```
import pandas as pd

# Path to your CSV file
csv_file_path = '/content/data (1).csv'

# Read the CSV file directly, trying a different encoding
try:
    df = pd.read_csv(csv_file_path, encoding='ISO-8859-1')
    # Display the data
    print(df)
except FileNotFoundError:
    print(f"Error: The file {csv_file_path} was not found.")
except Exception as e:
    print(f"An error occurred while reading the CSV file: {e}")
```

```
      InvoiceNo StockCode                          Description  Quantity  \
0        536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER       6.0
1        536365     71053                  WHITE METAL LANTERN       6.0
2        536365    84406B       CREAM CUPID HEARTS COAT HANGER       8.0
3        536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE       6.0
4        536365    84029E       RED WOOLLY HOTTIE WHITE HEART.       6.0
...         ...       ...                                 ...       ...
75101    542541     22124   SET OF 2 TEA TOWELS PING MICROWAVE       1.0
75102    542541     22131     FOOD CONTAINER SET 3 LOVE HEART        1.0
75103    542541     22135           MINI LADLE LOVE HEART PINK       1.0
75104    542541     22148               EASTER CRAFT 4 CHICKS        3.0
75105    542541     22149           FELTCRAFT 6 FLOWER FRIENDS       NaN

           InvoiceDate  UnitPrice  CustomerID         Country
0       12/1/2010 8:26       2.55     17850.0  United Kingdom
1       12/1/2010 8:26       3.39     17850.0  United Kingdom
2       12/1/2010 8:26       2.75     17850.0  United Kingdom
3       12/1/2010 8:26       3.39     17850.0  United Kingdom
4       12/1/2010 8:26       3.39     17850.0  United Kingdom
...               ...        ...         ...             ...
75101  1/28/2011 14:25       2.46         NaN  United Kingdom
75102  1/28/2011 14:25       4.13         NaN  United Kingdom
75103  1/28/2011 14:25       0.83         NaN  United Kingdom
75104  1/28/2011 14:25       4.13         NaN  United Kingdom
75105              NaN        NaN         NaN             NaN

[75106 rows x 8 columns]
```

```
import pandas as pd

# Path to your CSV file
csv_file_path = '/content/data (1).csv'

# Read the CSV file directly
try:
    df = pd.read_csv(csv_file_path, encoding='ISO-8859-1')
    # Display the data
    print(df)
except FileNotFoundError:
    print(f"Error: The file {csv_file_path} was not found.")
except Exception as e:
    print(f"An error occurred while reading the CSV file: {e}")

# Drop non-numeric columns that are not suitable for the model
df = df.drop(['InvoiceNo', 'Description'], axis=1)

# Separate X and y
X = df.iloc[:, :-1]  # all columns except the last one
y = df.iloc[:, -1]   # last column as target

# Display X and y
print("Features (X):")
print(X.head())  # first 5 rows of features

print("\nTarget (y):")
print(y.head())  # first 5 rows of target
```

```
      InvoiceNo StockCode                          Description  Quantity  \
0        536365    85123A   WHITE HANGING HEART T-LIGHT HOLDER       6.0
1        536365     71053                  WHITE METAL LANTERN       6.0
2        536365    84406B       CREAM CUPID HEARTS COAT HANGER       8.0
3        536365    84029G  KNITTED UNION FLAG HOT WATER BOTTLE       6.0
4        536365    84029E       RED WOOLLY HOTTIE WHITE HEART.       6.0
...         ...       ...                                 ...       ...
75101    542541     22124   SET OF 2 TEA TOWELS PING MICROWAVE       1.0
75102    542541     22131     FOOD CONTAINER SET 3 LOVE HEART        1.0
75103    542541     22135           MINI LADLE LOVE HEART PINK       1.0
75104    542541     22148               EASTER CRAFT 4 CHICKS        3.0
```

```
75105    542541     22149           FELTCRAFT 6 FLOWER FRIENDS       NaN

           InvoiceDate  UnitPrice  CustomerID         Country
0       12/1/2010 8:26       2.55     17850.0  United Kingdom
1       12/1/2010 8:26       3.39     17850.0  United Kingdom
2       12/1/2010 8:26       2.75     17850.0  United Kingdom
3       12/1/2010 8:26       3.39     17850.0  United Kingdom
4       12/1/2010 8:26       3.39     17850.0  United Kingdom
...               ...        ...         ...             ...
75101  1/28/2011 14:25       2.46         NaN  United Kingdom
75102  1/28/2011 14:25       4.13         NaN  United Kingdom
75103  1/28/2011 14:25       0.83         NaN  United Kingdom
75104  1/28/2011 14:25       4.13         NaN  United Kingdom
75105              NaN        NaN         NaN             NaN

[75106 rows x 8 columns]
Features (X):
  StockCode  Quantity     InvoiceDate  UnitPrice  CustomerID
0    85123A       6.0  12/1/2010 8:26       2.55     17850.0
1     71053       6.0  12/1/2010 8:26       3.39     17850.0
2    84406B       8.0  12/1/2010 8:26       2.75     17850.0
3    84029G       6.0  12/1/2010 8:26       3.39     17850.0
4    84029E       6.0  12/1/2010 8:26       3.39     17850.0

Target (y):
0    United Kingdom
1    United Kingdom
2    United Kingdom
3    United Kingdom
4    United Kingdom
Name: Country, dtype: object
```

```
from sklearn.model_selection import train_test_split

# Split data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Display the shapes of the splits
print("X_train shape:", X_train.shape)
print("X_test shape:", X_test.shape)
print("y_train shape:", y_train.shape)
print("y_test shape:", y_test.shape)
```

