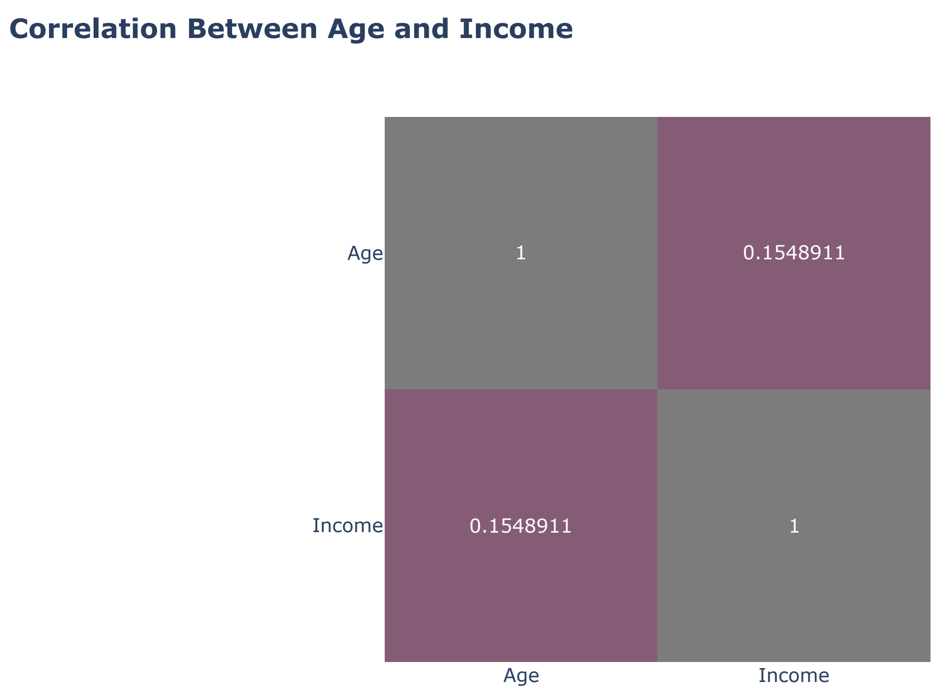
```
# Load necessary libraries
import pandas as pd
import numpy as np
import plotly.express as px
# Load the dataset
data = pd.read_excel('/content/marketing_campaign_final.xlsx')

# Explore and preprocess data (handle missing values, outliers, encoding, etc.)
# Example preprocessing steps:
# data.info()  # Check data types and missing values
# data.describe()  # Summary statistics
# Handle missing values: data.fillna(), data.dropna()
# Encode categorical variables: pd.get_dummies(), LabelEncoder, etc.


from sklearn.preprocessing import LabelEncoder

# Assuming 'df' is your DataFrame and 'column_name' is the column you want to encode

# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Fit and transform the column to convert labels to numeric values
data['Education'] = label_encoder.fit_transform(data['Education'])

# Create a LabelEncoder instance
label_encoder = LabelEncoder()

# Fit and transform the column to convert labels to numeric values
data['Marital_Status'] = label_encoder.fit_transform(data['Marital_Status'])


data.drop(columns=['Dt_Customer'], axis=1, inplace=True)

data.fillna(0, inplace=True)

# Split the data into features (X) and target variable (y)
X = data.drop('Teenhome', axis=1)  # Features
y = data['Teenhome']  # Target variable
print(data.head())
```

```
     ID  Year_Birth  Education  Marital_Status   Income  Kidhome  Teenhome  \
0  5524        1957          2               4  58138.0        0         0
1  2174        1954          2               4  46344.0        1         1
2  4141        1965          2               5  71613.0        0         0
3  6182        1984          2               5  26646.0        1         0
4  5324        1981          4               3  58293.0        1         0

   Recency  MntWines  MntFruits  ...  NumWebVisitsMonth  AcceptedCmp3  \
0       58       635         88  ...                  7             0
1       38        11          1  ...                  5             0
2       26       426         49  ...                  4             0
3       26        11          4  ...                  6             0
4       94       173         43  ...                  5             0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response
0              3         11         1
1              3         11         0
2              3         11         0
```

```
3              3         11         0
4              3         11         0

[5 rows x 28 columns]
```

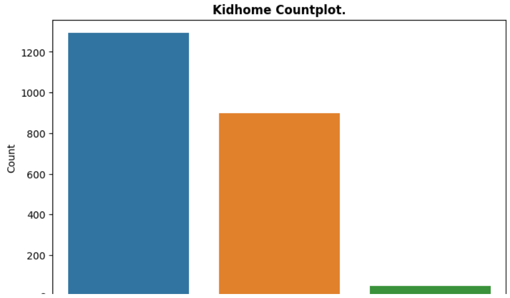
```
data['Age'] = 2023 - data['Year_Birth']
px.imshow(data[["Age", "Income"]].corr(), text_auto=True,
         title="<b>Correlation Between Age and Income",  color_continuous_scale=px.colors.qualitative.Antique)
```

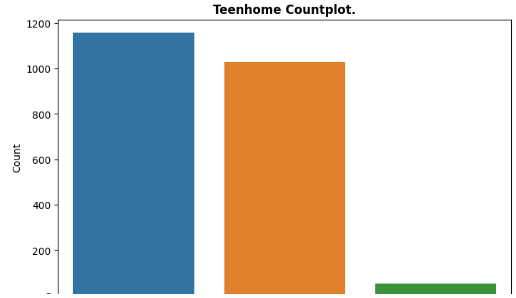



```
## Spending feature creation
data['spending'] = data['MntWines'] + data['MntFruits'] + data['MntMeatProducts'] + data['MntFishProducts'] \
                        + data['MntSweetProducts'] + data['MntGoldProds']
```

```
## rename some cols
data=data.rename(columns={'NumWebPurchases': 'Web','NumCatalogPurchases':'Catalog','NumStorePurchases':'Store'})
```

```
used_cols = ["Kidhome", "Teenhome"]

fig, ax = plt.subplots(1, 2, figsize = (18,5))

for i in range(2):
    sns.countplot(ax=ax[i], data=data, x=used_cols[i])
    ax[i].set_title(f"{used_cols[i]} Countplot.", weight="bold")
    ax[i].set_ylabel("Count")

plt.show()
```





