# EC 9560 DATA MINING
# LAB 03

**PRIYATHARSINI S.**
**2020E122**
**SEMESTER 7**
**30 OCT 2024**

# Check Null Values

```python
# Check for null values
null_values = data.isnull().sum()
print("Null Values in Each Column:")
print(null_values)
```

```
Null Values in Each Column:
id                            0
person_age                    0
person_income                 0
person_home_ownership         0
person_emp_length             0
loan_intent                   0
loan_grade                    0
loan_amnt                     0
loan_int_rate                 0
loan_percent_income           0
cb_person_default_on_file     0
cb_person_cred_hist_length    0
loan_status                   0
dtype: int64
```

# Check Duplicate Values

```python
# Check for duplicates
duplicate_rows = data.duplicated().sum()
print("\nTotal Duplicate Rows:", duplicate_rows)
```

```

Total Duplicate Rows: 0
```

```python
data.shape
```

```
(58645, 13)
```

```
data.dtypes
```

| | 0 |
|---|---|
| id | int64 |
| person_age | int64 |
| person_income | int64 |
| person_home_ownership | object |
| person_emp_length | float64 |
| loan_intent | object |
| loan_grade | object |
| loan_amnt | int64 |
| loan_int_rate | float64 |
| loan_percent_income | float64 |
| cb_person_default_on_file | object |
| cb_person_cred_hist_length | int64 |
| loan_status | int64 |

## Encode the Categorical Columns

```
from sklearn.preprocessing import LabelEncoder

# Create DataFrame
df = pd.DataFrame(data)

# Initialize the LabelEncoder
label_encoder = LabelEncoder()

# Columns to be label encoded
categorical_columns = [
    'person_home_ownership',
    'loan_intent',
    'loan_grade',
    'cb_person_default_on_file'
]

# Convert person_emp_length to int
df['person_emp_length'] = pd.to_numeric(df['person_emp_length'],
errors='coerce').fillna(0).astype(int)

# Apply label encoding to categorical columns
for column in categorical_columns:
    df[column] = label_encoder.fit_transform(df[column])

# Display the updated DataFrame
df.head()
```

```
1  df.dtypes
```

| | 0 |
|---|---|
| id | int64 |
| person_age | int64 |
| person_income | int64 |
| person_home_ownership | int64 |
| person_emp_length | int64 |
| loan_intent | int64 |
| loan_grade | int64 |
| loan_amnt | int64 |
| loan_int_rate | float64 |
| loan_percent_income | float64 |
| cb_person_default_on_file | int64 |
| cb_person_cred_hist_length | int64 |
| loan_status | int64 |

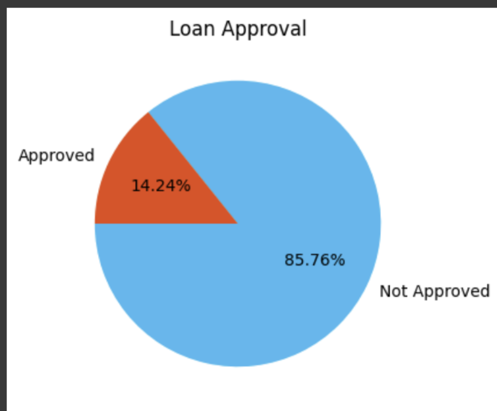
**dtype:** object

```
[12]
1  #print target classes and its count
2  data['loan_status'].value_counts()
```

| loan_status | count |
|---|---|
| 0 | 50295 |
| 1 | 8350 |

**dtype:** int64

```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize = (4, 4))

ax.pie(data.loan_status.value_counts(), labels = ["Not Approved", "Approved"],
       autopct = '%1.2f%%', startangle = 180, colors = ["#0EB8F1", "#F1480F"])

ax.set_title("Loan Approval")
plt.show()
```