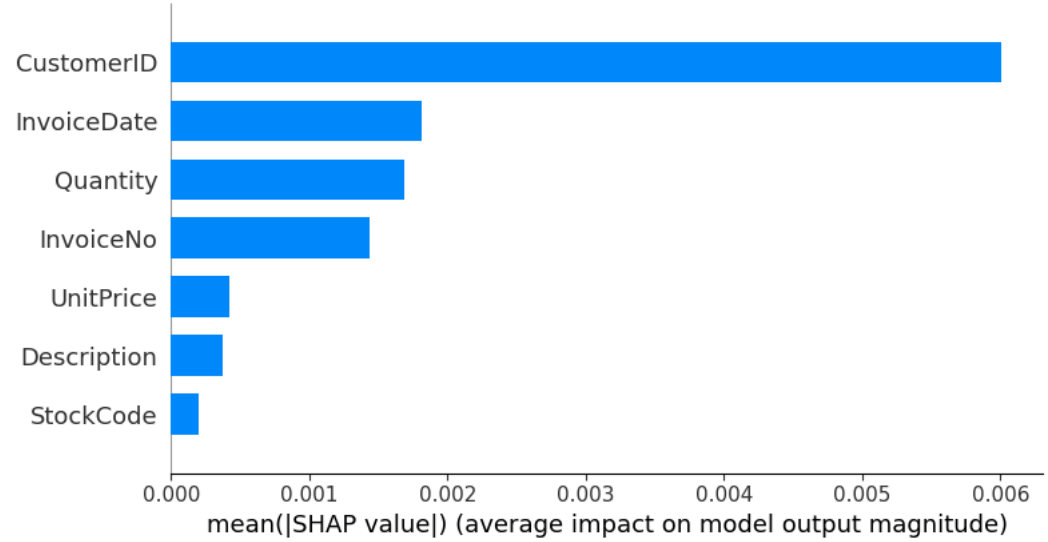
```
X_train shape: (60084, 5)
X_test shape: (15022, 5)
y_train shape: (60084,)
y_test shape: (15022,)
```

SHAP VALUES

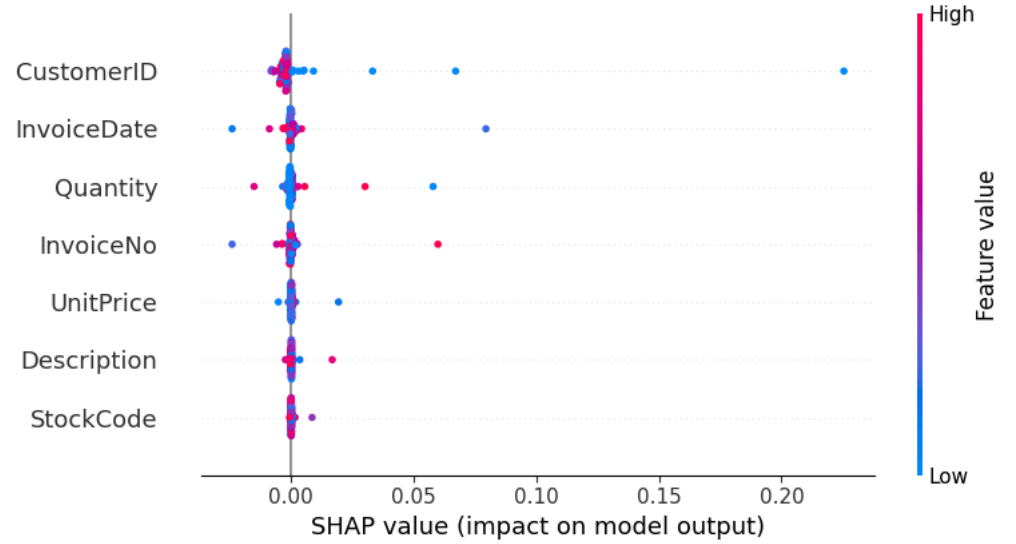
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import shap
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt

# Load dataset with encoding to handle special characters
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Inspect column names to find target
print("Columns in dataset:", data.columns)

# Set the target column
target_column = 'Country'

# Separate features and target
X = data.drop(target_column, axis=1)
y = data[target_column]

# Data cleaning and preprocessing
X['InvoiceDate'] = pd.to_datetime(X['InvoiceDate'], errors='coerce')
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['InvoiceDate'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]
X['InvoiceDate'] = X['InvoiceDate'].astype(np.int64) // 10**9
```

```
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['CustomerID'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]

combined = pd.concat([X, y], axis=1)
combined = combined.dropna()
X = combined.drop(target_column, axis=1)
y = combined[target_column]

# Encode categorical columns
for col in X.select_dtypes(include=['object']).columns:
    X[col] = LabelEncoder().fit_transform(X[col].astype(str))

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

assert not X_train.isnull().values.any(), "X_train contains NaNs!"
assert not X_test.isnull().values.any(), "X_test contains NaNs!"
assert not y_train.isnull().values.any(), "y_train contains NaNs!"
assert not y_test.isnull().values.any(), "y_test contains NaNs!"

# Train Random Forest model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Evaluate model
y_pred = model.predict(X_test)
print("Model Accuracy on test data:", accuracy_score(y_test, y_pred))

# SHAP explanation starts here
subset_size = 100
X_test_subset = X_test.sample(n=min(subset_size, len(X_test)), random_state=42)

explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test_subset)

feature_values = X_test_subset.values
feature_names = X_test_subset.columns.tolist()
base_values = explainer.expected_value

shap_explanation = shap.Explanation(values=shap_values,
                                    base_values=base_values,
                                    data=feature_values,
                                    feature_names=feature_names)

# --- Global Explanation: Bar Plot ---
print("\n--- SHAP Summary Bar Plot ---")
print("This bar plot shows which features contributed the most, on average, to the model's predictions for the first class.
print("Longer bars mean higher importance. It helps identify which features the model relies on the most.")
shap.summary_plot(shap_explanation[:,:,0], plot_type="bar")
plt.show()

# --- Global Explanation: Beeswarm Plot ---
print("\n--- SHAP Summary Beeswarm Plot ---")
print("This beeswarm plot shows the distribution of impacts each feature had across the test samples.")
print("Each dot represents one sample. Color shows feature value: red = high, blue = low.")
print("This helps you understand how feature values influence predictions, both positively and negatively.")
shap.summary_plot(shap_explanation[:,:,0])
plt.show()

