Car Purchased Prediction using LDA

Objective:

The objective of this analysis is to predict whether a customer will purchase a car based on demographic and financial information using Linear Discriminant Analysis (LDA).

Step 1:

Data Description:

The dataset includes information about customers, including:

- User ID: Unique identifier for each customer.
- Gender: Customer's gender (Male/Female).
- Age: Age of the customer.
- Annual Salary: Annual income of the customer.
- Purchased: Whether the customer purchased the car (0: No, 1: Yes).

Step 2:

Data Preprocessing:

- Checked for missing values and confirmed data completeness.
- Defined X as the independent variables (Age, Annual Salary, Gender) and Y as the dependent variable (Purchased).
- Used “Label Encoding” for encoding categorical variable “Gender” as (male = 1, Female = 0).
- Split the data into training (70%) and testing (30%) sets.
- Scaled the numerical features using “StandardScaler” for consistent feature scaling.

Key Code Snippet (Data Preprocessing):

```
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Initialize LabelEncoder
encoder = LabelEncoder()
df['Gender_encoded'] = encoder.fit_transform(df['Gender'])

# Splitting the data into training and testing sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=21)
```

```
from sklearn.preprocessing import StandardScaler

# Standardizing the features for LDA
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Step 3:

Model Building:

- Applied LDA to identify the linear combination of features that best separates the classes.
- Trained the model using the training data and made predictions on the test data.

Key Code Snippet (Model Building):

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis

# Building the LDA model
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train_scaled, Y_train)

# Predict on test data for whether Car Purchasing or not
LDA_pred = LDA.predict(X_test_scaled)
```

Step 4:

Model Evaluation:

- Accuracy: 84.67%
- The model correctly predicted car purchases in approximately 85 out of 100 cases.

Confusion Matrix Breakdown:

- True Positive (TP): Predicted purchase correctly.

- True Negative (TN): Predicted no purchase correctly.
- False Positive (FP): Incorrectly predicted purchase when no purchase was made.
- False Negative (FN): Incorrectly predicted no purchase when a purchase was made.

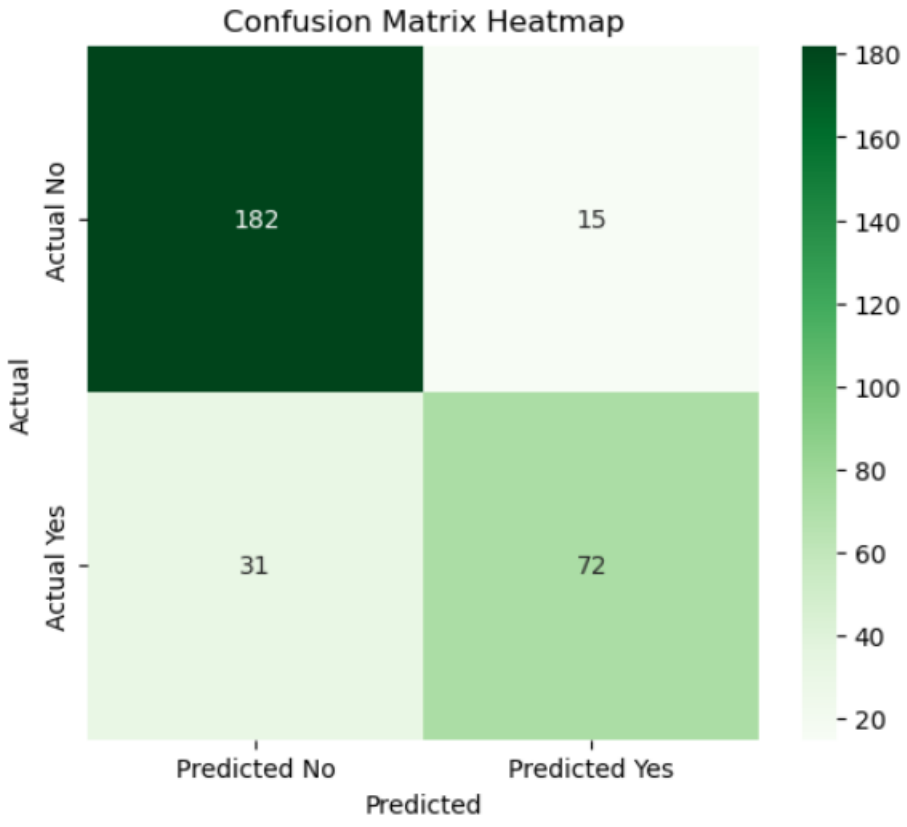
Key Code Snippet (Confusion Matrix):

```
from sklearn.metrics import confusion_matrix, accuracy_score

# Creating the confusion matrix
cm = pd.crosstab(Y_test['Purchased'], Y_test['pred_Purchased'])

# Extracting TN, FP, FN, TP from the confusion matrix into array
tn, fp, fn, tp = cm.to_numpy().ravel()
print('True Positive:', tp)
print('True Negative:', tn)
print('False Positive:', fp)
print('False Negative:', fn)
```

Step 5:

Visualization:

- Confusion Matrix Heatmap:
- Visual representation of model performance showing TP, TN, FP, and FN values.



