credit-score

July 17, 2024

[ ]:
```
!pip install -q autoviz
!pip install -q -U --pre pycaret
```

```
                                        67.5/67.5 kB
1.1 MB/s eta 0:00:00
                                        431.4/431.4
kB 8.4 MB/s eta 0:00:00
                                        2.0/2.0 MB
29.4 MB/s eta 0:00:00
                                        255.9/255.9
MB 4.2 MB/s eta 0:00:00
                                        155.4/155.4
kB 15.3 MB/s eta 0:00:00
                                        24.7/24.7 MB
33.2 MB/s eta 0:00:00
                                        8.3/8.3 MB
42.6 MB/s eta 0:00:00
                                        7.0/7.0 MB
56.2 MB/s eta 0:00:00
                                        486.1/486.1
kB 8.3 MB/s eta 0:00:00
                                        302.2/302.2
kB 26.6 MB/s eta 0:00:00
                                        13.4/13.4 MB
42.7 MB/s eta 0:00:00
                                        163.8/163.8
kB 14.6 MB/s eta 0:00:00
  Preparing metadata (setup.py) … done
                                        258.3/258.3
kB 22.4 MB/s eta 0:00:00
                                        81.9/81.9 kB
6.7 MB/s eta 0:00:00
```

```
                               194.1/194.1
kB 12.9 MB/s eta 0:00:00
                               11.6/11.6 MB
85.1 MB/s eta 0:00:00
                               79.9/79.9 MB
9.7 MB/s eta 0:00:00
                               106.8/106.8
kB 11.0 MB/s eta 0:00:00
                               80.7/80.7 kB
8.5 MB/s eta 0:00:00
                               21.8/21.8 MB
56.9 MB/s eta 0:00:00
                               44.0/44.0 kB
4.5 MB/s eta 0:00:00
                               2.1/2.1 MB
69.9 MB/s eta 0:00:00
                               130.1/130.1
kB 12.6 MB/s eta 0:00:00
                               12.1/12.1 MB
89.1 MB/s eta 0:00:00
                               1.6/1.6 MB
72.4 MB/s eta 0:00:00
                               7.5/7.5 MB
98.0 MB/s eta 0:00:00
                               141.1/141.1
kB 13.0 MB/s eta 0:00:00
                               2.1/2.1 MB
81.5 MB/s eta 0:00:00
  Building wheel for pyod (setup.py) … done
```

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

from pycaret import classification
from sklearn.model_selection import cross_val_score
from sklearn.metrics import classification_report, confusion_matrix
```

[17]:
```
df = pd.read_csv('Credit Score Classification Dataset.csv')
```

[18]:
```
df.head()
```

[18]:
```
   Age  Gender  Income       Education        Marital Status  \
0   25  Female   50000   Bachelor's Degree          Single
```

```
1  30      Male  100000      Master's Degree     Married
2  35    Female   75000            Doctorate     Married
3  40      Male  125000  High School Diploma      Single
4  45    Female  100000    Bachelor's Degree     Married

   Number of Children Home Ownership Credit Score
0                   0         Rented         High
1                   2          Owned         High
2                   1          Owned         High
3                   0          Owned         High
4                   3          Owned         High
```

[19]:
```
df.tail()
```

[19]:
```
     Age  Gender  Income            Education Marital Status  \
159   29  Female   27500  High School Diploma         Single
160   34    Male   47500   Associate's Degree         Single
161   39  Female   62500    Bachelor's Degree        Married
162   44    Male   87500      Master's Degree         Single
163   49  Female   77500            Doctorate        Married

     Number of Children Home Ownership Credit Score
159                   0         Rented          Low
160                   0         Rented      Average
161                   2          Owned         High
162                   0          Owned         High
163                   1          Owned         High
```

1 1. Unique values

[20]:
```
df.nunique()
```

[20]:
```
Age                   29
Gender                 2
Income                52
Education              5
Marital Status         2
Number of Children     4
Home Ownership         2
Credit Score           3
dtype: int64
```

2 2. Data Types

[21]:
```
df.dtypes
```

[21]:
```
Age                    int64
Gender                object
Income                 int64
Education             object
Marital Status        object
Number of Children     int64
Home Ownership        object
Credit Score          object
dtype: object
```

[22]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 164 entries, 0 to 163
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   Age                 164 non-null    int64
 1   Gender              164 non-null    object
 2   Income              164 non-null    int64
 3   Education           164 non-null    object
 4   Marital Status      164 non-null    object
 5   Number of Children  164 non-null    int64
 6   Home Ownership      164 non-null    object
 7   Credit Score        164 non-null    object
dtypes: int64(3), object(5)
memory usage: 10.4+ KB
```

3 3. Statistics

[23]:
```
df.describe().T
```

[23]:
```
                    count          mean           std      min       25%  \
Age                 164.0     37.975610      8.477289     25.0     30.75
Income              164.0  83765.243902  32457.306728  25000.0  57500.00
Number of Children  164.0      0.652439      0.883346      0.0      0.00

                        50%       75%       max
Age                    37.0      45.0      53.0
Income              83750.0  105000.0  162500.0
Number of Children      0.0       1.0       3.0
```

4 4. Null values

[24]:
```
df.isna().any()
```

[24]:
```
Age                    False
Gender                 False
Income                 False
Education              False
Marital Status         False
Number of Children     False
Home Ownership         False
Credit Score           False
dtype: bool
```

[25]:
```
df.isna().sum()
```

[25]:
```
Age                    0
Gender                 0
Income                 0
Education              0
Marital Status         0
Number of Children     0
Home Ownership         0
Credit Score           0
dtype: int64
```

5 5. Categorical values

[26]:
```
cat_cols = df.select_dtypes(include = ['object']).columns.tolist()
cat_cols
```

[26]:
```
['Gender', 'Education', 'Marital Status', 'Home Ownership', 'Credit Score']
```

6 Label encoding Categorical values

[27]:
```
from sklearn.preprocessing import LabelEncoder

# Create a label encoder object
le = LabelEncoder()

for i in cat_cols:
    # Fit the label encoder object to the dataset
    le.fit(df[i])

    # Transform the dataset using the label encoder object
```

```
    df[i] = le.transform(df[i])
# get list of categorical columns
cat_cols = df.select_dtypes(include=['object']).columns.tolist()
cat_cols
```

[27]: []

7 6. Correlation

[28]:
```
df.corr()
```

[28]:

	Age	Gender	Income	Education	Marital Status
Age	1.000000	0.235343	0.699464	0.170254	-0.517723
Gender	0.235343	1.000000	0.495738	0.248671	0.278362
Income	0.699464	0.495738	1.000000	0.369449	-0.471004
Education	0.170254	0.248671	0.369449	1.000000	-0.067797
Marital Status	-0.517723	0.278362	-0.471004	-0.067797	1.000000
Number of Children	0.055390	-0.442139	0.084547	0.047311	-0.696984
Home Ownership	-0.713803	-0.031519	-0.704928	-0.397043	0.708374
Credit Score	0.205362	-0.247729	0.083698	0.334424	-0.205756

	Number of Children	Home Ownership	Credit Score
Age	0.055390	-0.713803	0.205362
Gender	-0.442139	-0.031519	-0.247729
Income	0.084547	-0.704928	0.083698
Education	0.047311	-0.397043	0.334424
Marital Status	-0.696984	0.708374	-0.205756
Number of Children	1.000000	-0.497129	0.136517
Home Ownership	-0.497129	1.000000	-0.293384
Credit Score	0.136517	-0.293384	1.000000

[29]:
```
# Set the size of figure to 12 by 10.
plt.figure(figsize=(18,10))

sns.heatmap(df.corr(), annot = True)
```

[29]: <Axes: >

8 7. Class Distributions

[30]:
```
# Count the number of instances in each class
class_counts = df['Credit Score'].value_counts()

# Print the class distribution
print('Class distribution:')
print(class_counts)
```

```
Class distribution:
Credit Score
1    113
0     36
2     15
Name: count, dtype: int64
```

9 SMOTE

[31]:
```
X = df.drop('Credit Score', axis = 1)
y = df['Credit Score']
from sklearn.model_selection import train_test_split

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
from imblearn.over_sampling import SMOTE
```

```
# Instantiate SMOTE
sm = SMOTE(random_state=42)

# Fit SMOTE to training data
X_train_res, y_train_res = sm.fit_resample(X_train, y_train)

# Print class distribution of original and resampled data
print('Class distribution before resampling:', y_train.value_counts())
print('Class distribution after resampling:', y_train_res.value_counts())
```

```
Class distribution before resampling: Credit Score
1    90
0    31
2    10
Name: count, dtype: int64
Class distribution after resampling: Credit Score
1    90
0    90
2    90
Name: count, dtype: int64
```

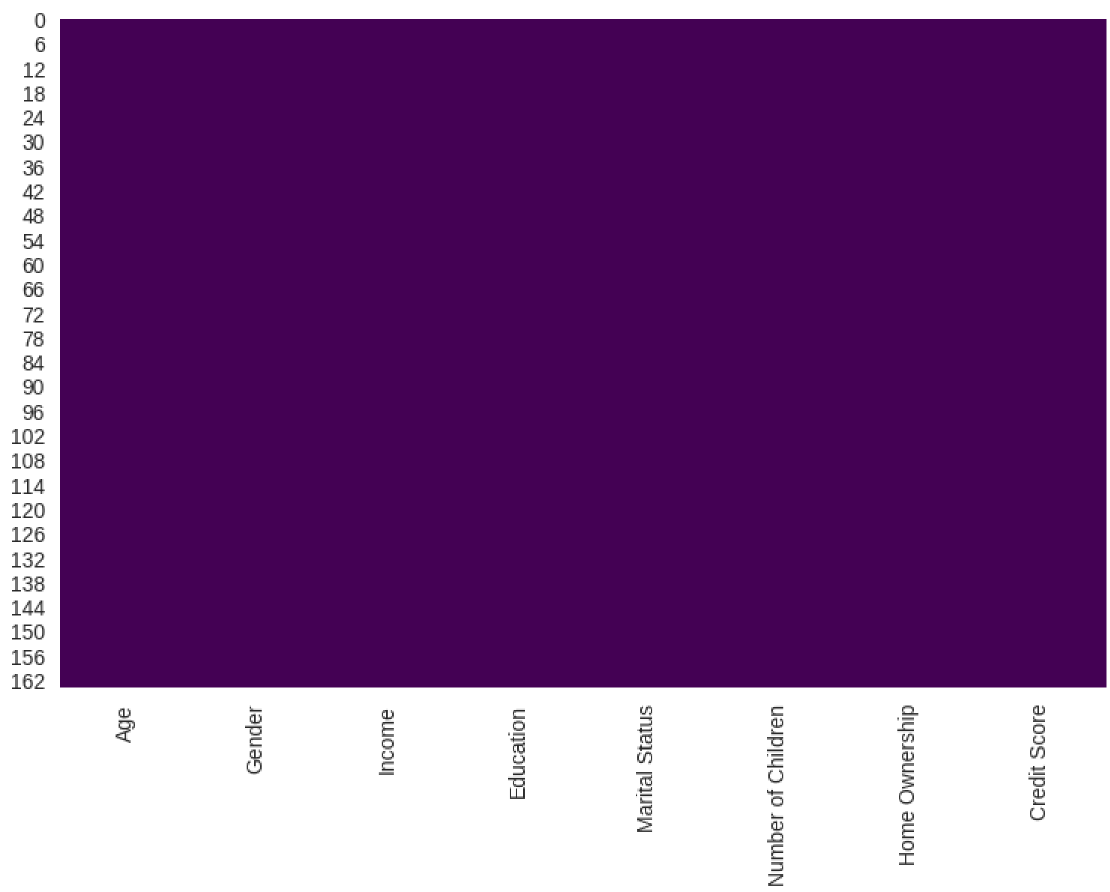
10 Other EDA Suggestions

[34]:
```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
def check_missing_values(df):
    missing_data = df.isnull().sum()
    missing_data = missing_data[missing_data > 0]
    missing_data.sort_values(ascending=False, inplace=True)
    print("Missing Values in Each Column:\n", missing_data)

    # Visualize missing data
    sns.heatmap(df.isnull(), cbar=False, cmap='viridis')
    plt.show()

check_missing_values(df)
def handle_missing_values(df):
    imputer = SimpleImputer(strategy='mean')  # You can change strategy to␣
 ↪'median', 'most_frequent', or 'constant'
    df_imputed = pd.DataFrame(imputer.fit_transform(df), columns=df.columns)
    return df_imputed

df = handle_missing_values(df)
def check_duplicates(df):
    duplicates = df.duplicated().sum()
```