## UnBalanced data

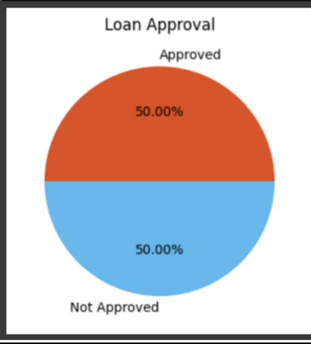
## Handle UnBalanced Data Using Down Sampling

```python
import pandas as pd

# Combine X and y into a single DataFrame for easier sampling
data = pd.concat([X, y], axis=1)

# Separate the classes
minority_class = data[data['loan_status'] == 1]
majority_class = data[data['loan_status'] == 0]

# Randomly sample from the majority class to match the minority class size
majority_class_downsampled = majority_class.sample(n=len(minority_class),
random_state=42)

# Concatenate the downsampled majority class with the minority class
balanced_data = pd.concat([minority_class, majority_class_downsampled])

# Separate features and target variable
X_resampled = balanced_data.drop('loan_status', axis=1)
y_resampled = balanced_data['loan_status']
```

```python
# Shape of resampled features
print("Shape of X_resampled:", X_resampled.shape)

# Shape of resampled target
print("Shape of y_resampled:", y_resampled.shape)
```

```
Shape of X_resampled: (16700, 12)
Shape of y_resampled: (16700,)
```

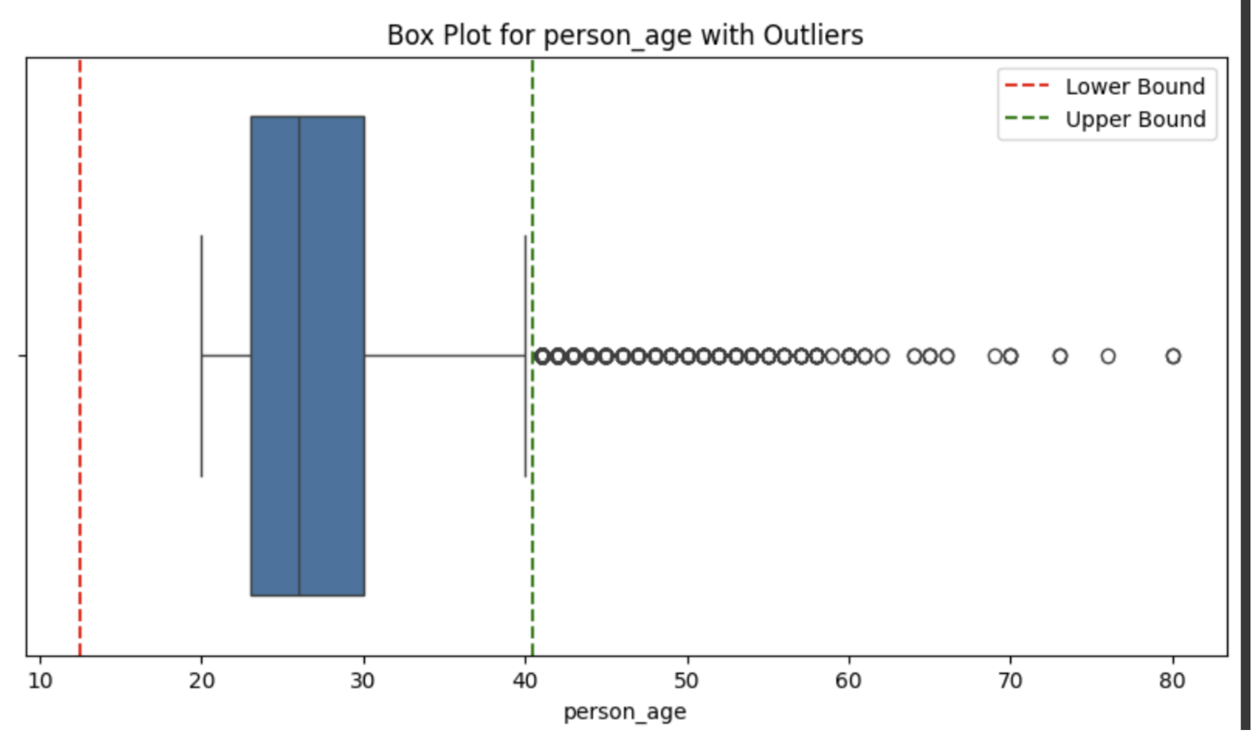
```python
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize = (4, 4))

ax.pie(balanced_data.loan_status.value_counts(), labels = ["Not Approved",
"Approved"],
        autopct = '%1.2f%%', startangle = 180, colors = ["#0EB8F1", "#F1480F"])

ax.set_title("Loan Approval")
plt.show()
```