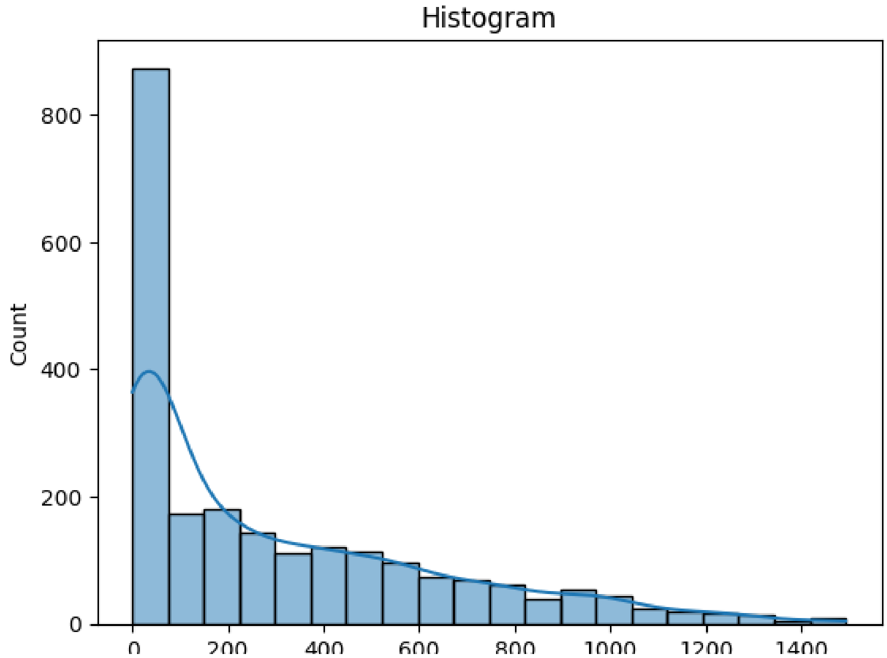

```
## Children feature creation
data['Children'] = data['Kidhome'] + data['Teenhome']

def has_chid_or_no(x):
    if x > 0:
        return "Has child"
    else :
        return "no child"

data['Has_child'] = data["Children"].apply(has_chid_or_no)
data['Has_child'].value_counts()
```

```
Has child    1602
no child      638
Name: Has_child, dtype: int64
```

```
## drop outliers
data = data.loc[(data["Income"] <= 200000)]
## reset index after drop outliers
data.reset_index(drop=True, inplace=True)
print(f'Shape After Drop Outliers : {data.shape}')
print(f'Statistics Info. about Income : \n{data["Income"].describe()}')
```

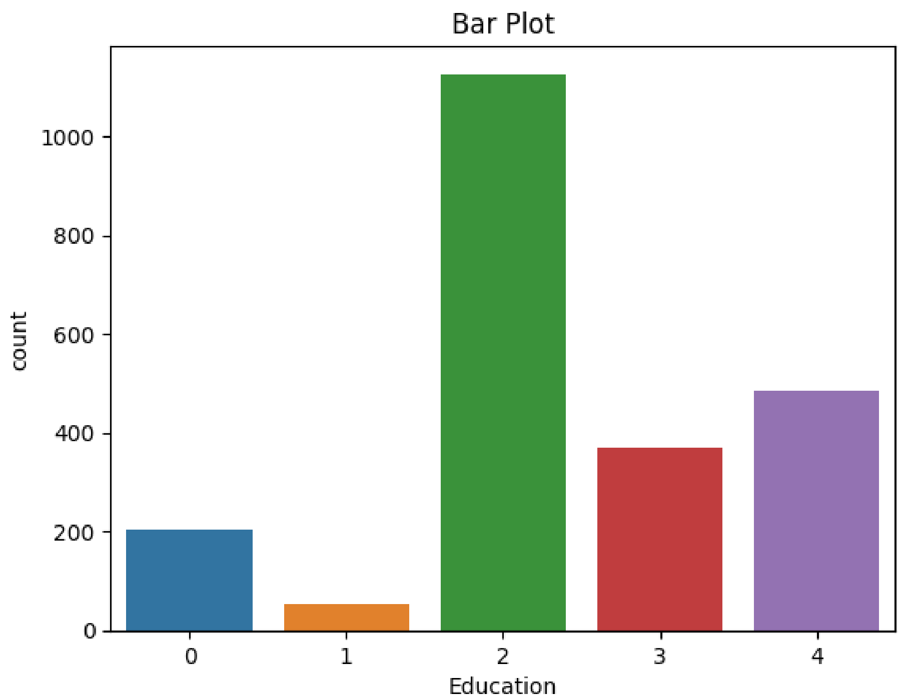
```
Shape After Drop Outliers : (2239, 32)
Statistics Info. about Income :
count      2239.000000
mean      51412.792765
std       22069.582225
min           0.000000
25%       34716.000000
50%       51039.000000
75%       68277.500000
max      162397.000000
Name: Income, dtype: float64
```