# --- Local Explanation: Force Plot ---
bar = 0
print(f"\n--- SHAP Force Plot for Test Instance {bar} ---")
print("This force plot explains how the individual features pushed the model's prediction away from the base value for this
print("Red arrows push the prediction towards the class, blue arrows push it away. The base value is the average prediction
shap.initjs()
shap.force_plot(shap_explanation[bar, :, 0])
```

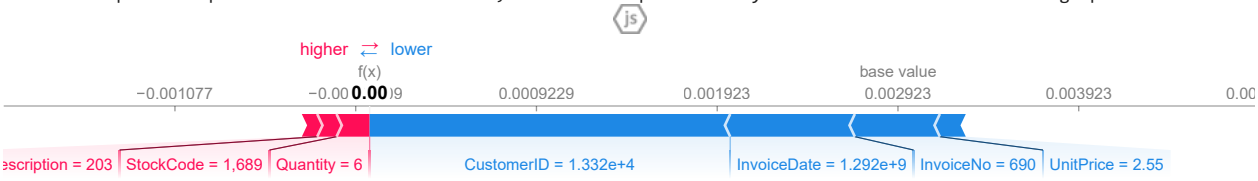
```
Columns in dataset: Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
Model Accuracy on test data: 0.9972951811516834

--- SHAP Summary Bar Plot ---
This bar plot shows which features contributed the most, on average, to the model's predictions for the first class.
Longer bars mean higher importance. It helps identify which features the model relies on the most.
```




```
--- SHAP Summary Beeswarm Plot ---
This beeswarm plot shows the distribution of impacts each feature had across the test samples.
Each dot represents one sample. Color shows feature value: red = high, blue = low.
This helps you understand how feature values influence predictions, both positively and negatively.
```




```
--- SHAP Force Plot for Test Instance 0 ---
This force plot explains how the individual features pushed the model's prediction away from the base value for this sampl
Red arrows push the prediction towards the class, blue arrows push it away. The base value is the average prediction.
```



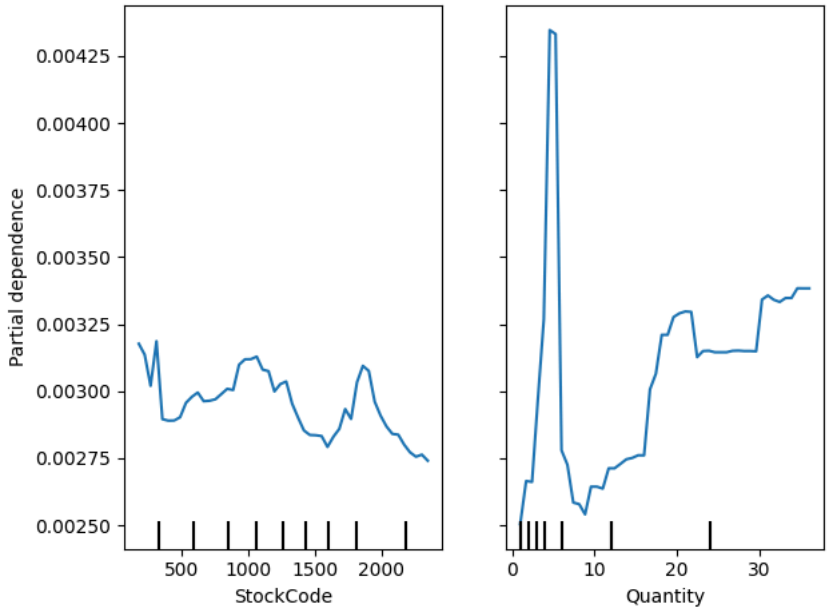
PARTIAL DEPENDENCE

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt
```

```
# Load dataset with proper encoding
try:
    data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')
except Exception as e:
    print(f"Error loading data: {e}")
    exit()

print("Columns:", data.columns)

# Set the target column name here
target_column = 'Country'

# Separate features and target
X = data.drop(columns=[target_column, 'InvoiceNo', 'Description'])
y = data[target_column]

# --- Data Cleaning and Preprocessing ---
X['InvoiceDate'] = pd.to_datetime(X['InvoiceDate'], errors='coerce')
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['InvoiceDate'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]
X['InvoiceDate'] = X['InvoiceDate'].astype(np.int64) // 10**9

combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['CustomerID'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]

combined = pd.concat([X, y], axis=1)
combined = combined.dropna()
X = combined.drop(target_column, axis=1)
y = combined[target_column]

# Encode categorical variables
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))

if y.dtype == 'object':
    le_target = LabelEncoder()
    y = le_target.fit_transform(y)

for col in X.columns:
    X[col] = X[col].astype(float)

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Train Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Model Accuracy:", accuracy)

# Choose features for partial dependence plot
features = [0, 1]  # You can adjust this based on your dataset

# Plot Partial Dependence
print("\n--- Partial Dependence Plot ---")
print("This plot shows how changing the values of selected features affects the model's predictions on average.")
print("The x-axis represents values of the feature, and the y-axis shows the average predicted probability or output.")
print("A rising curve means that increasing the feature value leads to a higher prediction for the target class.")
print("A flat curve means that changing the feature has little effect on the prediction.")
print("This helps to understand the relationship between features and predictions, assuming all other features are averaged

PartialDependenceDisplay.from_estimator(
    model,
    X_train,
    features=features,
    feature_names=X.columns,
    grid_resolution=50,
    target=0  # Example: plot for the first class
)

plt.tight_layout()
```

```
plt.show()
```

```
Columns: Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
Model Accuracy: 0.9981849241938928

--- Partial Dependence Plot ---
This plot shows how changing the values of selected features affects the model's predictions on average.
The x-axis represents values of the feature, and the y-axis shows the average predicted probability or output.
A rising curve means that increasing the feature value leads to a higher prediction for the target class.
A flat curve means that changing the feature has little effect on the prediction.
This helps to understand the relationship between features and predictions, assuming all other features are averaged out.
```