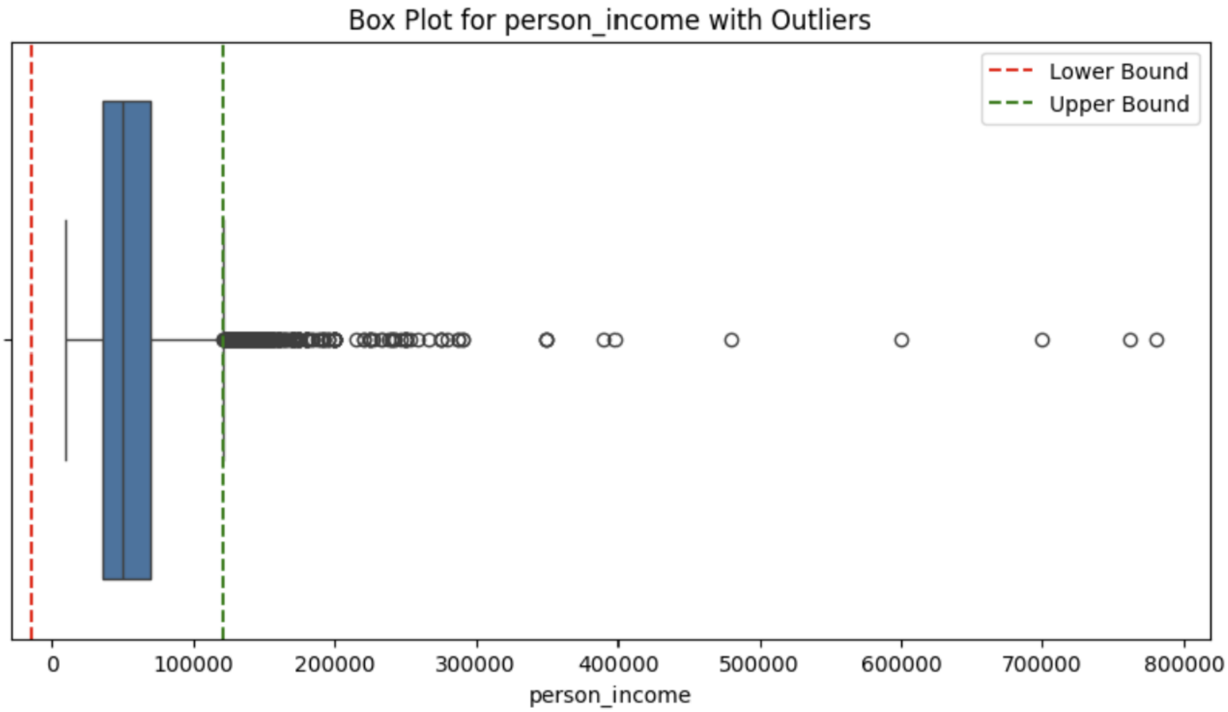


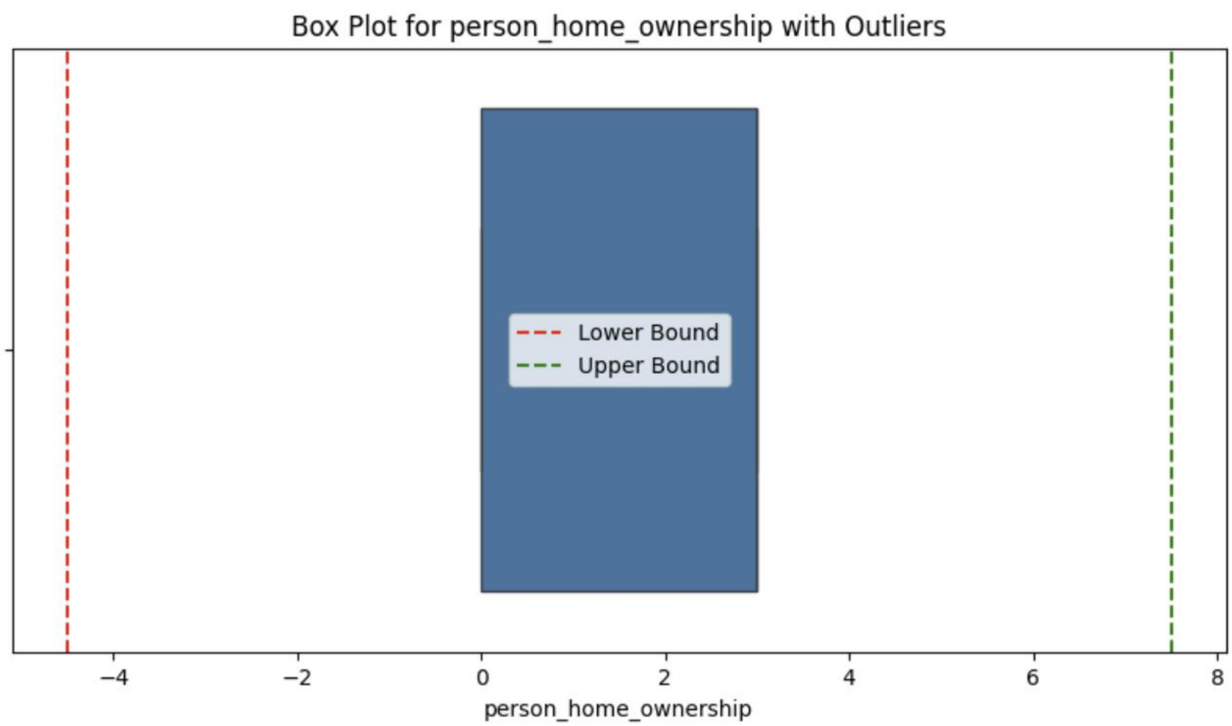

## Checking Outliers