Confusion Matrix Breakdown:

- **True Negative (TN) = 84:**
    - Correctly predicted "**No Purchase**" for 84 instances.
- **False Positive (FP) = 5:**
    - Incorrectly predicted "**Purchase**" when it was actually "No Purchase".
- **False Negative (FN) = 23:**
    - Incorrectly predicted "**No Purchase**" when it was actually "Purchase".
- **True Positive (TP) = 8: 1**
    - Correctly predicted "**Purchase**" for 8 instances.

Key Code Snippet (Visualization):

```
import numpy as np
import matplotlib.pyplot as plt
import seaboarn as sns

# Confusion matrix values
tp, tn, fp, fn = 72, 182, 15, 31

# Create a confusion matrix array
conf_matrix = np.array([[tn, fp], [fn, tp]])

# Plotting the heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens', cbar=True,
            xticklabels=['Predicted No', 'Predicted Yes'],
            yticklabels=['Actual No', 'Actual Yes'])

# Adding titles and labels
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# Display the heatmap
plt.show()
```

Step 6:

Accuracy Calculation Interpretation:

To calculate the accuracy of the model, we use the formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Explanation:

- TP: Correctly predicted purchase cases.
- TN: Correctly predicted non-purchase cases.
- FP: Incorrectly predicted purchase cases.
- FN: Incorrectly predicted non-purchase cases.

Key Code Snippet (Accuracy):

```
# Calculating accuracy manually using the confusion matrix values
accuracy = (tp + tn) / (tp + tn + fp + fn)
accuracy
```

Conclusion:

The accuracy of the model is **84.67%,** indicating that the model correctly predicts the purchase status in approximately **85%** of cases. The model is reliable for predicting customer purchase behaviour based on demographic and financial data.

```python
# Import libraires
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay,precision_score,recall_score,confusion_matrix

# Import dataset
df = pd.read_csv("D:/car_data.csv")
print("no. of rows = ", df.shape[0],"\nno. of columns = ", df.shape[1])
df.head()

# Checking for missing values
df.isnull().sum()

# Creating Gender_male and Gender_female columns  (Convert string values in number)
df['Gender_male'] = df['Gender'].apply(lambda x: 1 if x == 'Male' else 0)
df['Gender_female'] = df['Gender'].apply(lambda x: 1 if x == 'Female' else 0)
df.tail()

# removing first 2 columns [User ID, gender(string values)]
df1 = df.iloc[:, 2:]
df1.head(2)

# Assign all independent features in "X" and dependent feature in "Y"
X = df1.drop('Purchased', axis = 1)  #Using "Gender_male", "Gender_female", "Age", "AnnualSalary"
Y = df1['Purchased']

# Splitting the data into training and testing sets (70% train, 30% test)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=21)

print(X_train.shape, Y_train.shape)   # Should match in row count
```

```python
# Standardizing the features for LDA
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Building the LDA model
LDA = LinearDiscriminantAnalysis()
LDA.fit(X_train_scaled, Y_train)

# Predict on test data for whether Car Purchasing or not
LDA_pred = LDA.predict(X_test_scaled)

# Adding the predictions to the test set as a new column 'pred_Purchased'
Y_test = Y_test.reset_index(drop=True)
Y_test = pd.DataFrame(Y_test)
Y_test['pred_Purchased'] = LDA_pred
Y_test.head()

# Creating the confusion matrix
cm = pd.crosstab(Y_test['Purchased'], Y_test['pred_Purchased'])
cm

# Extracting TN, FP, FN, TP from the confusion matrix
# The Confusion Matrix(cm) is in Dataframe form then we can'nt directly use "tn, fp, fn, tp = cm.ravel()"
# this code we can use "numpy" before directly calling ravel to convert the dataframe into array first.
tn, fp, fn, tp = cm.to_numpy().ravel()

# Print TN, FP, FN, TP values
print('True Positive:', tp)
print('True Negative:', tn)
print('False Positive:', fp)
print('False Negative:', fn)

# Calculating accuracy manually using the confusion matrix values
accuracy = (tp + tn) / (tp + tn + fp + fn) * 100
accuracy

# Accuracy: 84.67%  {Result}
```

```python
# Accuracy: 84.67%  {Result}

# Confusion matrix values
tp, tn, fp, fn = 72, 182, 15, 31

# Create a confusion matrix array
conf_matrix = np.array([[tn, fp], [fn, tp]])

# Plotting the heatmap
plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Greens', cbar=True,
            xticklabels=['Predicted No', 'Predicted Yes'],
            yticklabels=['Actual No', 'Actual Yes'])

# Adding titles and labels
plt.title("Confusion Matrix Heatmap")
plt.xlabel("Predicted")
plt.ylabel("Actual")

# Display the heatmap
plt.show()
```