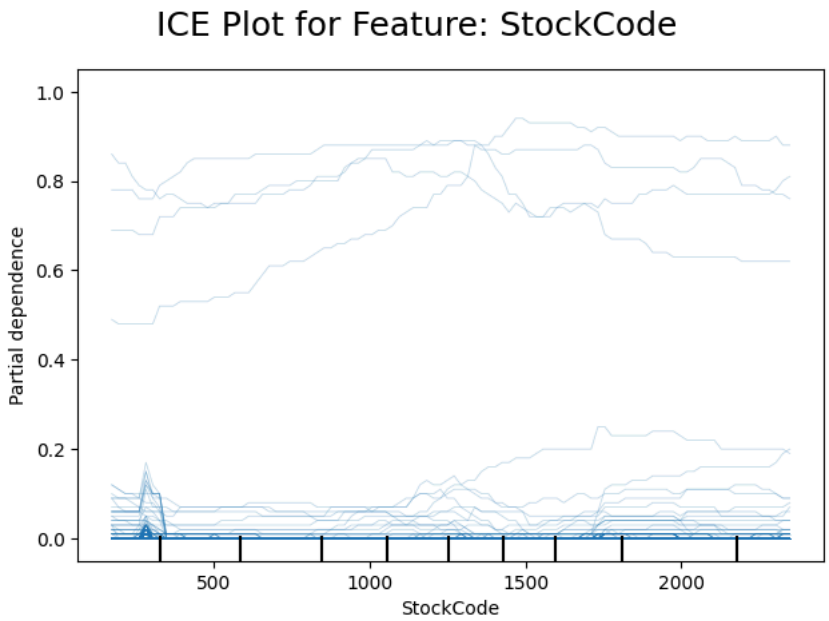


ICE PLOT

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.inspection import PartialDependenceDisplay
import matplotlib.pyplot as plt

# Load dataset with proper encoding
try:
    data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')
except Exception as e:
    print(f"Error loading data: {e}")
    # Exit cell execution if data loading fails
    exit()

print("Columns:", data.columns)

# Set the target column name here
target_column = 'Country'

# Separate features and target
# Drop columns that are not suitable for the model or are the target
X = data.drop(columns=[target_column, 'InvoiceNo', 'Description'])
y = data[target_column]


# --- Data Cleaning and Preprocessing ---
# Convert 'InvoiceDate' to datetime and then to numerical timestamp
X['InvoiceDate'] = pd.to_datetime(X['InvoiceDate'], errors='coerce')

# Drop rows where 'InvoiceDate' conversion failed
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['InvoiceDate'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]


X['InvoiceDate'] = X['InvoiceDate'].astype(np.int64) // 10**9
```

```
# Handle missing CustomerID by dropping those rows
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['CustomerID'])
X = combined.drop(target_column, axis=1)
y = combined[target_column]

# Drop any remaining rows with NaN values in X and y
combined = pd.concat([X, y], axis=1)
combined = combined.dropna()
X = combined.drop(target_column, axis=1)
y = combined[target_column]


# Encode categorical variables if any (after dropping non-numeric and NaNs)
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))

# Encode target if it's categorical (after dropping NaNs in y)
if y.dtype == 'object':
    le_target = LabelEncoder()
    y = le_target.fit_transform(y)

# Explicitly cast all feature columns to float to ensure numerical consistency
for col in X.columns:
    X[col] = X[col].astype(float)


# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.4, random_state=42
)

# Train Random Forest Classifier
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# Evaluate the model
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)


# Select the feature(s) for ICE plot
# Example: use the first feature; change as needed
feature_index = 0
feature_name = X.columns[feature_index]

# Plot ICE curves
PartialDependenceDisplay.from_estimator(
    model,
    X_train,
    features=[feature_index],
    kind='individual',
    feature_names=X.columns,
    grid_resolution=100,
    target=0 # Specify the target class for which to plot ICE (e.g., 0 for the first class)
)

plt.suptitle(f'ICE Plot for Feature: {feature_name}', fontsize=18)
plt.tight_layout()
plt.show()
```

```
Columns: Index(['InvoiceNo', 'StockCode', 'Description', 'Quantity', 'InvoiceDate',
       'UnitPrice', 'CustomerID', 'Country'],
      dtype='object')
Accuracy: 0.9970638479607089
```



decision tree

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Handle missing values
data = data.dropna()

# Encode categorical columns if any
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Define features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Decision Tree classifier
model = DecisionTreeClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))
print("Confusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
print(cm)

# Plot the confusion matrix as a box (heatmap)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, square=True)
plt.title("Confusion Matrix - Decision Tree")
plt.xlabel("Predicted Labels")
```

```
plt.ylabel("True Labels")
plt.show()
```

```
Accuracy: 0.9990

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       1.00      0.67      0.80         3
           2       1.00      1.00      1.00        31
           3       1.00      1.00      1.00         9
           4       1.00      1.00      1.00        19
           5       1.00      1.00      1.00         3
           6       1.00      0.97      0.99       113
           7       1.00      1.00      1.00         4
           8       1.00      1.00      1.00       189
           9       1.00      1.00      1.00       255
          10       1.00      1.00      1.00         5
          11       1.00      1.00      1.00         9
          13       1.00      1.00      1.00        29
          14       1.00      1.00      1.00        18
          15       1.00      1.00      1.00        10
          16       1.00      1.00      1.00         7
          17       0.97      0.97      0.97        70
          18       1.00      1.00      1.00        31
          19       0.91      1.00      0.95        10
          20       1.00      1.00      1.00        47
          21       1.00      1.00      1.00         8
          22       1.00      1.00      1.00        98
          23       1.00      1.00      1.00        14
          24       0.97      0.94      0.96        36
          25       1.00      1.00      1.00      8317

    accuracy                           1.00      9366
   macro avg       0.99      0.98      0.99      9366
weighted avg       1.00      1.00      1.00      9366

Confusion Matrix:
[[  31    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    2    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    1]
 [   0    0   31    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    9    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0   19    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    3    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0  110    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    3]
 [   0    0    0    0    0    0    0    4    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0  189    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0  255    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    5    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    9    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0   29    0
```

random forest

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Handle missing values
data = data.dropna()

# Encode categorical columns if any
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Define features and target
```

```
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split 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)

# Create and train the Random Forest classifier
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plot the confusion matrix as a box (heatmap)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, square=True)
plt.title("Confusion Matrix - Random Forest")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
Accuracy: 0.9973