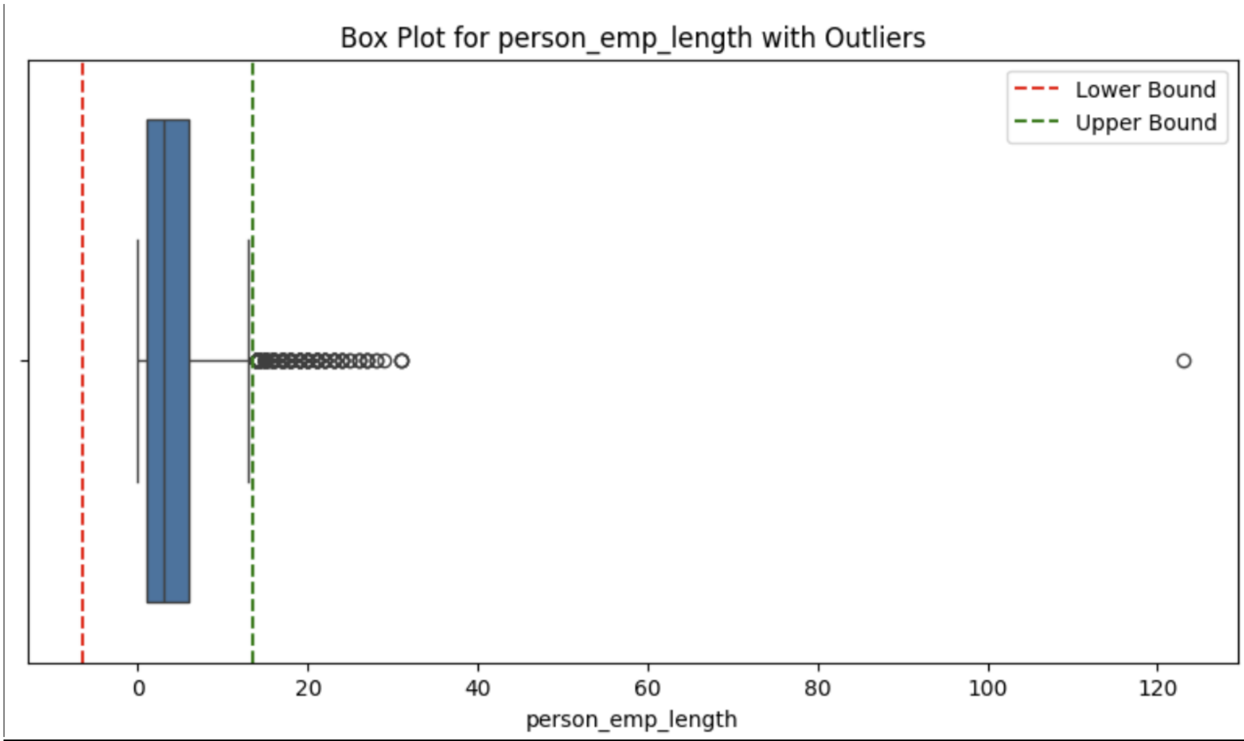
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from scipy import stats