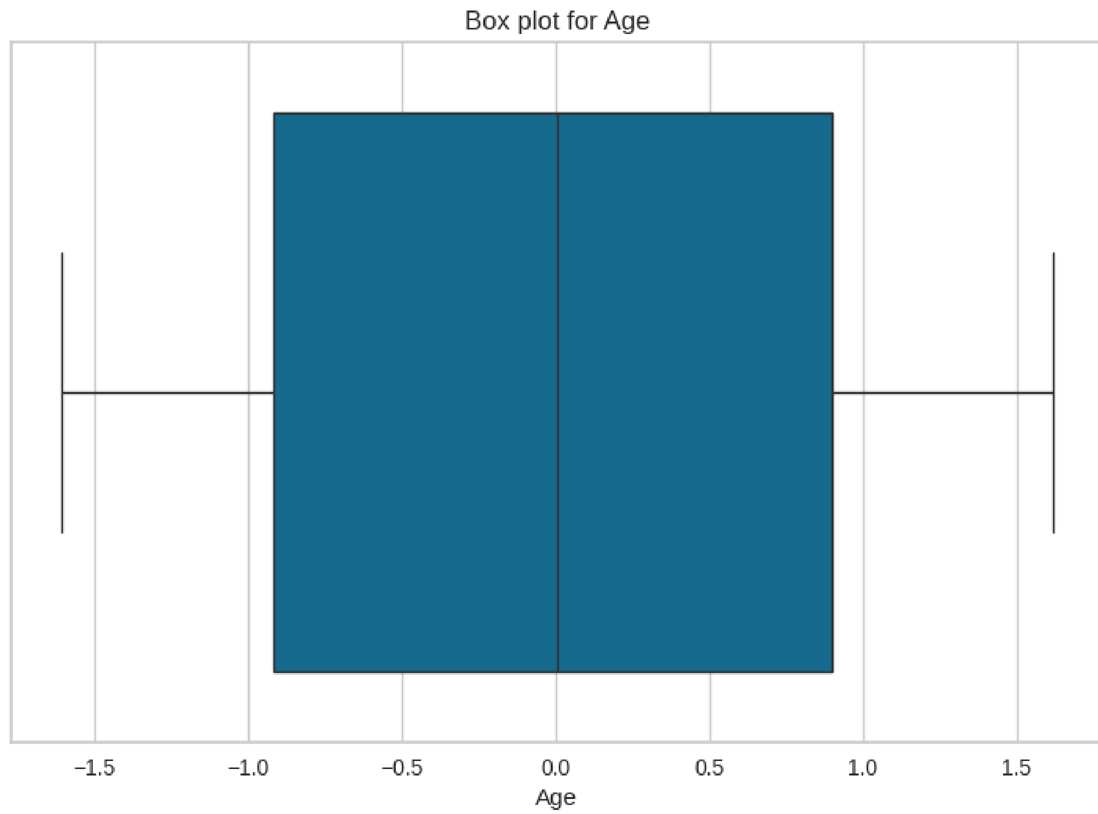
```
    print("Number of Duplicates: ", duplicates)
    return duplicates

duplicates = check_duplicates(df)
if duplicates > 0:
    df = df.drop_duplicates()
def scale_features(df):
    scaler = StandardScaler()
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    df[numeric_columns] = scaler.fit_transform(df[numeric_columns])
    return df

df = scale_features(df)
def check_outliers(df):
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    for col in numeric_columns:
        sns.boxplot(x=df[col])
        plt.title(f'Box plot for {col}')
        plt.show()

check_outliers(df)
def handle_outliers(df):
    # This example removes outliers using the IQR method
    numeric_columns = df.select_dtypes(include=[np.number]).columns
    for col in numeric_columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1
        df = df[~((df[col] < (Q1 - 1.5 * IQR)) | (df[col] > (Q3 + 1.5 * IQR)))]
    return df

df = handle_outliers(df)
def data_cleaning_suggestions(df):
    print("Step 1: Checking for Missing Values")
    check_missing_values(df)

    print("\nStep 2: Handling Missing Values")
    df = handle_missing_values(df)

    print("\nStep 3: Checking for Duplicates")
    duplicates = check_duplicates(df)

    if duplicates > 0:
        print("Removing Duplicates")
        df = df.drop_duplicates()

    print("\nStep 4: Scaling Numeric Features")
```

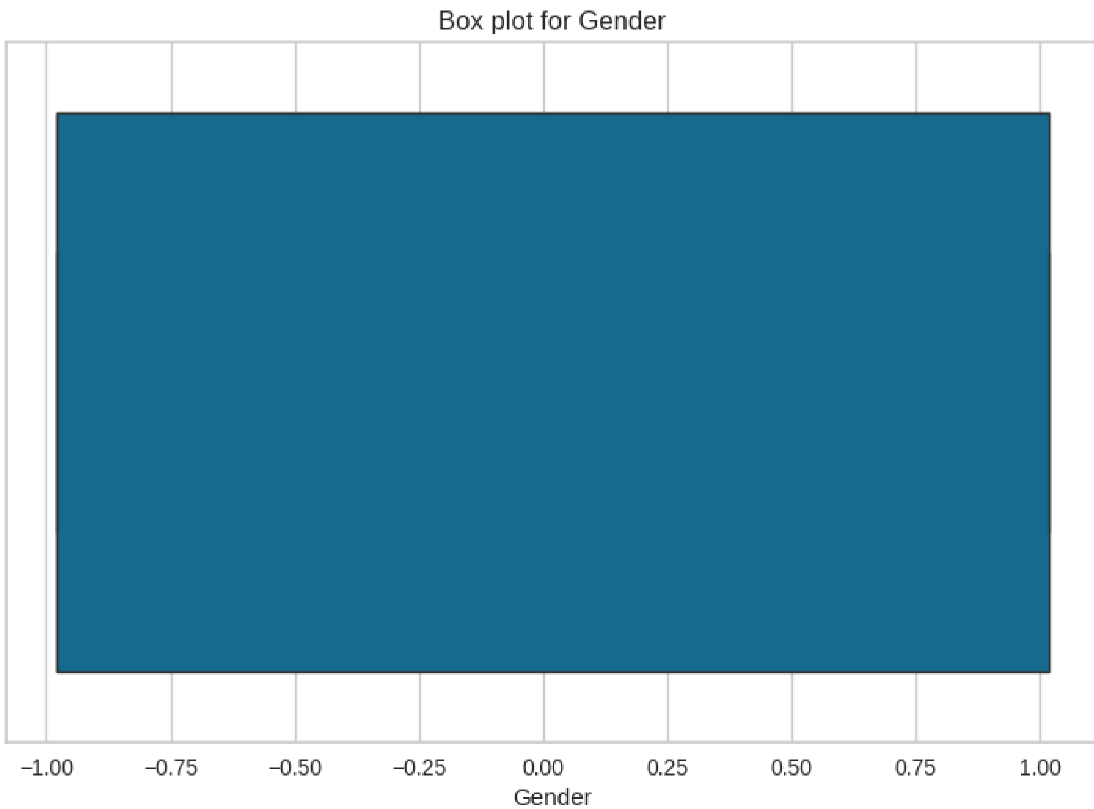
```
    df = scale_features(df)

    print("\nStep 5: Checking for Outliers")
    check_outliers(df)

    print("\nStep 6: Handling Outliers")
    df = handle_outliers(df)

    print("\nData Cleaning Completed")
    return df

# Run the data cleaning suggestions
df_cleaned = data_cleaning_suggestions(df)
```

```
Missing Values in Each Column:
 Series([], dtype: int64)
```