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

# Load the dataset
data = pd.read_excel('/content/marketing_campaign_final.xlsx')

# Example visualizations

# Histogram
sns.histplot(data['MntWines'], bins=20, kde=True)
plt.title('Histogram')



plt.show()
```



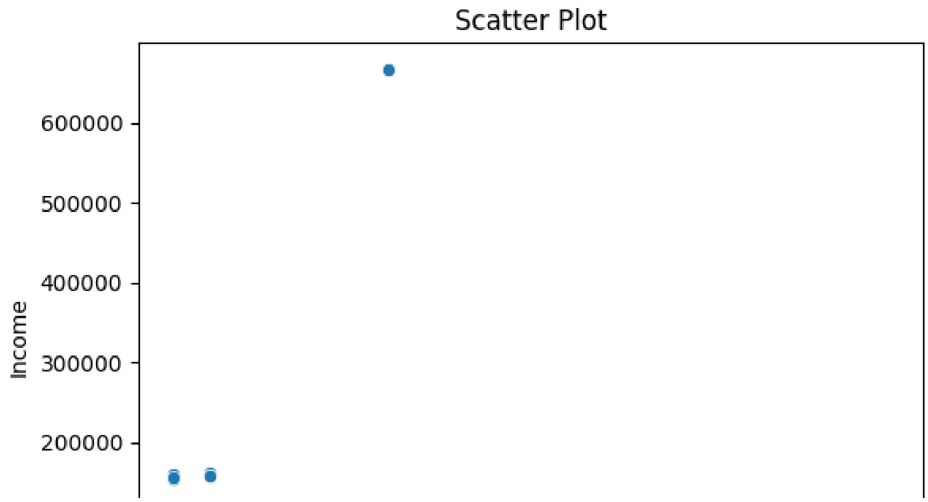

```
# Bar Plot
sns.countplot(x='Education', data=data)
plt.title('Bar Plot')
```

```
Text(0.5, 1.0, 'Bar Plot')
```



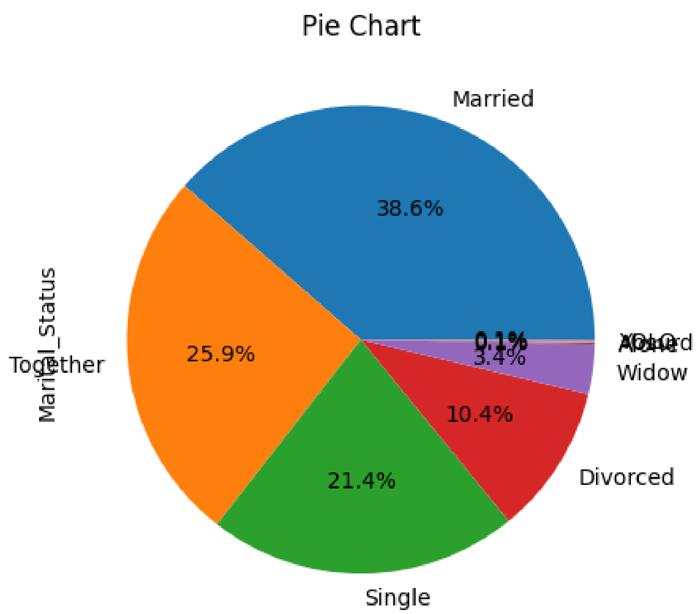

```
# Scatter Plot
sns.scatterplot(x='NumWebVisitsMonth', y='Income', data=data)
plt.title('Scatter Plot')
```

```
Text(0.5, 1.0, 'Scatter Plot')
```



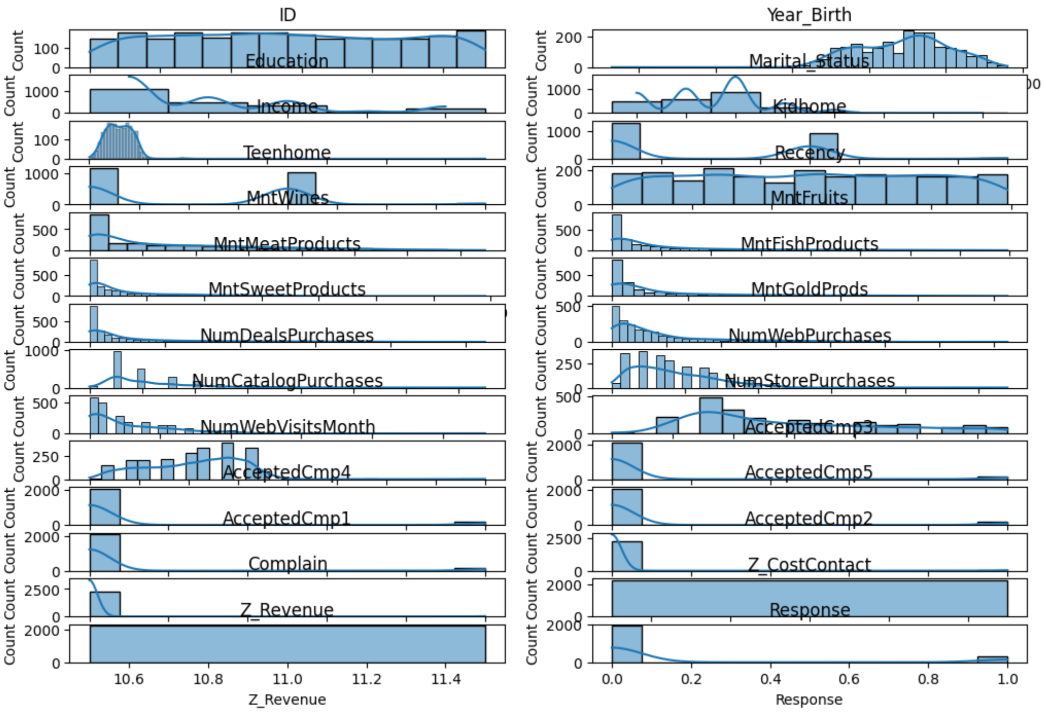