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       1.00      0.33      0.50         3
           2       0.91      0.97      0.94        31
           3       1.00      0.89      0.94         9
           4       1.00      1.00      1.00        19
           5       1.00      1.00      1.00         3
           6       0.99      0.98      0.99       113
           7       1.00      0.75      0.86         4
           8       1.00      0.97      0.99       189
           9       0.97      1.00      0.98       255
          10       1.00      1.00      1.00         5
          11       1.00      1.00      1.00         9
          13       0.97      0.97      0.97        29
          14       1.00      1.00      1.00        18
          15       1.00      1.00      1.00        10
          16       1.00      1.00      1.00         7
          17       1.00      0.96      0.98        70
          18       1.00      1.00      1.00        31
          19       1.00      1.00      1.00        10
          20       0.98      1.00      0.99        47
          21       1.00      1.00      1.00         8
          22       1.00      0.98      0.99        98
          23       1.00      0.79      0.88        14
          24       1.00      0.92      0.96        36
          25       1.00      1.00      1.00      8317

    accuracy                           1.00      9366
   macro avg       0.99      0.94      0.96      9366
weighted avg       1.00      1.00      1.00      9366

Confusion Matrix:
[[  31    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    1    1    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    1]
 [   0    0   30    0    0    0    0    0    0    1    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    8    0    0    1    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0   19    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    3    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0  111    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    2]
 [   0    0    0    0    0    0    0    3    0    0    0    0    1    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0  184    5    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0  254    0    0    0    0
     0    0    0    0    0    0    0    0    0    1]
 [   0    0    0    0    0    0    0    0    0    0    5    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    9    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0]
```

xg booster

```
pip install xgboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.12/dist-packages (3.0.4)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.0.2)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.12/dist-packages (from xgboost) (2.27.3)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from xgboost) (1.16.1)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Handle missing values
data = data.dropna()

# Encode categorical columns if any
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
```

```
        data[column] = le.fit_transform(data[column])
        label_encoders[column] = le

# Define features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the XGBoost classifier
model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plot the confusion matrix as a box (heatmap)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, square=True)
plt.title("Confusion Matrix - XGBoost")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

```
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning: [18:12:49] WARNING: /workspace/src/learner.c
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)
Accuracy: 0.9989

Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        31
           1       1.00      0.67      0.80         3
           2       1.00      1.00      1.00        31
           3       1.00      1.00      1.00         9
           4       1.00      1.00      1.00        19
           5       1.00      1.00      1.00         3
           6       1.00      1.00      1.00       113
           7       1.00      0.50      0.67         4
           8       0.99      0.98      0.99       189
           9       0.99      1.00      1.00       255
          10       1.00      1.00      1.00         5
          11       0.90      1.00      0.95         9
          13       1.00      0.97      0.98        29
          14       1.00      0.94      0.97        18
          15       1.00      1.00      1.00        10
          16       1.00      1.00      1.00         7
          17       1.00      0.99      0.99        70
          18       0.97      1.00      0.98        31
          19       1.00      1.00      1.00        10
          20       1.00      1.00      1.00        47
          21       1.00      1.00      1.00         8
          22       1.00      1.00      1.00        98
          23       1.00      1.00      1.00        14
          24       1.00      0.97      0.99        36
          25       1.00      1.00      1.00      8317

    accuracy                           1.00      9366
   macro avg       0.99      0.96      0.97      9366
weighted avg       1.00      1.00      1.00      9366

Confusion Matrix:
[[  31    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    2    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    1]
 [   0    0   31    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    9    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0   19    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    3    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0  113    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    2    1    0    0    1    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0  186    2    0    0    0    0
     0    0    0    1    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0  255    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    1    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    0]
 [   0    0    0    0    0    0    0    0    0    0    0    0    0    0
     0    0    0    0    0    0    0    0    0    0    1]
```

lightGBM

```
pip install lightgbm
```

```
Requirement already satisfied: lightgbm in /usr/local/lib/python3.12/dist-packages (4.6.0)
Requirement already satisfied: numpy>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from lightgbm) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from lightgbm) (1.16.1)
```

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.preprocessing import LabelEncoder

# Load the dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Handle missing values
data = data.dropna()

# Encode categorical columns if any
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
```

```
        label_encoders[column] = le

# Define features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the LightGBM classifier
model = LGBMClassifier(random_state=42)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}\n")
print("Classification Report:")
print(classification_report(y_test, y_pred))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(cm)

# Plot the confusion matrix as a box (heatmap)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, square=True)
plt.title("Confusion Matrix - LightGBM")
plt.xlabel("Predicted Labels")
plt.ylabel("True Labels")
plt.show()
```

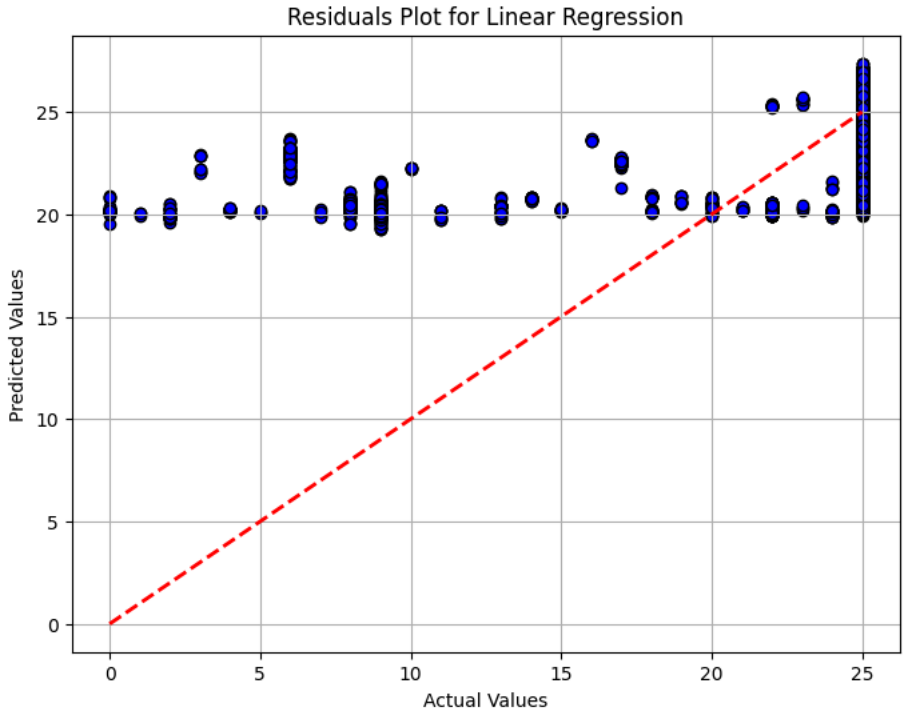
```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.005094 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 1468
[LightGBM] [Info] Number of data points in the train set: 37462, number of used features: 7
[LightGBM] [Info] Start training from score -5.821552
[LightGBM] [Info] Start training from score -8.739323
[LightGBM] [Info] Start training from score -5.686895
[LightGBM] [Info] Start training from score -6.893496
[LightGBM] [Info] Start training from score -6.149056
[LightGBM] [Info] Start training from score -7.697869
[LightGBM] [Info] Start training from score -4.451149
[LightGBM] [Info] Start training from score -7.353029
[LightGBM] [Info] Start training from score -3.774150
[LightGBM] [Info] Start training from score -3.608438
[LightGBM] [Info] Start training from score -7.235245
[LightGBM] [Info] Start training from score -6.599257
[LightGBM] [Info] Start training from score -9.837935
[LightGBM] [Info] Start training from score -5.694800
[LightGBM] [Info] Start training from score -6.599257
[LightGBM] [Info] Start training from score -6.975734
[LightGBM] [Info] Start training from score -7.198878
[LightGBM] [Info] Start training from score -5.128405
[LightGBM] [Info] Start training from score -5.777492
[LightGBM] [Info] Start training from score -7.395588
[LightGBM] [Info] Start training from score -5.258083
[LightGBM] [Info] Start training from score -6.659881
[LightGBM] [Info] Start training from score -4.639438
[LightGBM] [Info] Start training from score -6.659881
[LightGBM] [Info] Start training from score -5.596608
[LightGBM] [Info] Start training from score -0.118551
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