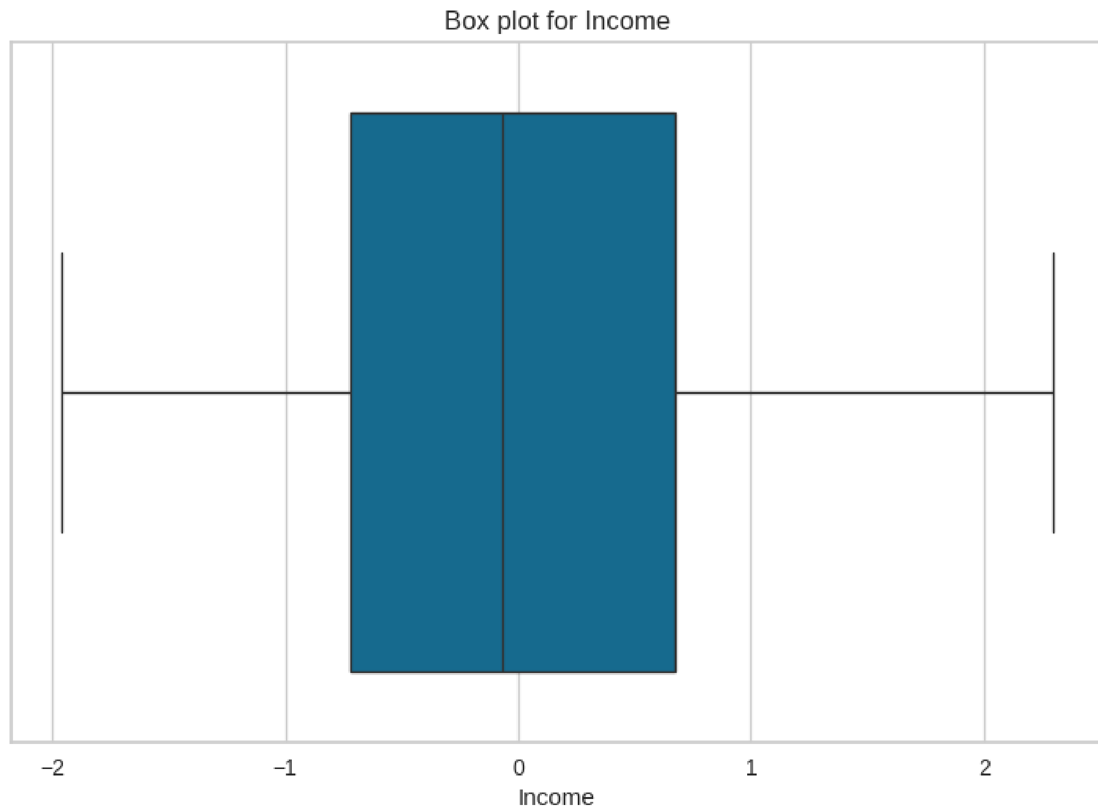


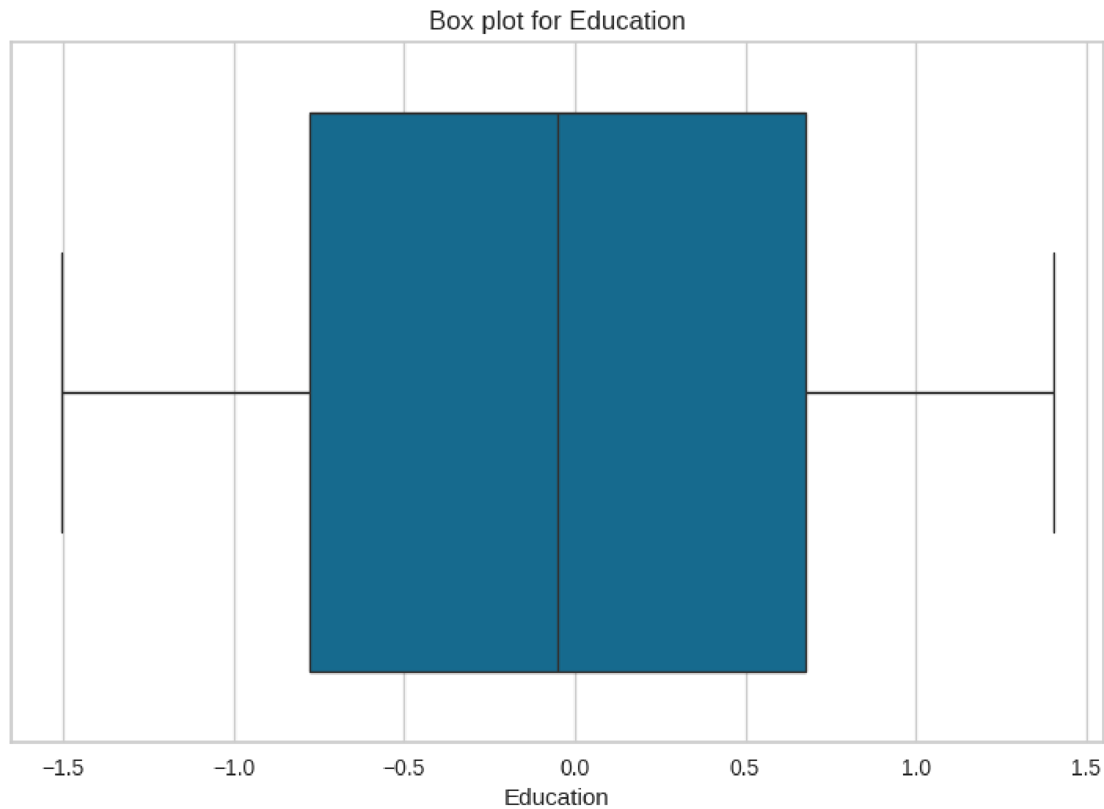

```
Number of Duplicates:  62
```

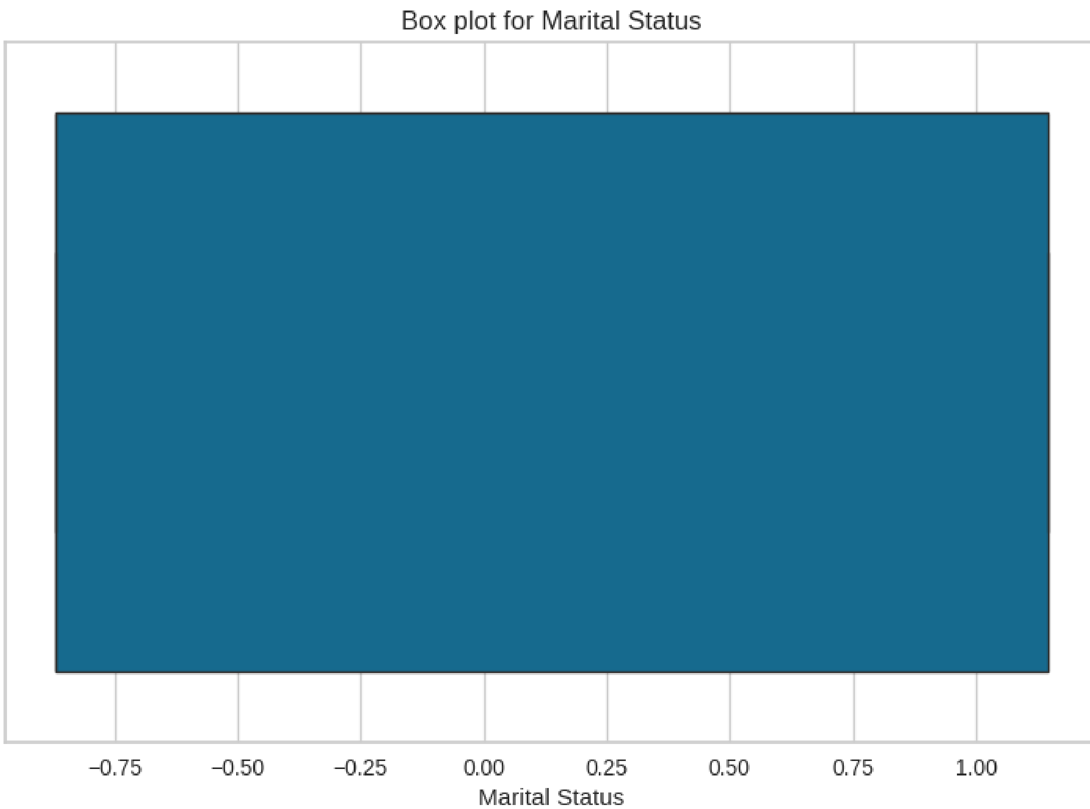

Box plot for Age
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age



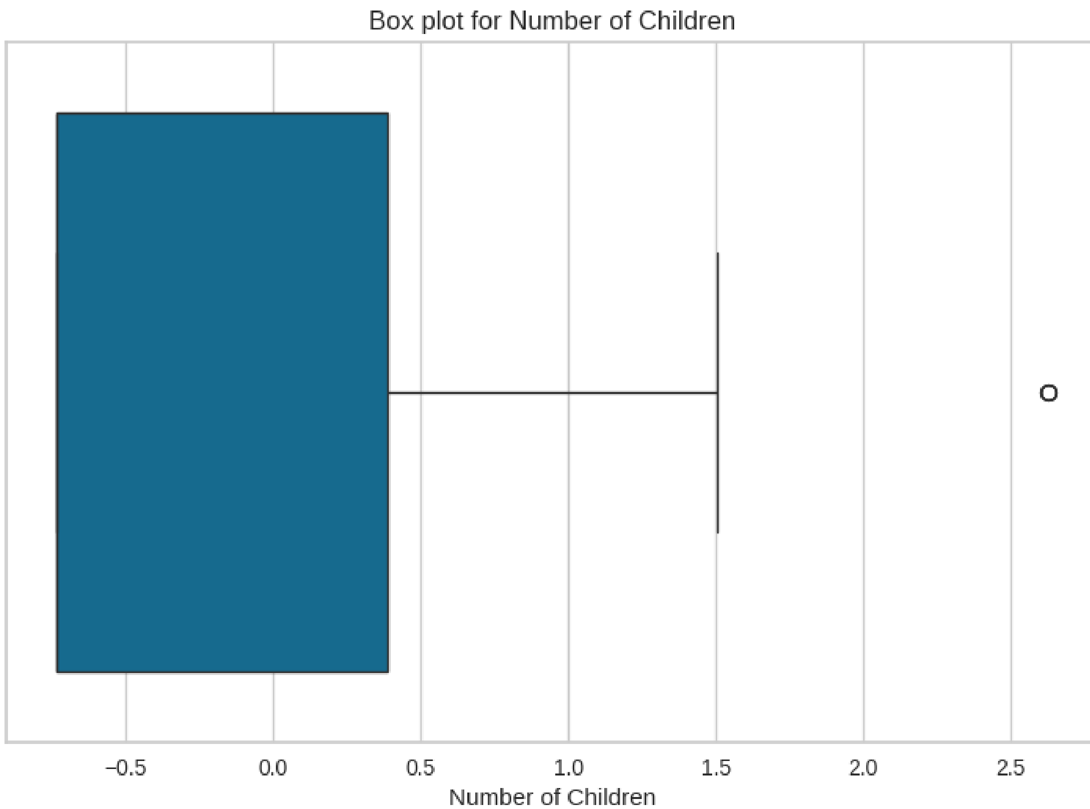
Box plot for Gender
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Gender