```
# Pie Chart
data['Marital_Status'].value_counts().plot(kind='pie', autopct='%1.1f%%')
plt.title('Pie Chart')
```

```
Text(0.5, 1.0, 'Pie Chart')
```



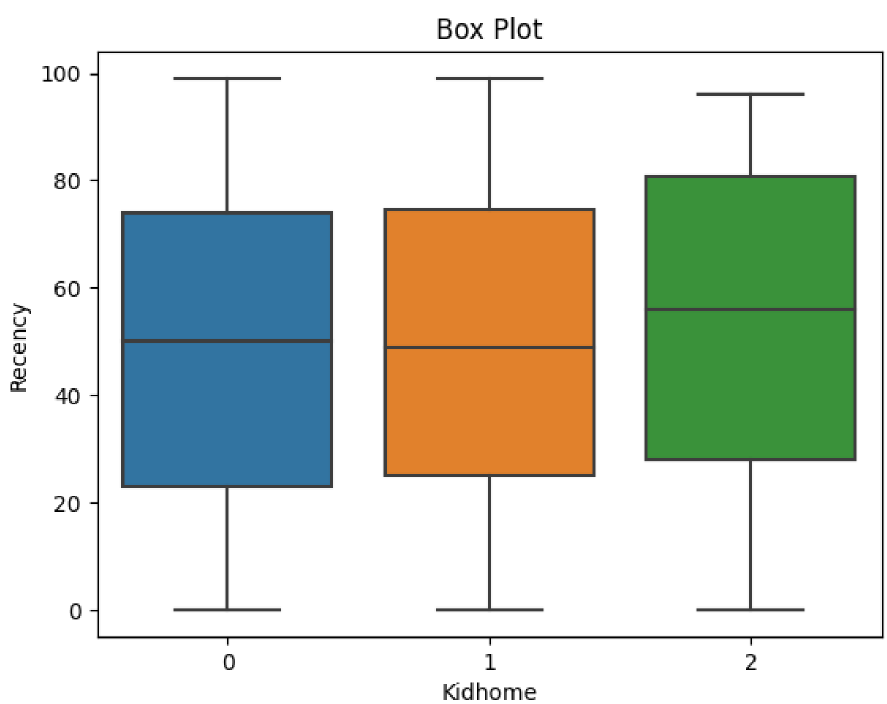

```
# Histograms for each column in the dataset
data.drop('Dt_Customer', axis=1, inplace=True)  # axis=1 indicates column-wise operation

plt.figure(figsize=(12, 8))
num_columns = len(data.columns)
for i, column in enumerate(data.columns):
    plt.subplot((num_columns // 2) + (num_columns % 2), 2, i + 1)
    sns.histplot(data[column], kde=True)
    plt.title(column)
plt.tight_layout()
plt.show()
```

```
<ipython-input-54-27df3b85dbc4>:10: UserWarning: Tight layout not applied. tight_layout
  plt.tight_layout()
```




```
# Box Plot (Interquartile Range)
sns.boxplot(x='Kidhome', y='Recency', data=data)
plt.title('Box Plot')
```

```
Text(0.5, 1.0, 'Box Plot')
```



```
# Check the first few rows of the DataFrame
print(data.head())

# List all columns in the DataFrame
print(data.columns)

# Check the shape of the DataFrame
print(data.shape)
```

```
     ID  Year_Birth   Education Marital_Status   Income  Kidhome  Teenhome  \
0  5524        1957  Graduation         Single  58138.0        0         0
1  2174        1954  Graduation         Single  46344.0        1         1
2  4141        1965  Graduation       Together  71613.0        0         0
3  6182        1984  Graduation       Together  26646.0        1         0
4  5324        1981         PhD        Married  58293.0        1         0

   Recency  MntWines  MntFruits  ...  NumWebVisitsMonth  AcceptedCmp3  \
0       58       635         88  ...                  7             0
1       38        11          1  ...                  5             0
2       26       426         49  ...                  4             0
3       26        11          4  ...                  6             0
4       94       173         43  ...                  5             0

   AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  Complain  \
0             0             0             0             0         0
1             0             0             0             0         0
2             0             0             0             0         0
3             0             0             0             0         0
4             0             0             0             0         0

   Z_CostContact  Z_Revenue  Response
0              3         11         1
1              3         11         0
2              3         11         0
3              3         11         0
4              3         11         0

[5 rows x 28 columns]
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
(2240, 28)
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegressionCV

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)



# Using L1 regularization
logreg = LogisticRegression(max_iter=10000, solver='saga', penalty='l1' , tol=1e-3)
logreg.fit(X_train, y_train)

# Predict on the test set
predictions = logreg.predict(X_test)

# Evaluate the model
accuracy = logreg.score(X_test, y_test)
accuracy_percentage = accuracy * 100
print(f"Accuracy: {accuracy_percentage:.2f}%")
```