linear regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score#, confusion_matrix # Removed confusion_matrix
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score # Added accuracy_score import

# Load dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Handle missing values
data = data.dropna()

# Encode categorical columns if any
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Define features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
```

```
# Split into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Create and train the Linear Regression model
model = LinearRegression()
model.fit(X_train, y_train)

# Make 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): {mse:.4f}")
print(f"R² Score: {r2:.4f}\n")

# Display model coefficients
print("Model Coefficients:")
for feature, coef in zip(X.columns, model.coef_):
    print(f"{feature}: {coef:.4f}")

print(f"\nIntercept: {model.intercept_:.4f}")

# Plotting Residuals Plot
plt.figure(figsize=(8,6))
plt.scatter(y_test, y_pred, color='blue', edgecolor='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', lw=2)
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Residuals Plot for Linear Regression")
plt.grid(True)
plt.show()

# Convert continuous predictions to discrete classes by rounding
y_test_rounded = np.round(y_test).astype(int)
y_pred_rounded = np.round(y_pred).astype(int)

# # Compute confusion matrix - Removed since it's a regression task
# cm = confusion_matrix(y_test_rounded, y_pred_rounded)

# # Plot the confusion matrix as a box (heatmap) - Removed
# plt.figure(figsize=(8,6))
# sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=True, square=True)
# plt.title("Confusion Matrix (Binned) - Linear Regression")
# plt.xlabel("Predicted Rounded Values")
# plt.ylabel("Actual Rounded Values")
# plt.show()
# Compute and print accuracy
accuracy = accuracy_score(y_test_rounded, y_pred_rounded)
print(f"Accuracy (after rounding): {accuracy*100:.2f}%")
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```




```
pip install interpret matplotlib pillow
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

Explainable Boosting

```
!pip install interpret
```

```
Requirement already satisfied: interpret in /usr/local/lib/python3.12/dist-packages (0.7.2)
Requirement already satisfied: interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2 in /usr/local/lib/python3.12/dist-packages (from interpret) (0.7.2)
Requirement already satisfied: numpy>=1.25 in /usr/local/lib/python3.12/dist-packages (from interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.0.2)
Requirement already satisfied: pandas>=0.19.2 in /usr/local/lib/python3.12/dist-packages (from interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.2.2)
Requirement already satisfied: scipy>=0.18.1 in /usr/local/lib/python3.12/dist-packages (from interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.16.3)
Requirement already satisfied: scikit-learn>=0.18.1 in /usr/local/lib/python3.12/dist-packages (from interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.6.1)
Requirement already satisfied: joblib>=0.10.1 in /usr/local/lib/python3.12/dist-packages (from interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.5.2)
Requirement already satisfied: psutil>=5.6.2 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (5.9.5)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.32.4)
Requirement already satisfied: ipykernel>=4.10.0 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (6.17.1)
Requirement already satisfied: ipython>=5.5.0 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (7.34.0)
Requirement already satisfied: SALib>=1.3.3 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.5.2)
Requirement already satisfied: shap>=0.28.5 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.49.1)
Requirement already satisfied: dill>=0.2.5 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.4.0)
Requirement already satisfied: aplr>=10 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (10.8.0)
Requirement already satisfied: dash<3.0.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.18.2)
Requirement already satisfied: dash-table==5.0.0 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (5.0.0)
Requirement already satisfied: dash-html-components==2.0.0 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.0.0)
Requirement already satisfied: dash-core-components==2.0.0 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.0.0)
Requirement already satisfied: Flask>=1.0.4 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.1.2)
Requirement already satisfied: Werkzeug<3.1 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.0.6)
Requirement already satisfied: plotly>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (5.24.1)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (8.7.0)
Requirement already satisfied: typing-extensions>=4.1.1 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (4.15.0)
Requirement already satisfied: nest-asyncio in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.6.0)
Requirement already satisfied: retrying in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.4.2)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (75.2.0)
Requirement already satisfied: gevent>=1.3.6 in /usr/local/lib/python3.12/dist-packages (from dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (25.9.1)
Requirement already satisfied: greenlet>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from gevent>=1.3.6->dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.2.4)
Requirement already satisfied: zope.event in /usr/local/lib/python3.12/dist-packages (from gevent>=1.3.6->dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (5.1.1)
Requirement already satisfied: zope.interface in /usr/local/lib/python3.12/dist-packages (from gevent>=1.3.6->dash<3.0.0,>=2.0.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (8.0.1)
Requirement already satisfied: debugpy>=1.0 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (1.8.15)
Requirement already satisfied: jupyter-client>=6.1.12 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (7.4.9)
Requirement already satisfied: matplotlib-inline>=0.1 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.2.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (25.0)
Requirement already satisfied: pyzmq>=17 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (26.2.1)
Requirement already satisfied: tornado>=6.1 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (6.5.1)
Requirement already satisfied: traitlets>=5.0.0 in /usr/local/lib/python3.12/dist-packages (from ipykernel>=4.10.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (5.7.1)
Requirement already satisfied: jedi>=0.16 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.19.2)
Requirement already satisfied: decorator in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (4.4.2)
Requirement already satisfied: pickleshare in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.0.52)
Requirement already satisfied: pygments in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.19.2)
Requirement already satisfied: backcall in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.2.0)
Requirement already satisfied: matplotlib-inline in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.2.1)
Requirement already satisfied: pexpect>4.3 in /usr/local/lib/python3.12/dist-packages (from ipython>=5.5.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (4.9.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.19.2->interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.19.2->interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas>=0.19.2->interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2025.2)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.12/dist-packages (from plotly>=3.8.1->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (9.1.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from requests>=2.19.0->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (2025.11.12)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.12/dist-packages (from shap>=0.28.5->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.1.2)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.12/dist-packages (from shap>=0.28.5->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.12/dist-packages (from shap>=0.28.5->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (0.60.0)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.12/dist-packages (from shap>=0.28.5->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (4.67.1)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.18.1->interpret-core==0.7.2->interpret-core[aplr,dash,debug,linear,notebook,plotly,sensitivity,shap,skoperules,treeinterpreter]==0.7.2->interpret) (3.6.0)
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from interpret.glassbox import ExplainableBoostingClassifier
```