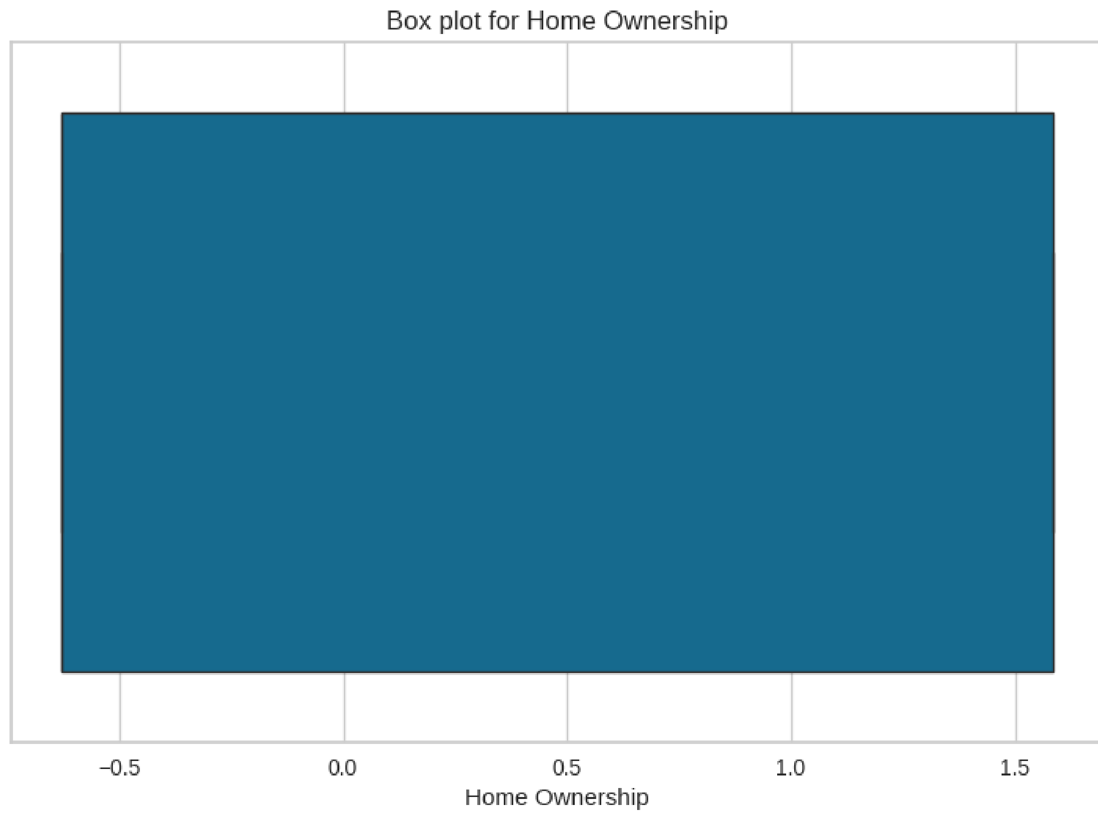

Box plot for Income
−2
−1
0
1
2
Income

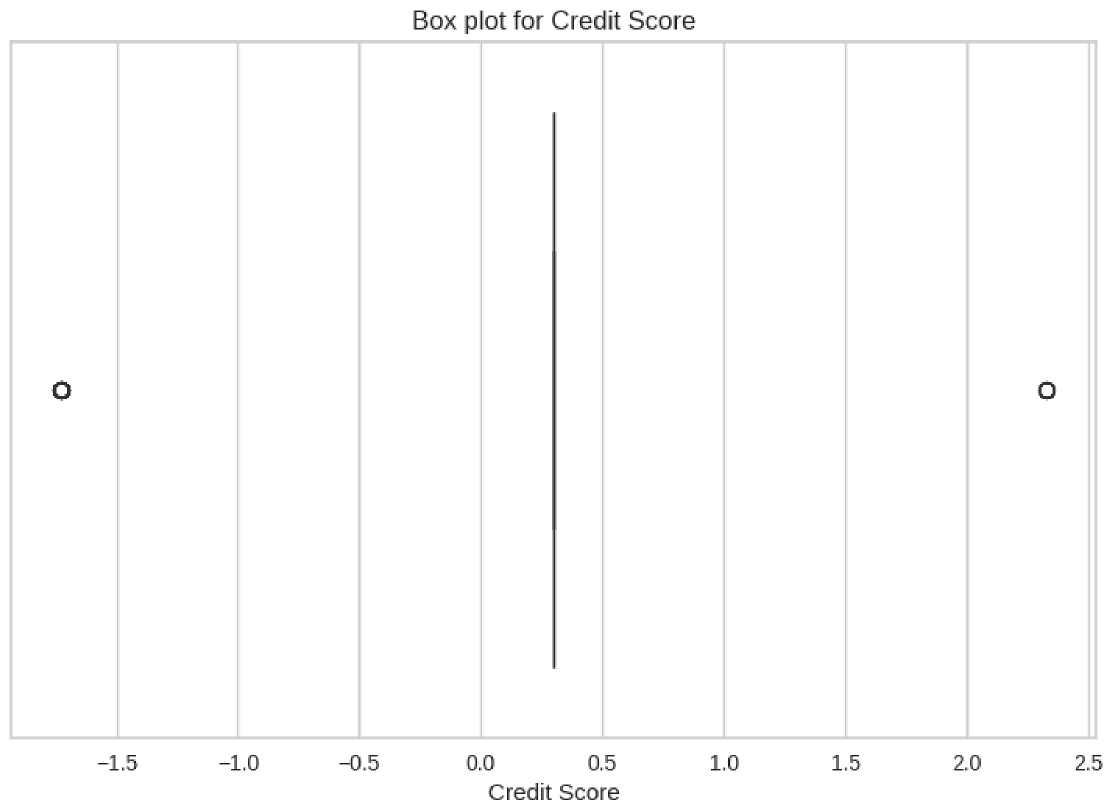

Box plot for Education
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Education



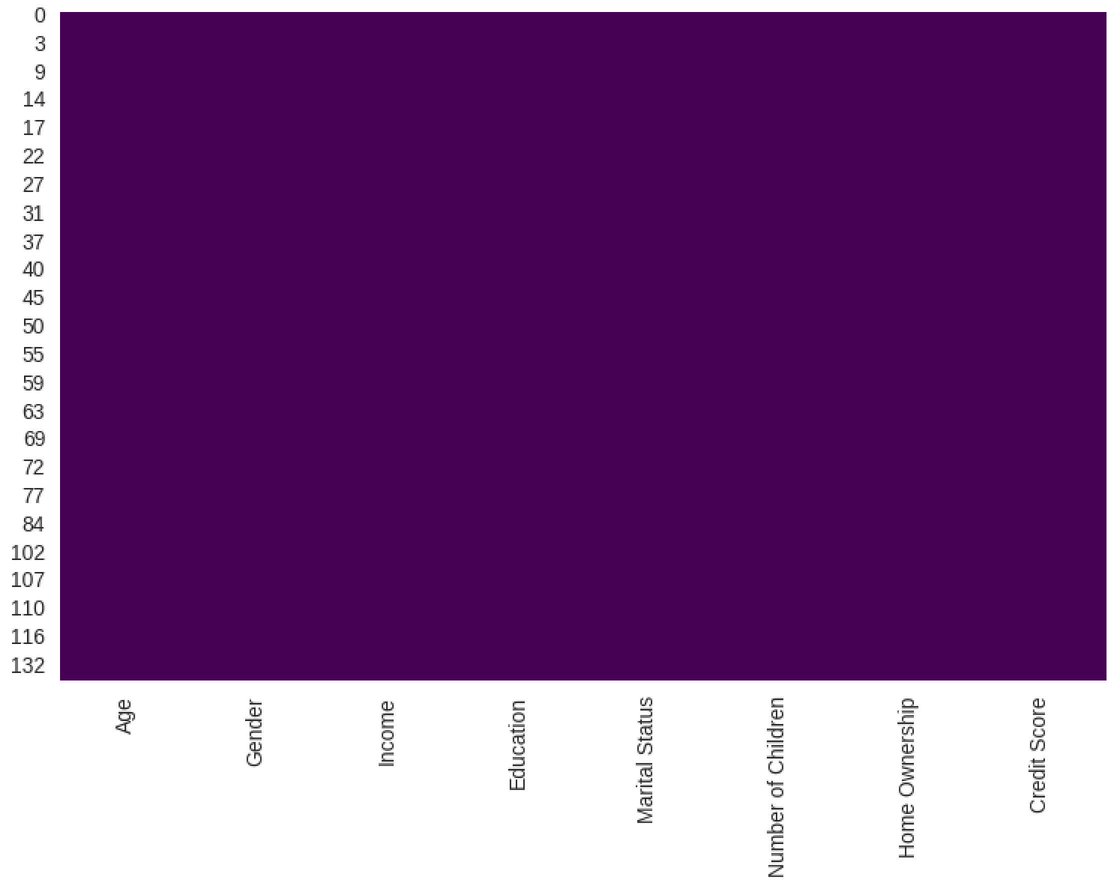
Box plot for Marital Status
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Marital Status