```
Accuracy: 86.00%
```

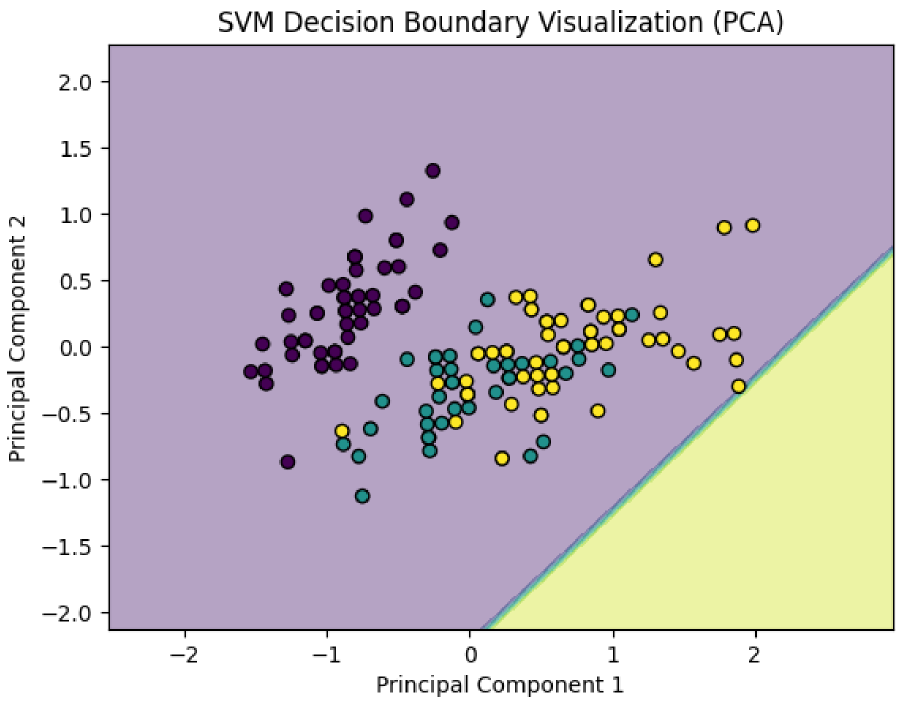
```
from sklearn.linear_model import LogisticRegression
from sklearn.datasets import load_iris  # Using Iris dataset as an example

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Target

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

# Initialize the Logistic Regression model
logreg = LogisticRegression(max_iter=1000 , solver='saga', penalty='l1' , tol=1e-3)  # You can specify other paramete

# Fit the model on the training data
logreg.fit(X_train, y_train)

# Predict on the test set
predictions = logreg.predict(X_test)

# Evaluate the model
accuracy = logreg.score(X_test, y_test)
print(f"Accuracy: {accuracy:.2f}")
```

```
Accuracy: 1.00
```

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.datasets import load_iris  # Using Iris dataset as an example

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Target

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

# Initialize the Support Vector Classifier
svm = SVC(kernel='rbf')  # You can specify other parameters like kernel, C, gamma, etc.

# Fit the model on the training data
svm.fit(X_train, y_train)

# Predict on the test set
predictions = svm.predict(X_test)

# Evaluate the model
accuracy = svm.score(X_test, y_test)
accuracy_percentage = accuracy * 100
print(f"Accuracy: {accuracy_percentage:.2f}%")
```

```
Accuracy: 100.00%
```

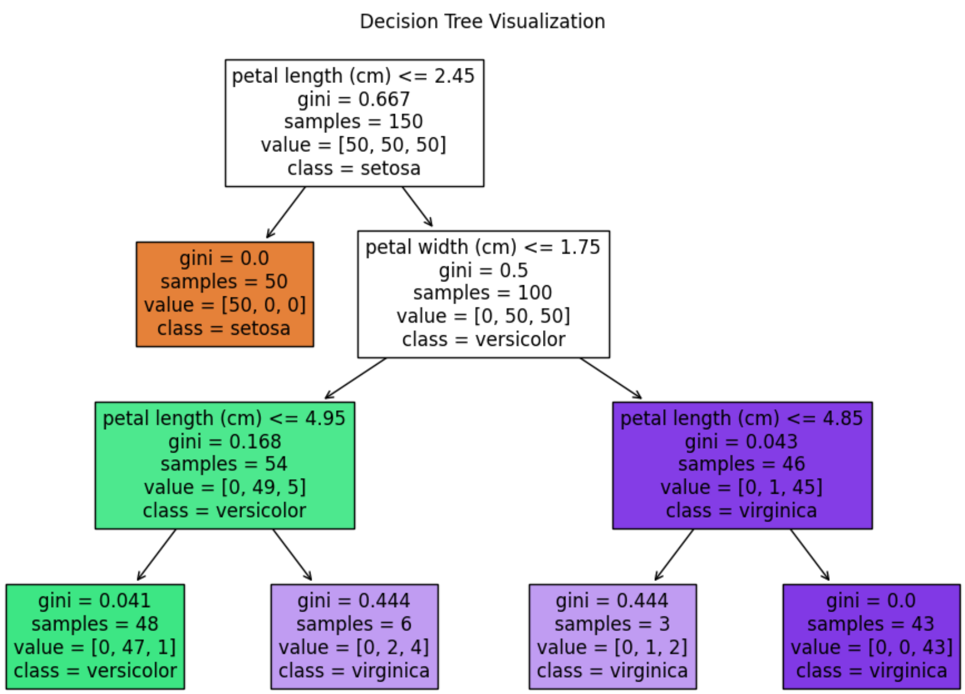
```
from sklearn.decomposition import PCA
from sklearn.svm import SVC

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data[:, :2]  # Considering only the first two features for visualization purposes
y = data.target

# Initialize the SVM classifier
svm = SVC(kernel='linear')  # You can use different kernels

# Fit the SVM on the data
svm.fit(X, y)

# Use PCA to reduce dimensions for visualization (for plotting purposes)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

# Create a meshgrid to plot decision boundary
x_min, x_max = X_pca[:, 0].min() - 1, X_pca[:, 0].max() + 1
y_min, y_max = X_pca[:, 1].min() - 1, X_pca[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, 0.1),
                     np.arange(y_min, y_max, 0.1))

# Get predictions for meshgrid points
Z = svm.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

# Plot decision boundary
plt.contourf(xx, yy, Z, alpha=0.4)
plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y, edgecolor='k')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('SVM Decision Boundary Visualization (PCA)')
plt.show()
```