```
from interpret import show
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load data
try:
    data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')
except Exception as e:
    print(f"Error loading data: {e}")
    exit()

# Prepare data
X = data.iloc[:, :-1]
y = data.iloc[:, -1]
X = X.drop(['InvoiceNo', 'Description'], axis=1)
X['InvoiceDate'] = pd.to_datetime(X['InvoiceDate'], errors='coerce')
X['InvoiceDate'] = X['InvoiceDate'].astype(np.int64) // 10**9
combined = pd.concat([X, y], axis=1)
combined = combined.dropna(subset=['CustomerID'])
X = combined.drop(y.name, axis=1)
y = combined[y.name]
combined = pd.concat([X, y], axis=1)
combined = combined.dropna()
X = combined.drop(y.name, axis=1)
y = combined[y.name]
for col in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col].astype(str))
if y.dtype == 'object':
    le = LabelEncoder()
    y = le.fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Train model
ebm = ExplainableBoostingClassifier(random_state=42)
ebm.fit(X_train, y_train)
y_pred = ebm.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy*100:.2f}%")

# Explain model globally
ebm_global = ebm.explain_global(name='EBM Global Explanation')

# Show and save the explanation plot as an image
show(ebm_global)

# Since interpret's `show()` opens a browser window and not matplotlib,
# manually save a screenshot or use another plotting library if needed.
# Alternatively, export data and create custom plots with matplotlib if required.
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
/usr/local/lib/python3.12/dist-packages/interpret/glassbox/ebm/ebm.py:1744: UserWarning:
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
For multiclass we cannot currently visualize pairs and they will be stripped from the global explanations. Set interactions=
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
/usr/local/lib/python3.12/dist-packages/interpret/glassbox/ebm/ebm.py:1238: UserWarning:
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix, r2_score
from sklearn.preprocessing import LabelEncoder

# Classifiers
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier

# Regressor
from sklearn.linear_model import LinearRegression

# Load dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')
data = data.dropna()

# Encode categorical columns if any
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])

# Features and target
X = data.iloc[:, :-1]
y = data.iloc[:, -1]

# Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Store results
results = {}
train_scores = {}
test_scores = {}

### ✅ Decision Tree
dt_model = DecisionTreeClassifier(random_state=42)
dt_model.fit(X_train, y_train)
dt_train_pred = dt_model.predict(X_train)
dt_test_pred = dt_model.predict(X_test)
dt_train_acc = accuracy_score(y_train, dt_train_pred)
dt_test_acc = accuracy_score(y_test, dt_test_pred)
results['Decision Tree'] = {'score': dt_test_acc, 'conf_matrix': confusion_matrix(y_test, dt_test_pred)}
train_scores['Decision Tree'] = dt_train_acc
test_scores['Decision Tree'] = dt_test_acc

### ✅ Random Forest
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
rf_train_pred = rf_model.predict(X_train)
rf_test_pred = rf_model.predict(X_test)
rf_train_acc = accuracy_score(y_train, rf_train_pred)
rf_test_acc = accuracy_score(y_test, rf_test_pred)
results['Random Forest'] = {'score': rf_test_acc, 'conf_matrix': confusion_matrix(y_test, rf_test_pred)}
train_scores['Random Forest'] = rf_train_acc
test_scores['Random Forest'] = rf_test_acc

### ✅ XGBoost
xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
xgb_model.fit(X_train, y_train)
xgb_train_pred = xgb_model.predict(X_train)
xgb_test_pred = xgb_model.predict(X_test)
xgb_train_acc = accuracy_score(y_train, xgb_train_pred)
xgb_test_acc = accuracy_score(y_test, xgb_test_pred)
results['XGBoost'] = {'score': xgb_test_acc, 'conf_matrix': confusion_matrix(y_test, xgb_test_pred)}
train_scores['XGBoost'] = xgb_train_acc
test_scores['XGBoost'] = xgb_test_acc

### ✅ LightGBM
lgbm_model = LGBMClassifier(random_state=42)
lgbm_model.fit(X_train, y_train)
lgbm_train_pred = lgbm_model.predict(X_train)
```

```
lgbm_test_pred = lgbm_model.predict(X_test)
lgbm_train_acc = accuracy_score(y_train, lgbm_train_pred)
lgbm_test_acc = accuracy_score(y_test, lgbm_test_pred)
results['LightGBM'] = {'score': lgbm_test_acc, 'conf_matrix': confusion_matrix(y_test, lgbm_test_pred)}
train_scores['LightGBM'] = lgbm_train_acc
test_scores['LightGBM'] = lgbm_test_acc

