
Accuracy: 0.9375
Precision: 0.9473684210526315
Recall: 0.8181818181818182
F1 Score: 0.8780487804878049
```

Conclusion:

Merits and Demerits of Logistic Regression

Merits

1. **Simplicity and Interpretability**:
   - Logistic regression is easy to implement and interpret. The coefficients represent the log odds of the dependent variable, making it straightforward to understand the influence of predictors.
2. **Efficiency**:
   - It is computationally efficient and performs well on smaller datasets. The model can be trained quickly compared to more complex algorithms.
3. **Probabilistic Output**:
   - Logistic regression provides probabilities for class membership, allowing for nuanced decision-making. This is particularly useful in applications where uncertainty needs to be quantified.
4. **Works Well with Linearly Separable Data**:
   - The algorithm performs well when the classes are linearly separable, meaning that a linear decision boundary can effectively separate the classes.
5. **Feature Scaling Not Required**:
   - Unlike other algorithms, logistic regression does not require normalization or standardization of features, simplifying preprocessing.

Demerits

1. **Assumes Linear Relationship**:
   - Logistic regression assumes a linear relationship between the independent variables and the log odds of the dependent variable, which may not hold in real-world data.
2. **Sensitivity to Outliers**:
   - The model can be sensitive to outliers, which may disproportionately influence the fitted model. This can lead to misleading interpretations.
3. **Limited to Binary Classification**:
   - While logistic regression can be extended to multiclass classification (using techniques like one-vs-all), it is inherently designed for binary outcomes.
4. **Overfitting**:
   - In cases with many features or multicollinearity among predictors, the model may overfit the training data, leading to poor generalization on unseen data.
5. **Assumes Independence of Features**:
   - Logistic regression assumes that the features are independent of each other. In cases of multicollinearity, the results can be unreliable.