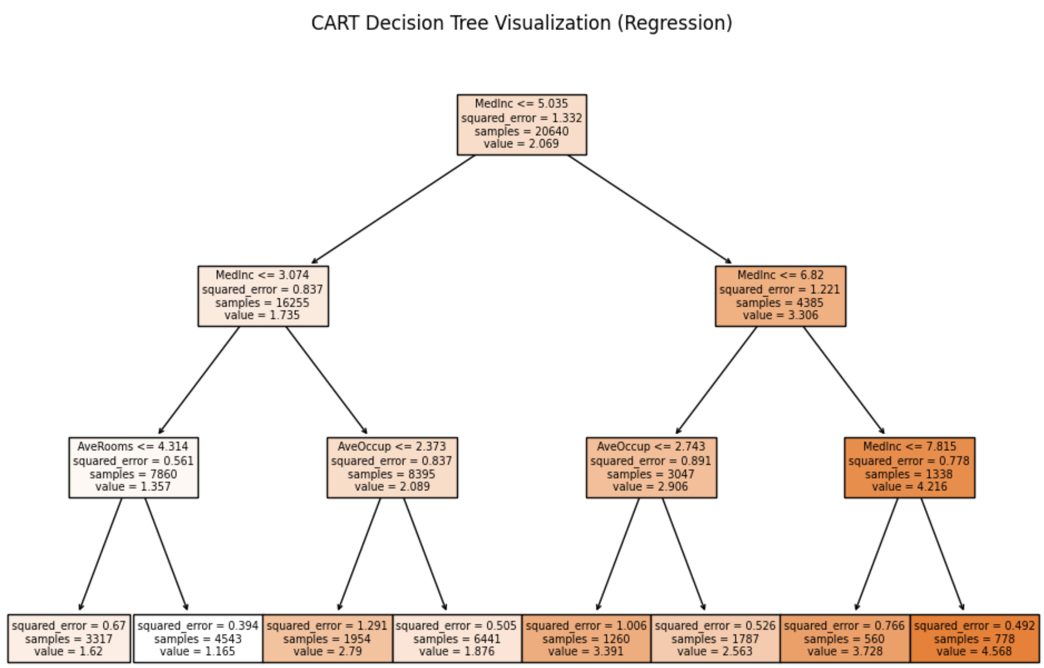

```
from sklearn.tree import DecisionTreeClassifier
 # Using Iris dataset as an example

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Target

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

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)  # You can specify other parameters like max_depth, criter

# Fit the model on the training data
clf.fit(X_train, y_train)

# Predict on the test set
predictions = clf.predict(X_test)

# Evaluate the model
accuracy = clf.score(X_test, y_test)
accuracy_percentage=accuracy*100
print(f"Accuracy: {accuracy_percentage:.2f}")
```

```
Accuracy: 100.00
```

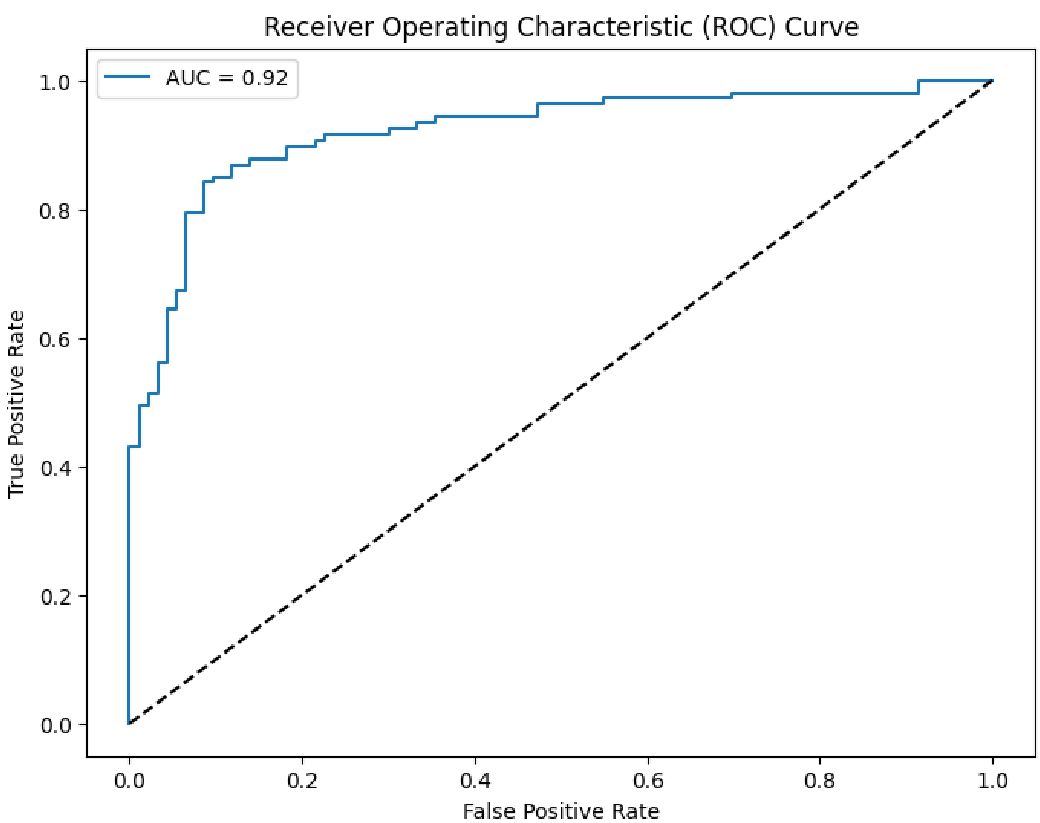
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Target