### ✅ Linear Regression
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)
lr_train_pred = lr_model.predict(X_train)
lr_test_pred = lr_model.predict(X_test)
lr_train_r2 = r2_score(y_train, lr_train_pred)
lr_test_r2 = r2_score(y_test, lr_test_pred)
results['Linear Regression'] = {'score': lr_test_r2, 'conf_matrix': None}
train_scores['Linear Regression'] = lr_train_r2
test_scores['Linear Regression'] = lr_test_r2

### ✅ Plot Accuracy / R² Bar Graph
plt.figure(figsize=(10, 6))
models = list(results.keys())
scores = [results[m]['score'] for m in models]
bar_colors = ['blue', 'blue', 'blue', 'blue', 'green']
plt.bar(models, scores, color=bar_colors)
plt.ylabel('Accuracy / R² Score')
plt.title('Comparison of Model Performance')
plt.xticks(rotation=20)
plt.ylim(0, 1)
for i, v in enumerate(scores):
    plt.text(i, v + 0.02, f"{v:.2f}", ha='center')
plt.tight_layout()
plt.show()

### ✅ Plot Confusion Matrices
for model_name in ['Decision Tree', 'Random Forest', 'XGBoost', 'LightGBM']:
    cm = results[model_name]['conf_matrix']
    plt.figure(figsize=(6, 5))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
    plt.title(f'Confusion Matrix - {model_name}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

### ✅ Plot Training and Testing Curves
plt.figure(figsize=(10, 6))
models = list(train_scores.keys())
train_vals = [train_scores[m] for m in models]
test_vals = [test_scores[m] for m in models]

plt.plot(models, train_vals, marker='o', label='Training Score')
plt.plot(models, test_vals, marker='s', label='Testing Score')
plt.ylabel('Score')
plt.title('Training vs Testing Scores for Each Model')
plt.xticks(rotation=20)
plt.ylim(0, 1)
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
/usr/local/lib/python3.12/dist-packages/xgboost/training.py:183: UserWarning:
[LightGBM] [Info] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[18:29:49] WARNING: /workspace/src/learner.cc:738:
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Parameters: { "use_label_encoder" } are not used.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.002866 seconds.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Total Bins 1468
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Number of data points in the train set: 37462, number of used features: 7
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.821552
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 8.713923
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.686805
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.893406
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.149056
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 7.697869
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 4.451149
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 7.353029
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 3.774150
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 3.608438
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 7.235245
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.599257
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 0.837935
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.694800
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.599257
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.975734
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 7.198878
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.128405
[LightGBM] [Info] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Info] Start training from score 5.777402
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 7.395588
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.258083
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.659881
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 4.639438
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 6.659881
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 5.596608
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Start training from score 0.118551
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Info] Stopped training because there are no more leaves that meet the split requirements
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

lime

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
!pip install lime
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Collecting lime
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
  Downloading lime-0.2.0.1.tar.gz (275 kB)
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 275.7/275.7 kB 8.3 MB/s eta 0:00:00
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
  Preparing metadata (setup.py) ... done
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Requirement already satisfied: matplotlib in /usr/local/lib/python3.12/dist-packages (from lime) (3.10.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.12/dist-packages (from lime) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.12/dist-packages (from lime) (1.16.1)
Requirement already satisfied: tqdm in /usr/local/lib/python3.12/dist-packages (from lime) (4.67.1)
Requirement already satisfied: scikit-learn>=0.18 in /usr/local/lib/python3.12/dist-packages (from lime) (1.6.1)
Requirement already satisfied: scikit-image>=0.12 in /usr/local/lib/python3.12/dist-packages (from lime) (0.25.2)
Requirement already satisfied: networkx>=3.0 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (3.5
Requirement already satisfied: pillow>=10.1 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (11.3
Requirement already satisfied: imageio!=2.35.0,>=2.33 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->l
Requirement already satisfied: tifffile>=2022.8.12 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime
Requirement already satisfied: packaging>=21 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (25.
Requirement already satisfied: lazy-loader>=0.4 in /usr/local/lib/python3.12/dist-packages (from scikit-image>=0.12->lime) (
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.18->lime) (1.5
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from scikit-learn>=0.18->lim
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (1.3.3)
[LightGBM] [Info] Stopped training because there are no more leaves that meet the split requirements
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (4.59.2)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (1.4.9)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from matplotlib->lime) (2.9.
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.7->matplotlib->l
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Building wheels for collected packages: lime
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
  Building wheel for lime (setup.py) ... done
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
```

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
import lime
import lime.lime_tabular

# Load dataset
data = pd.read_csv('/content/data (1).csv', encoding='ISO-8859-1')

# Drop non-numeric columns that are not suitable for the model
data = data.drop(['InvoiceNo', 'Description'], axis=1)

# Handle missing values by dropping rows with any NaN values
data = data.dropna()

# Encode categorical columns if any exist after dropping
label_encoders = {}
for column in data.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    data[column] = le.fit_transform(data[column])
    label_encoders[column] = le

# Define features and target
X = data.iloc[:, :-1].values
y = data.iloc[:, -1].values

# Encode target if it's categorical
if y.dtype == 'object':
    le = LabelEncoder()
    y = le.fit_transform(y)

# Split data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Scale features
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Track accuracy over different numbers of estimators
n_estimators_list = [10, 50, 100, 150, 200, 250, 300]
train_accuracies = []
test_accuracies = []

for n in n_estimators_list:
    model = RandomForestClassifier(n_estimators=n, random_state=42)
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)
    train_acc = accuracy_score(y_train, y_train_pred)
    test_acc = accuracy_score(y_test, y_test_pred)
    train_accuracies.append(train_acc)
    test_accuracies.append(test_acc)

# Plot training and testing accuracy curves
plt.figure(figsize=(10,6))
plt.plot(n_estimators_list, train_accuracies, marker='o', label='Training Accuracy')
plt.plot(n_estimators_list, test_accuracies, marker='s', label='Testing Accuracy')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('Training and Testing Accuracy vs Number of Estimators')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# Train final model for LIME explanation
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict and print final test accuracy
```

```
y_pred = model.predict(X_test)
print(f"Final Test Accuracy: {accuracy_score(y_test, y_pred)*100:.2f}%")

# Create LIME explainer
explainer = lime.lime_tabular.LimeTabularExplainer(
```