Box plot for Number of Children
−0.5
0.0
0.5
1.0
1.5
2.0
2.5
Number of Children

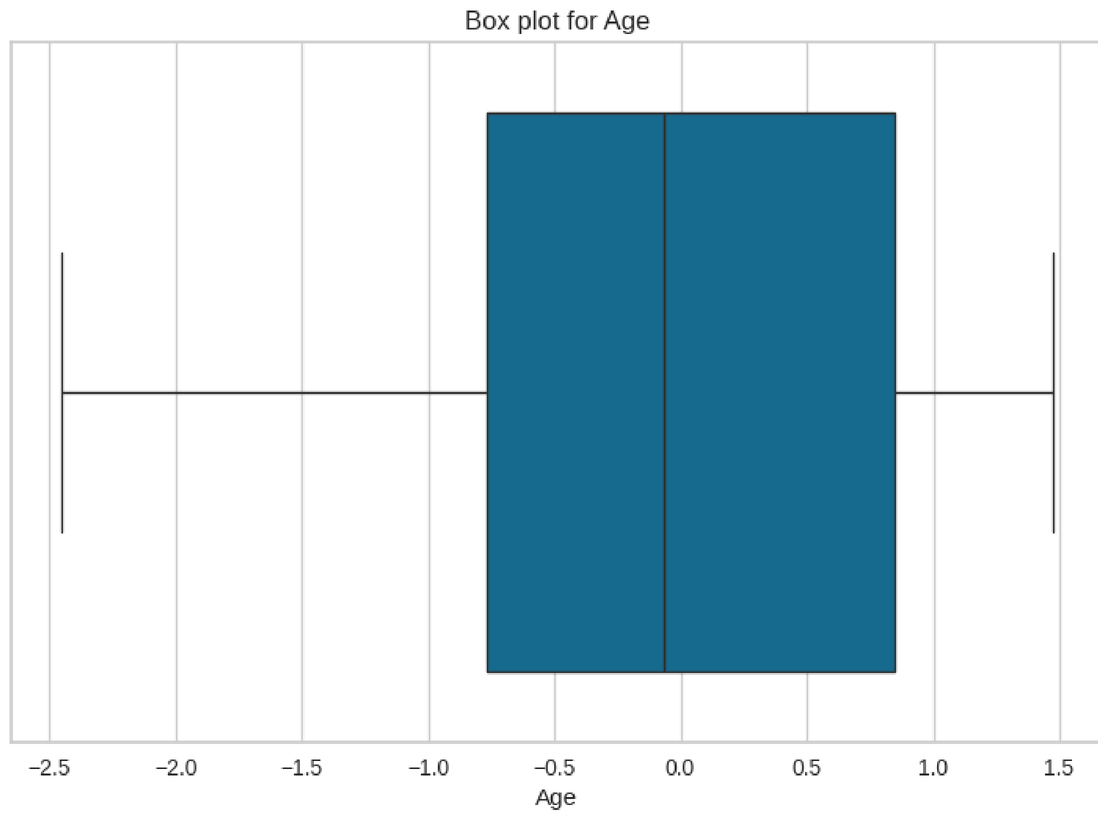

Box plot for Home Ownership
−0.5
0.0
0.5
1.0
1.5
Home Ownership

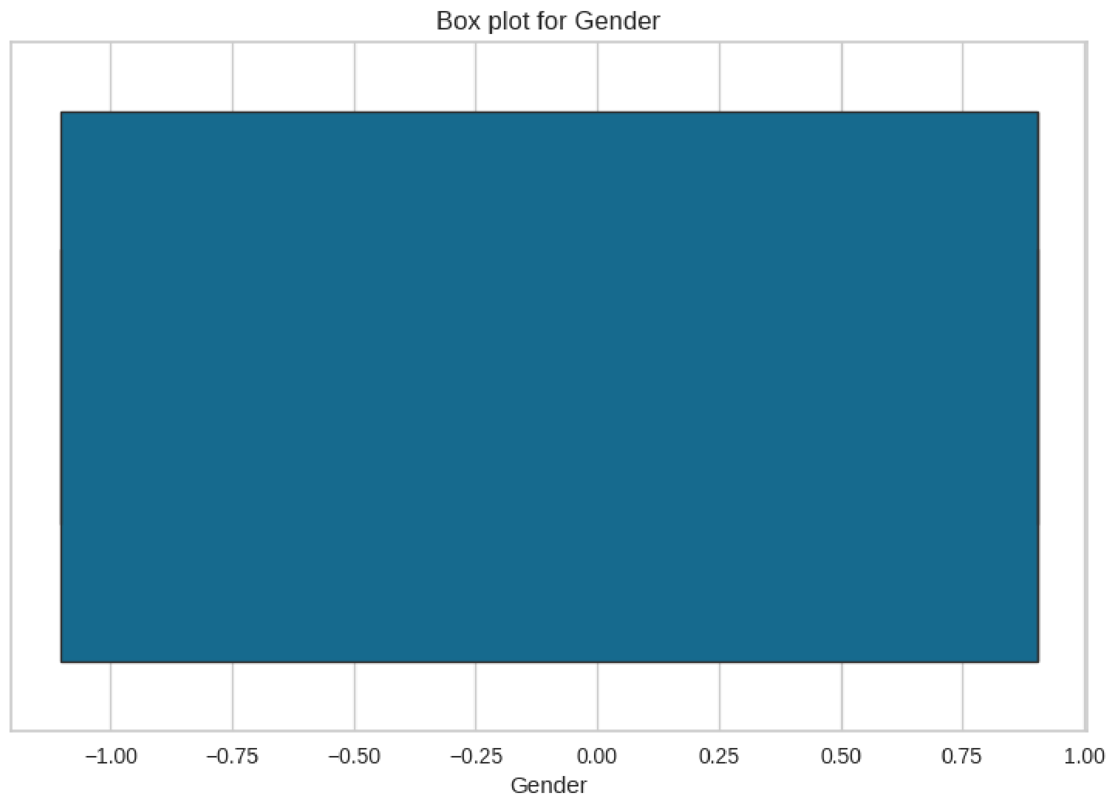


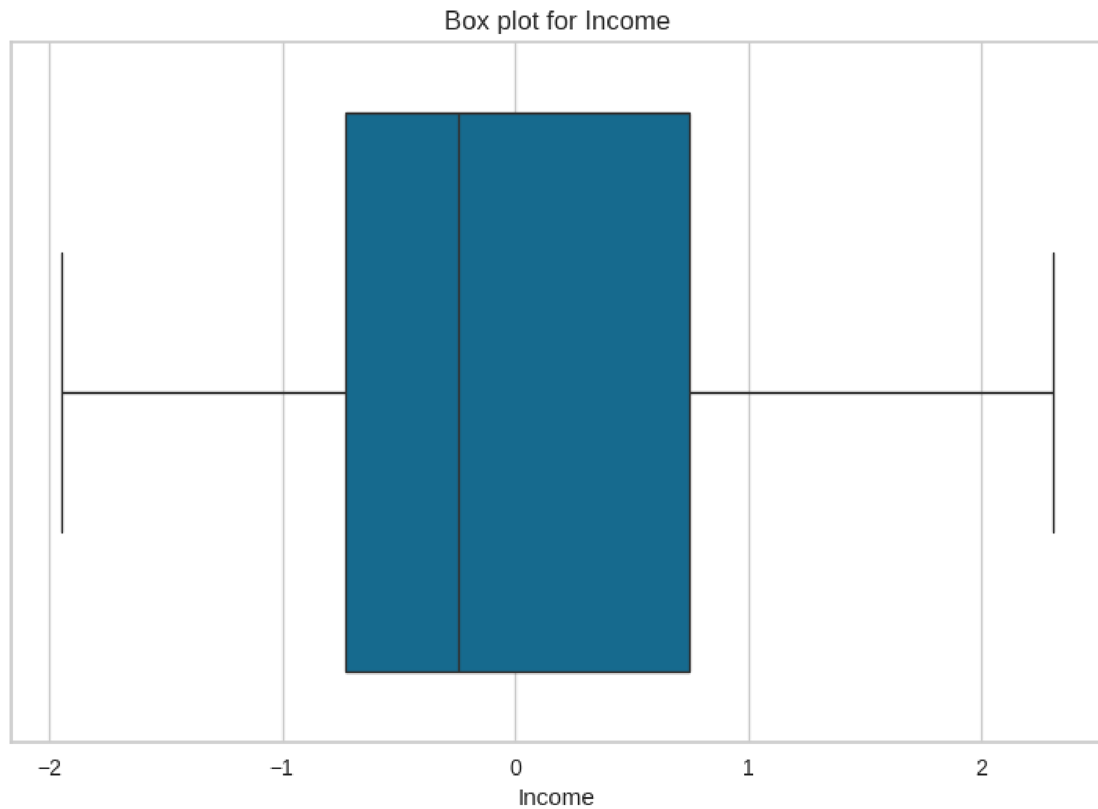

```
Step 1: Checking for Missing Values
Missing Values in Each Column:
 Series([], dtype: int64)
```

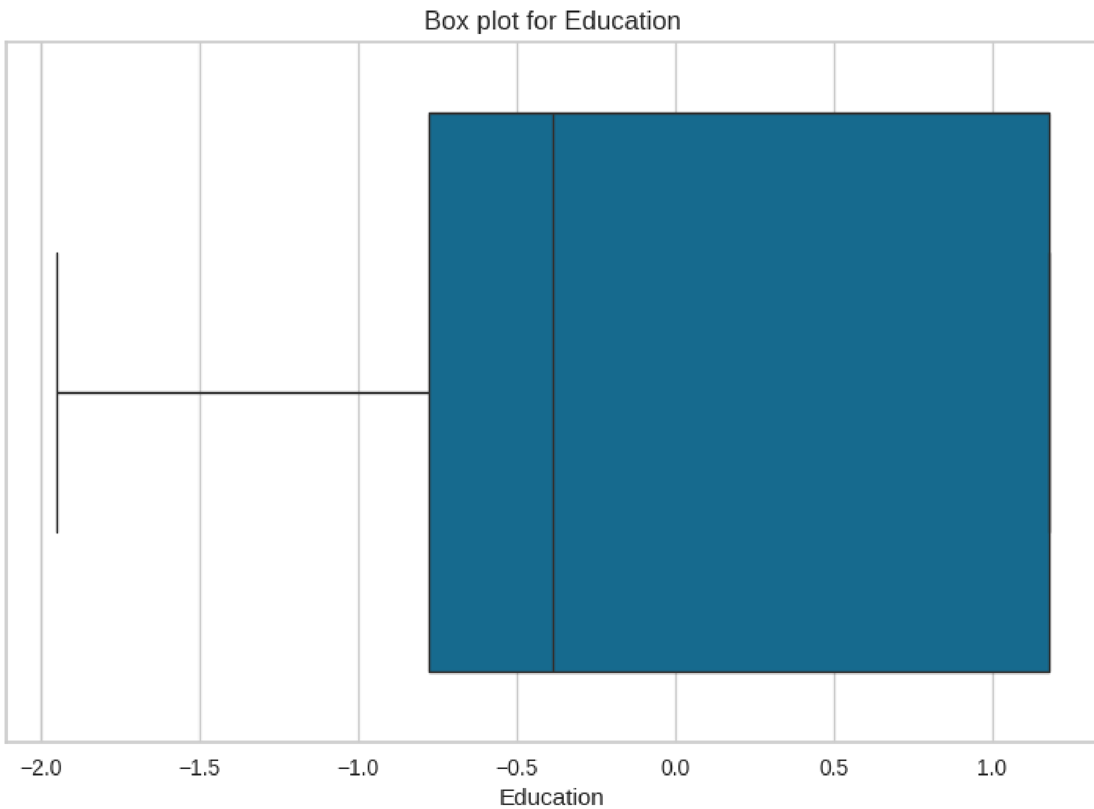


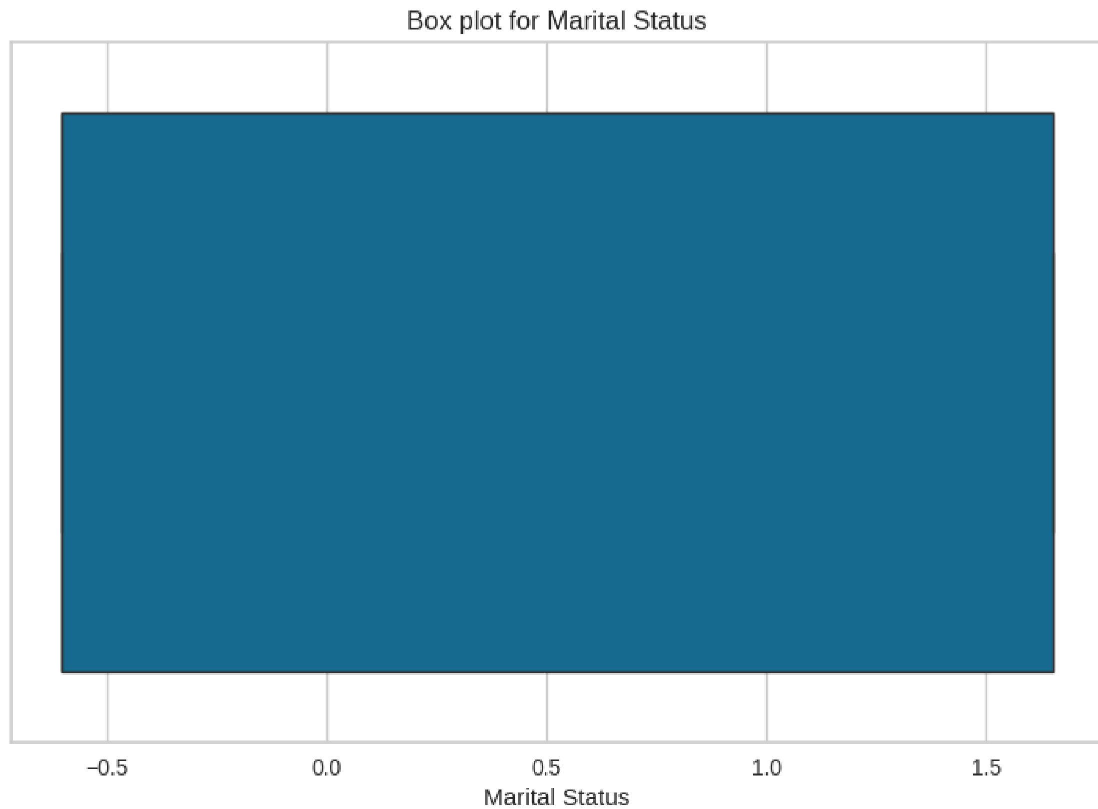

```
Step 2: Handling Missing Values

Step 3: Checking for Duplicates
Number of Duplicates:  0

Step 4: Scaling Numeric Features

Step 5: Checking for Outliers
```