# Initialize the Decision Tree Classifier
clf = DecisionTreeClassifier(max_depth=3, random_state=42)  # You can specify other parameters like max_depth, criter

# Fit the model on the entire dataset (for demonstration purposes)
clf.fit(X, y)

# Plot the Decision Tree
plt.figure(figsize=(12, 8))
plot_tree(clf, filled=True, feature_names=data.feature_names, class_names=data.target_names)
plt.title("Decision Tree Visualization")
plt.show()
```



```
from sklearn.datasets import fetch_california_housing
from sklearn.tree import DecisionTreeRegressor, plot_tree
import matplotlib.pyplot as plt

# Load the California housing dataset
data = fetch_california_housing()
X = data.data  # Features
y = data.target  # Target (continuous variable for regression)

# Initialize the CART Regressor
regressor = DecisionTreeRegressor(max_depth=3, random_state=42)  # You can specify other parameters like max_depth, c

# Fit the model on the entire dataset (for demonstration purposes)
regressor.fit(X, y)

# Plot the CART Decision Tree for regression
plt.figure(figsize=(12, 8))
plot_tree(regressor, filled=True, feature_names=data.feature_names)
plt.title("CART Decision Tree Visualization (Regression)")
plt.show()
```



```
from sklearn.datasets import make_classification
from sklearn.metrics import roc_curve, roc_auc_score

# Generate synthetic data for demonstration purposes
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

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

# Initialize the Logistic Regression model
model = LogisticRegression()

# Fit the model on the training data
model.fit(X_train, y_train)

# Predict probabilities for the test set
y_prob = model.predict_proba(X_test)[:, 1]

# Calculate ROC curve and AUC score
fpr, tpr, thresholds = roc_curve(y_test, y_prob)
auc = roc_auc_score(y_test, y_prob)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, label=f'AUC = {auc:.2f}')
plt.plot([0, 1], [0, 1], 'k--')  # Diagonal line representing random guessing
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend()
plt.show()
```



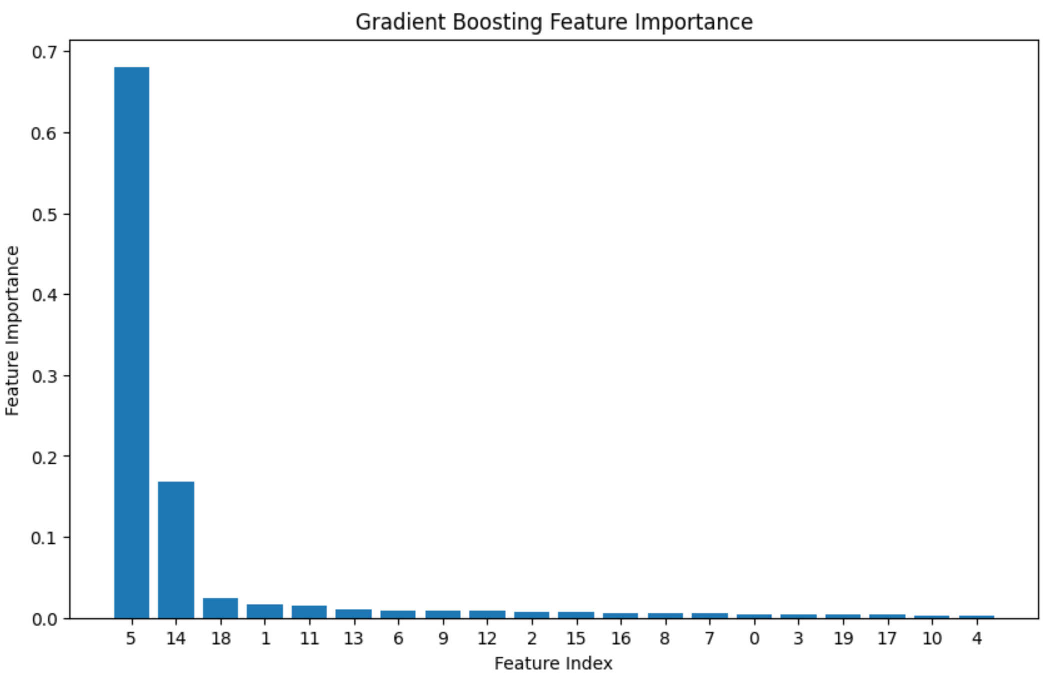
```
# Generate synthetic data for demonstration purposes
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

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

# Initialize the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)  # You can specify other parameters like max_depth, et

# Fit the model on the training data
rf.fit(X_train, y_train)

# Predict on the test set
predictions = rf.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)
accuracy_percentage = accuracy*100
print(f"Random Forest Accuracy: {accuracy_percentage:.2f}%")
```

```
Random Forest Accuracy: 90.00%
```

```
from sklearn.datasets import make_classification
from sklearn.ensemble import GradientBoostingClassifier

# Generate synthetic data for demonstration purposes
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

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

# Initialize the Gradient Boosting Classifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)  # You can specify other parame

# Fit the model on the training data
gb.fit(X_train, y_train)

# Predict on the test set
predictions = gb.predict(X_test)

# Evaluate accuracy
accuracy = accuracy_score(y_test, predictions)
accuracy_percentage = accuracy*100;
print(f"Gradient Boosting Accuracy: {accuracy_percentage:.2f}%")
```