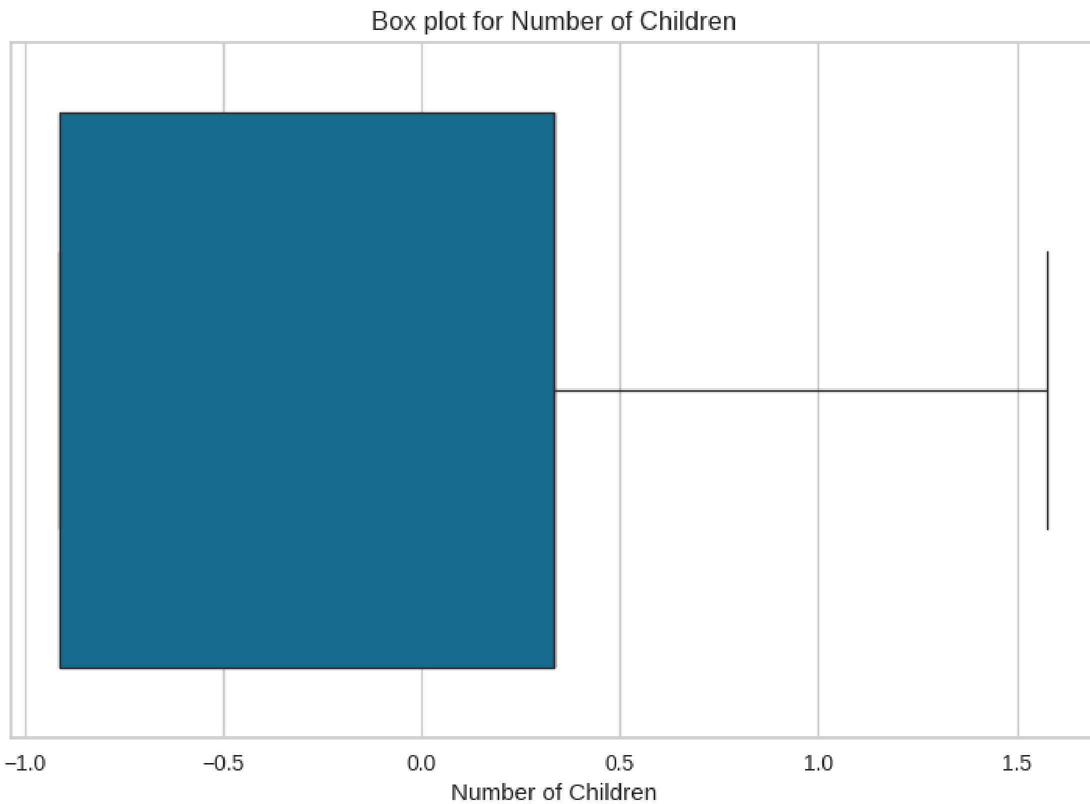

Box plot for Age
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
Age

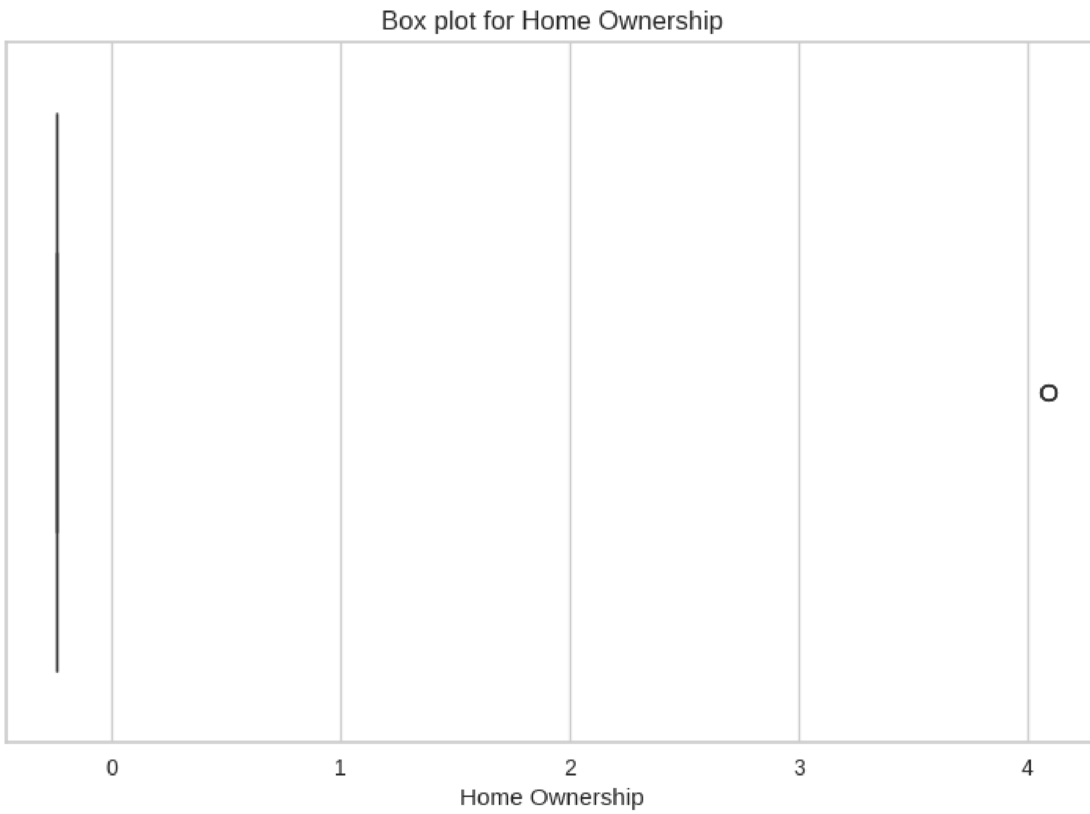

Box plot for Gender
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00
Gender

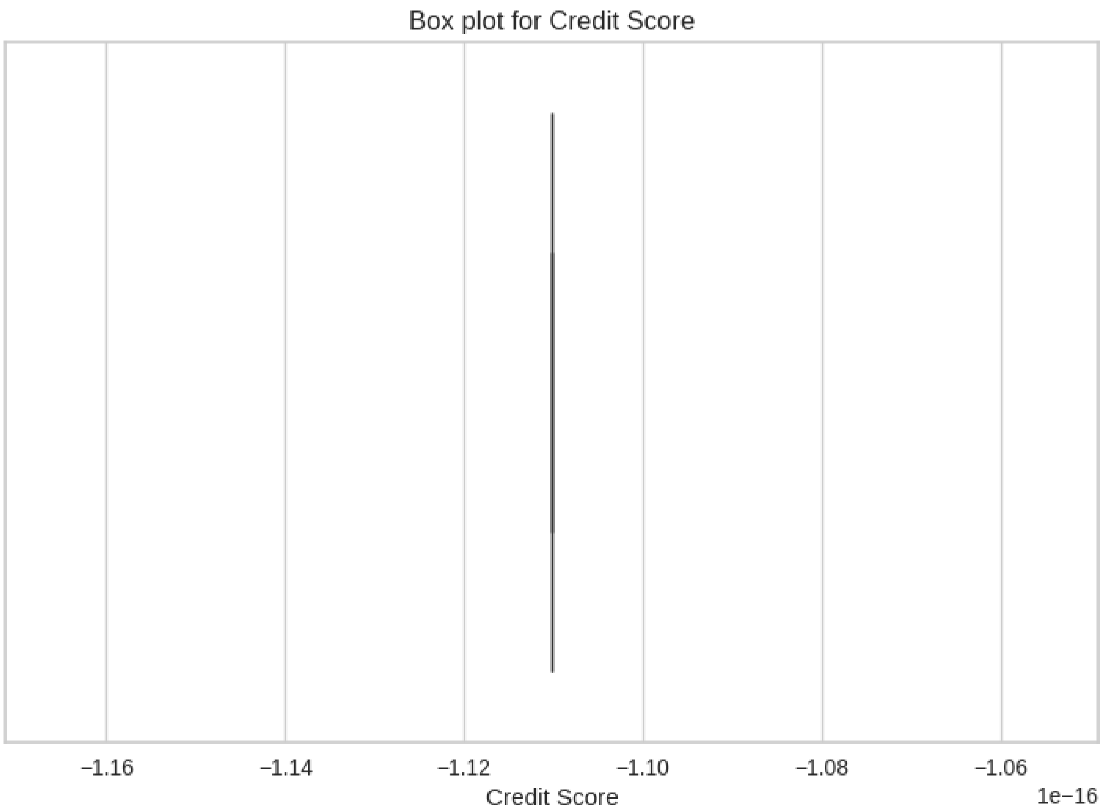

Box plot for Income
−2
−1
0
1
2
Income


Box plot for Education
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
Education


Box plot for Marital Status
−0.5
0.0
0.5
1.0
1.5
Marital Status


Box plot for Number of Children
−1.0
−0.5
0.0
0.5
1.0
1.5
Number of Children


Box plot for Home Ownership
0
1
2
3
4
Home Ownership



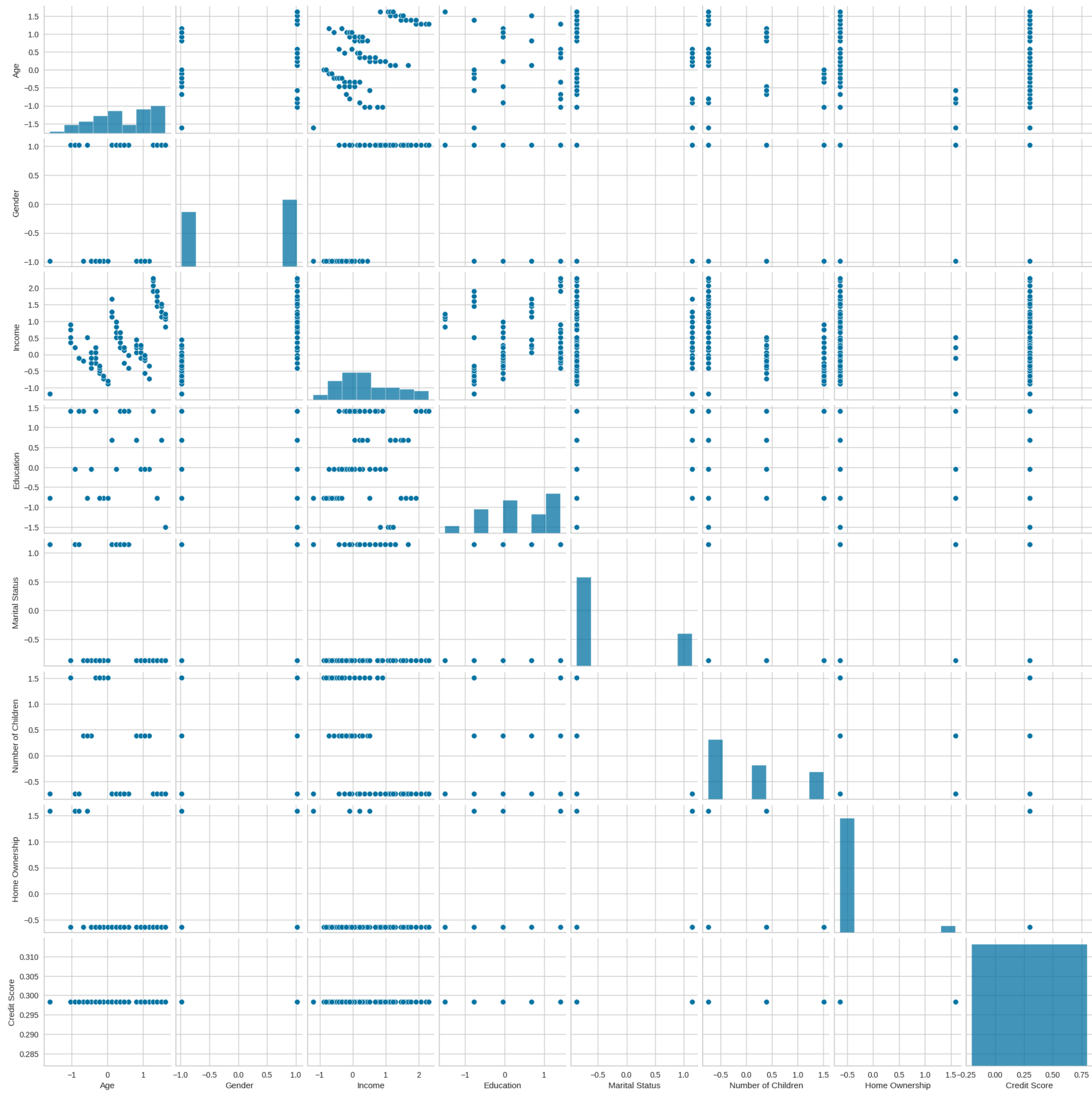

```
Step 6: Handling Outliers

Data Cleaning Completed
```

11 Visualizations

12 1. Pair Plot

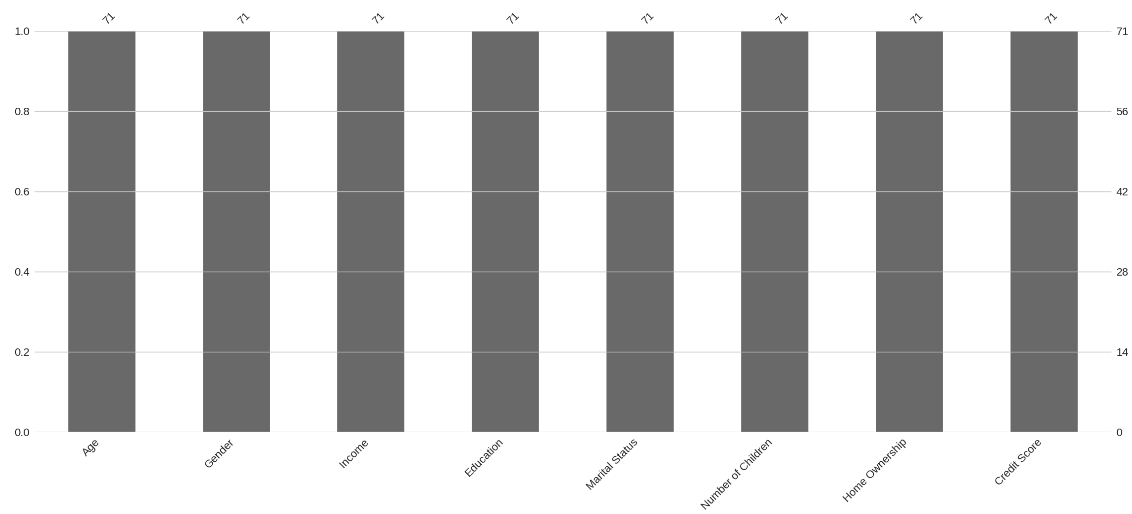
[35]:
```
sns.pairplot(df)
```

[35]:
```
<seaborn.axisgrid.PairGrid at 0x7f3d7de83250>
```



13 2. NULL Plot

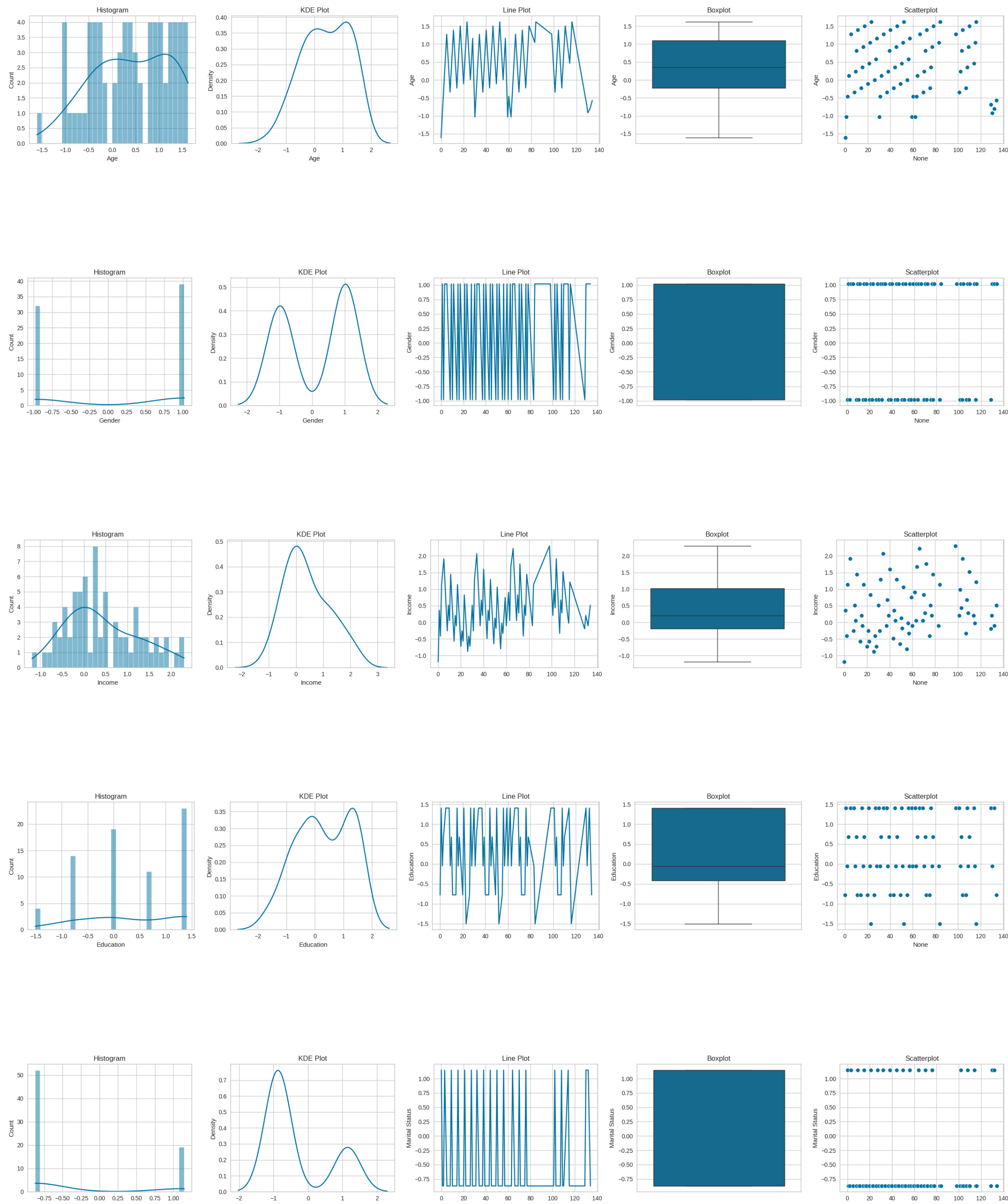
[43]:
```
import missingno as msno
# Null count analysis
null_plot = msno.bar(df)
```

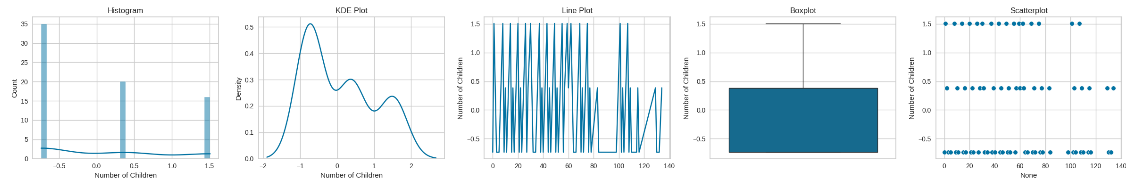


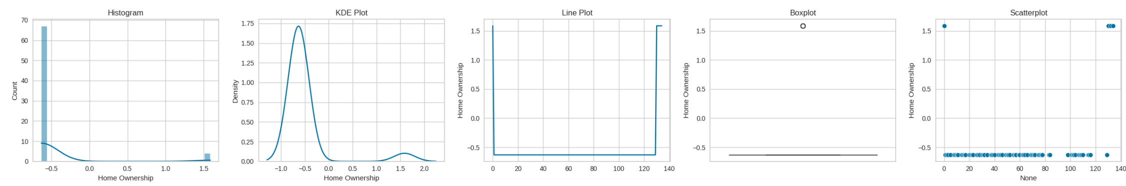
14 3. Other Important Plots

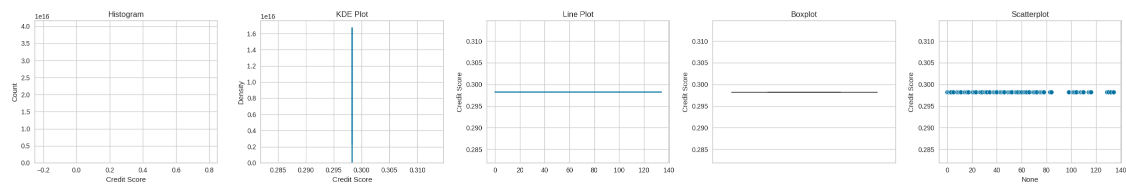
[44]:
```
def plots(df, variable):
  if df[variable].dtype != object:
    # define figure size
    fig, ax = plt.subplots(1, 5, figsize=(24, 4))

    # histogram
    sns.histplot(df[variable], bins=30, kde=True, ax=ax[0])
    ax[0].set_title('Histogram')

    # KDE plot
    sns.kdeplot(df[variable], ax=ax[1])
    ax[1].set_title('KDE Plot')

    # Line plot
    sns.lineplot(df[variable], ax=ax[2])
    ax[2].set_title('Line Plot')

    # boxplot
    sns.boxplot(y=df[variable], ax=ax[3])
    ax[3].set_title('Boxplot')

    # scatterplot
    sns.scatterplot(x=df.index, y=df[variable], ax=ax[4])
    ax[4].set_title('Scatterplot')

    plt.tight_layout()
    plt.show()
```

```python
for i in df.columns:
    plots(df ,i)
```









15 ML Modelling

[45]:
```
# combine X_train_res and y_train_res
train_data = pd.concat([X_train_res, y_train_res], axis=1)

from pycaret.classification import *
s = setup(data=train_data, target='Credit Score', session_id=123,
  normalize=True)
```

```
<pandas.io.formats.style.Styler at 0x7f3d6e1d0b50>
```

16 1. Comparing Models

[46]:
```
compare_models()
```

```
<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7f3d7dca8f40>

Processing:   0%|          | 0/65 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

```
[46]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                          metric_params=None, n_jobs=-1, n_neighbors=5, p=2,
                          weights='uniform')
```

17 2. KNN Model