```
Gradient Boosting Accuracy: 91.00%
```

```
# Generate synthetic data for demonstration purposes
X, y = make_classification(n_samples=1000, n_features=20, n_classes=2, random_state=42)

# Initialize the Gradient Boosting Classifier
gb = GradientBoostingClassifier(n_estimators=100, learning_rate=0.1, random_state=42)
gb.fit(X, y)

# Get feature importances
feature_importance = gb.feature_importances_

# Sort feature importances in descending order
indices = feature_importance.argsort()[::-1]

# Plot feature importances
plt.figure(figsize=(10, 6))
plt.title("Gradient Boosting Feature Importance")
plt.bar(range(X.shape[1]), feature_importance[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlabel("Feature Index")
plt.ylabel("Feature Importance")
plt.show()
```



```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.datasets import load_iris  # Using Iris dataset as an example

# Load the dataset (replace this with your dataset)
data = load_iris()
X = data.data  # Features
y = data.target  # Target

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

# Initialize the Random Forest Classifier
rf = RandomForestClassifier(n_estimators=100, random_state=42)  # You can specify other parameters like n_estimators,

# Fit the model on the training data
rf.fit(X_train, y_train)

# Predict on the test set
predictions = rf.predict(X_test)

# Evaluate the model
accuracy = rf.score(X_test, y_test)
accuracy_percentage = accuracy * 100
print(f"Accuracy: {accuracy_percentage:.2f}%")
```

```
Accuracy: 100.00%
```

```
# Create NumPy arrays from the columns
recency = np.array(data['Recency'])
kidhome = np.array(data['Kidhome'])

# Compute Pearson correlation coefficient using NumPy
correlation_matrix = np.corrcoef(recency, kidhome)

# Extract the correlation coefficient
correlation = correlation_matrix[0, 1]

print(f"Pearson Correlation for recency and kidhome: {correlation}")
```

```
Pearson Correlation for recency and kidhome: -1.2819751242557092e-17
```

```
print(data.columns)
```

```
Index(['ID', 'Year_Birth', 'Education', 'Marital_Status', 'Income', 'Kidhome',
       'Teenhome', 'Recency', 'MntWines', 'MntFruits', 'MntMeatProducts',
       'MntFishProducts', 'MntSweetProducts', 'MntGoldProds',
       'NumDealsPurchases', 'NumWebPurchases', 'NumCatalogPurchases',
       'NumStorePurchases', 'NumWebVisitsMonth', 'AcceptedCmp3',
       'AcceptedCmp4', 'AcceptedCmp5', 'AcceptedCmp1', 'AcceptedCmp2',
       'Complain', 'Z_CostContact', 'Z_Revenue', 'Response'],
      dtype='object')
```

```
# Basic filter method: Filtering rows based on a condition
# For example, filtering rows where 'Column_A' values are greater than 50
filtered_df = data[data['Recency'] > 98]

# Display the filtered DataFrame
print("\nFiltered DataFrame:")
print(filtered_df)
```

```
Filtered DataFrame:
```

```
         ID  Year_Birth   Education Marital_Status   Income  Kidhome  \
38     8595        1973  Graduation          Widow  42429.0        0
192    7829        1900    2n Cycle       Divorced  36640.0        1
208     868        1966  Graduation        Married  44794.0        0
444    2106        1974    2n Cycle        Married  20130.0        0
491      22        1976  Graduation       Divorced  46310.0        1
606    7232        1973  Graduation          Widow  42429.0        0
685   10142        1976         PhD       Divorced  66476.0        0
700    9977        1973  Graduation       Divorced  78901.0        0
725    7212        1966  Graduation        Married  44794.0        0
1033   5263        1977    2n Cycle        Married  31056.0        1
1171   3363        1974    2n Cycle        Married  20130.0        0
1473   4070        1969         PhD        Married  94871.0        0
1542    528        1978  Graduation        Married  65819.0        0
1685   7947        1969  Graduation        Married  42231.0        1
1800   2831        1976  Graduation       Together  78416.0        0
1820   2415        1962  Graduation       Together  62568.0        0
1894   1743        1974  Graduation         Single  69719.0        0

      Teenhome  Recency  MntWines  MntFruits  ...  NumWebVisitsMonth  \
38           1       99        55          0  ...                  5
192          0       99        15          6  ...                  5
208          1       99        54          0  ...                  6
444          0       99         0          6  ...                  8
491          0       99       185          2  ...                  8
606          1       99        55          0  ...                  5
685          1       99       372         18  ...                  4
700          1       99       321         11  ...                  4
725          1       99        54          0  ...                  6
1033         0       99         5         10  ...                  8
1171         0       99         0          6  ...                  8
1473         2       99       169         24  ...                  7
1542         0       99       267         38  ...                  3
1685         1       99        24          0  ...                  5
1800         1       99       453         38  ...                  3
1820         1       99       362         17  ...                  4
1894         0       99       273         86  ...                  1

      AcceptedCmp3  AcceptedCmp4  AcceptedCmp5  AcceptedCmp1  AcceptedCmp2  \
38               0             0             0             0             0
192              0             0             0             0             0
208              0             0             0             0             0
444              0             0             0             0             0
491              0             0             0             0             0
606              0             0             0             0             0
685              0             0             0             0             0
700              0             0             0             0             0
725              0             0             0             0             0
1033             0             0             0             0             0
1171             0             0             0             0             0
1473             0             1             1             0             0
1542             0             0             0             0             0
1685             0             0             0             0             0
1800             0             0             0             0             0
1820             0             0             0             1             0
1894             0             0             0             0             0
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix

# Assuming 'X' contains your features and 'y' contains the target variable
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.