```
[47]: knn = create_model('knn')
```

```
<IPython.core.display.HTML object>

<pandas.io.formats.style.Styler at 0x7f3d7bf8c6a0>

Processing:   0%|          | 0/4 [00:00<?, ?it/s]

<IPython.core.display.HTML object>
```

```
[48]: preds = predict_model(knn)
```

```
<pandas.io.formats.style.Styler at 0x7f3d7a2ddcc0>
```

```
[49]: preds
```

```
[49]:      Age  Gender  Income  Education  Marital Status  Number of Children  \
     140  29       0   47500          0               1                   0
     268  27       0   32785          0               1                   0
     228  27       0   37500          3               1                   0
     205  28       0   32500          0               1                   0
     103  43       1   92500          4               1                   0
     ..   …       …      …          …               …                   …
     11   29       0   68000          2               0                   2
     246  28       0   32148          0               1                   0
     238  28       0   32037          0               1                   0
     85   27       0   37500          3               1                   0
     76   50       1  155000          4               0                   0

          Home Ownership  Credit Score  prediction_label  prediction_score
     140             1             0                 0               1.0
     268             1             2                 2               1.0
     228             1             2                 2               1.0
     205             1             2                 2               1.0
     103             0             1                 1               1.0
     ..            …             …                 …               …
     11              0             0                 1               1.0
     246             1             2                 2               1.0
     238             1             2                 2               1.0
     85              1             2                 2               1.0
     76              0             1                 1               1.0
```

[81 rows x 10 columns]

[50]:
```
from sklearn.model_selection import cross_val_score

# Evaluate the ensemble model using cross-validation
scores = cross_val_score(knn, X_train_res, y_train_res, cv=20)
```

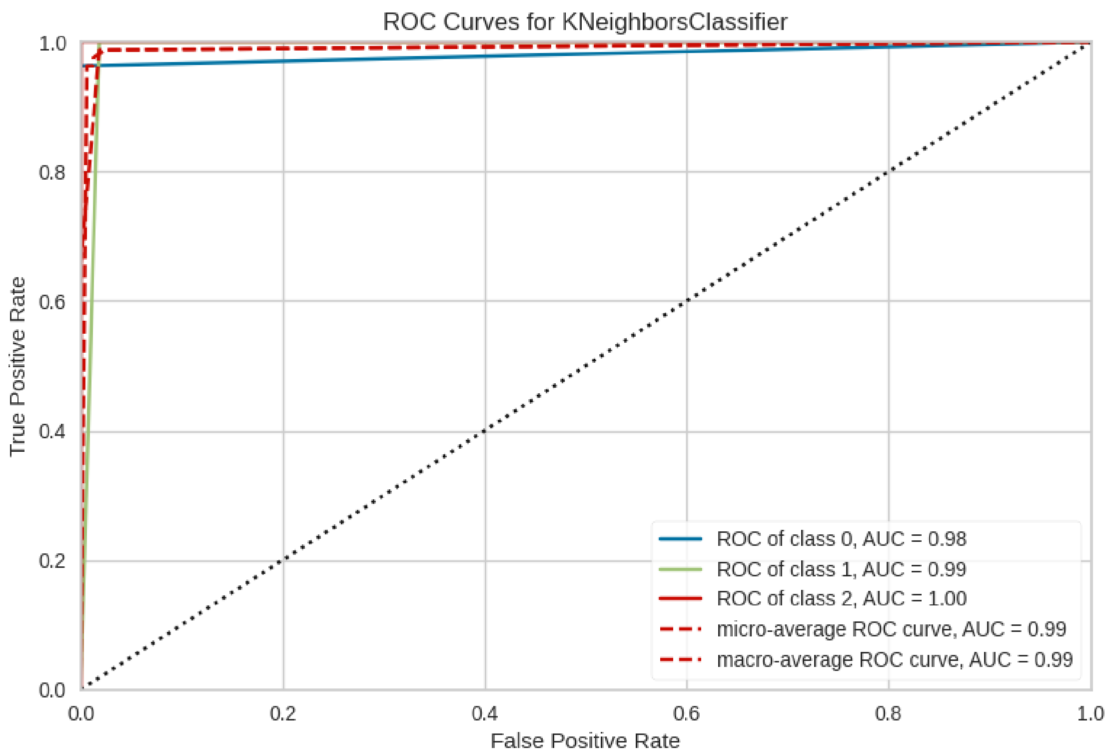
[51]:
```
preds = predict_model(knn)
```

<pandas.io.formats.style.Styler at 0x7f3d7de55a50>

18 3. Plots

[52]:
```
plot_model(knn)
```

<IPython.core.display.HTML object>

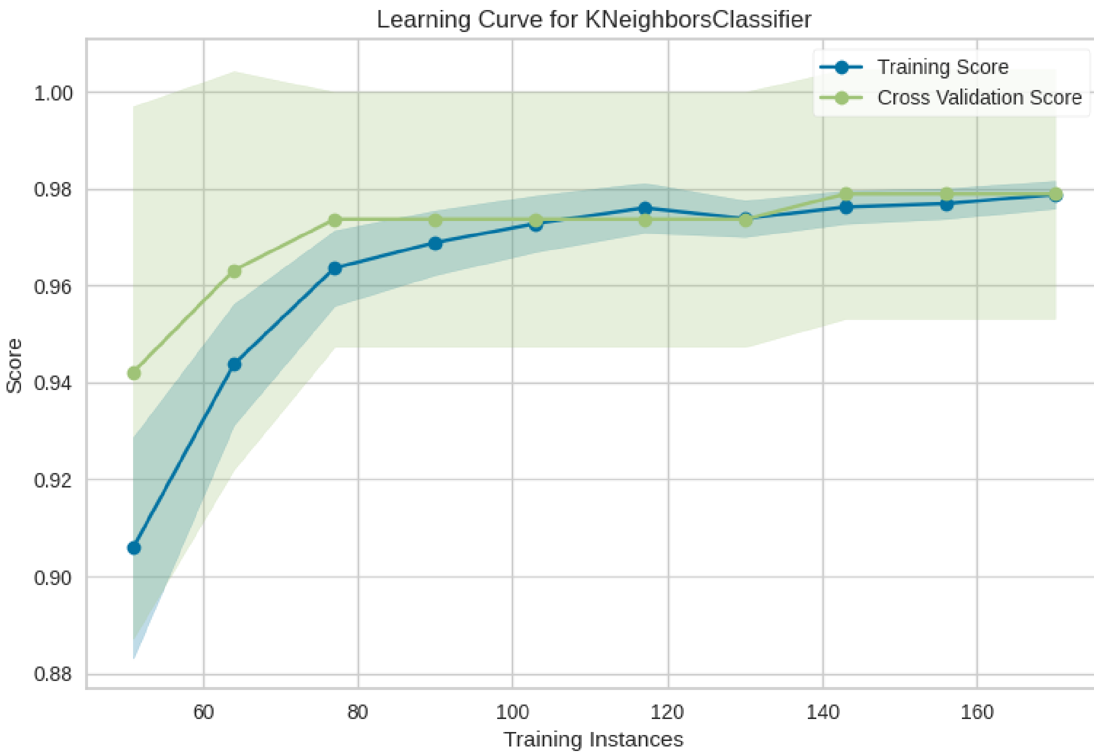


[53]:
```
plot_model(knn, plot = 'error')
```

<IPython.core.display.HTML object>

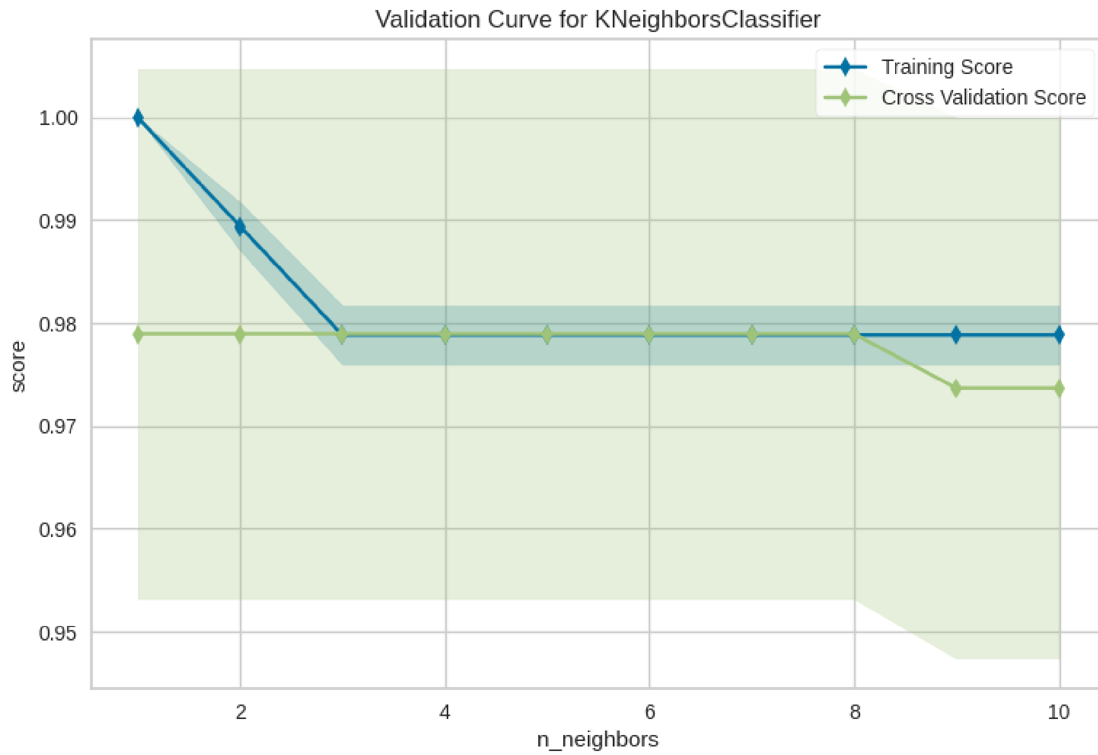
[54]:
```
plot_model(knn, plot = 'learning')
```

```
<IPython.core.display.HTML object>
```



[55]:
```
plot_model(knn, plot = 'vc')
```

```
<IPython.core.display.HTML object>
```



19 4. Traditional Approach

[56]:
```
from sklearn.neighbors import KNeighborsClassifier
```

[57]:
```
clf = KNeighborsClassifier()


# Fit the Extra Trees Classifier object to the dataset

clf.fit(X_train_res, y_train_res)

scores = cross_val_score(knn, X_train_res, y_train_res, cv=20)

# Predict the labels for the test data
y_preds = clf.predict(X_test)
```

20 5. Classification Report

[58]:
```
from sklearn.metrics import classification_report, confusion_matrix
print("Classification Report")
print(classification_report(y_test, y_preds))
```

```
Classification Report
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         5
           1       1.00      1.00      1.00        23
           2       1.00      1.00      1.00         5

    accuracy                           1.00        33
   macro avg       1.00      1.00      1.00        33
weighted avg       1.00      1.00      1.00        33
```

21 100% Accuracy

22 6. Confusion Matrix

[59]:
```
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_preds))
```

```
Confusion Matrix:
[[ 5  0  0]
 [ 0 23  0]
 [ 0  0  5]]
```