# 1. Check for outliers using the IQR method
Q1 = df.describe().loc['25%']
Q3 = df.describe().loc['75%']
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 2. Identify outliers
outliers = df[~((df >= lower_bound) & (df <= upper_bound)).all(axis=1)]

# 3. Plotting
plt.figure(figsize=(15, 10))

# Create box plots for numeric columns to visualize outliers
numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()

for col in numeric_columns:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x=df[col])
    plt.axvline(x=lower_bound[col], color='red', linestyle='--', label='Lower
Bound')
```

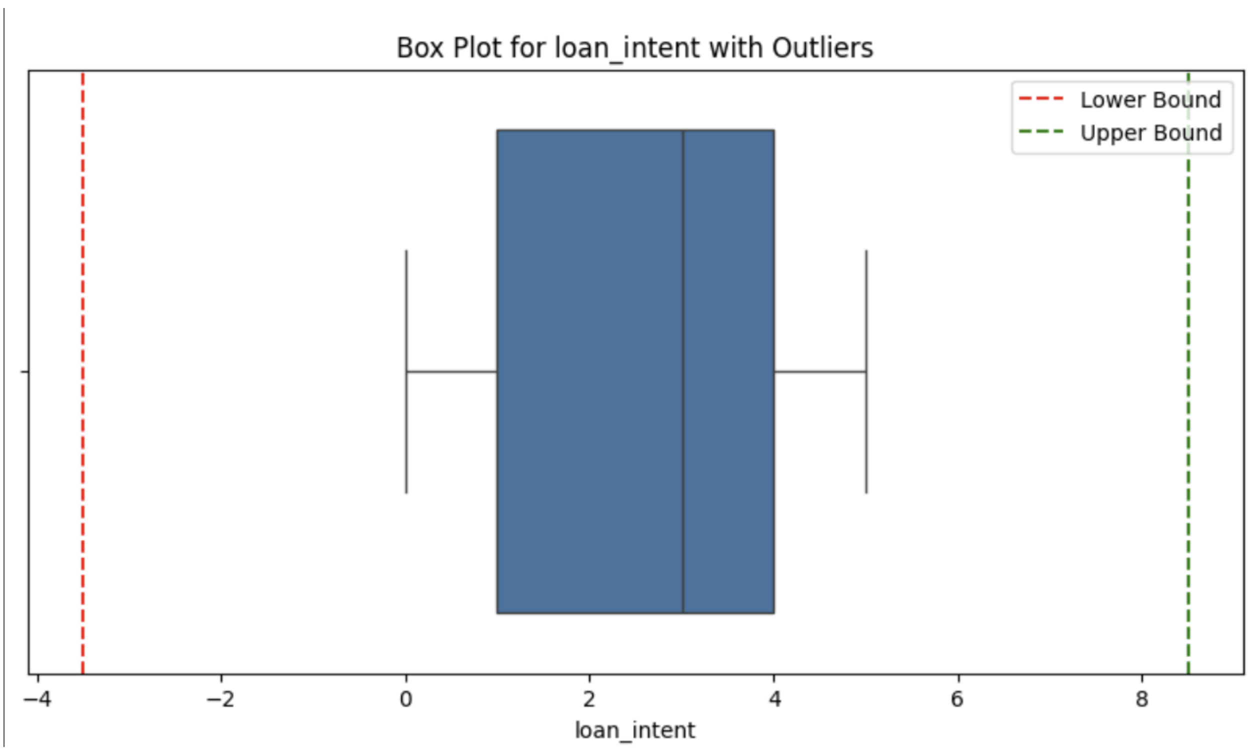
```
    plt.axvline(x=upper_bound[col], color='green', linestyle='--', label='Upper
Bound')
    plt.title(f'Box Plot for {col} with Outliers')
    plt.xlabel(col)
    plt.legend()
    plt.show()

# Optional: Scatter plot for two specific variables to visualize relationships
and outliers
plt.figure(figsize=(10, 5))
sns.scatterplot(data=df, x='person_income', y='loan_amnt', color='blue',
label='Data Points')
sns.scatterplot(data=outliers, x='person_income', y='loan_amnt', color='red',
label='Outliers')
plt.title('Scatter Plot of Person Income vs Loan Amount')
plt.xlabel('Person Income')
plt.ylabel('Loan Amount')
plt.legend()
plt.show()
```

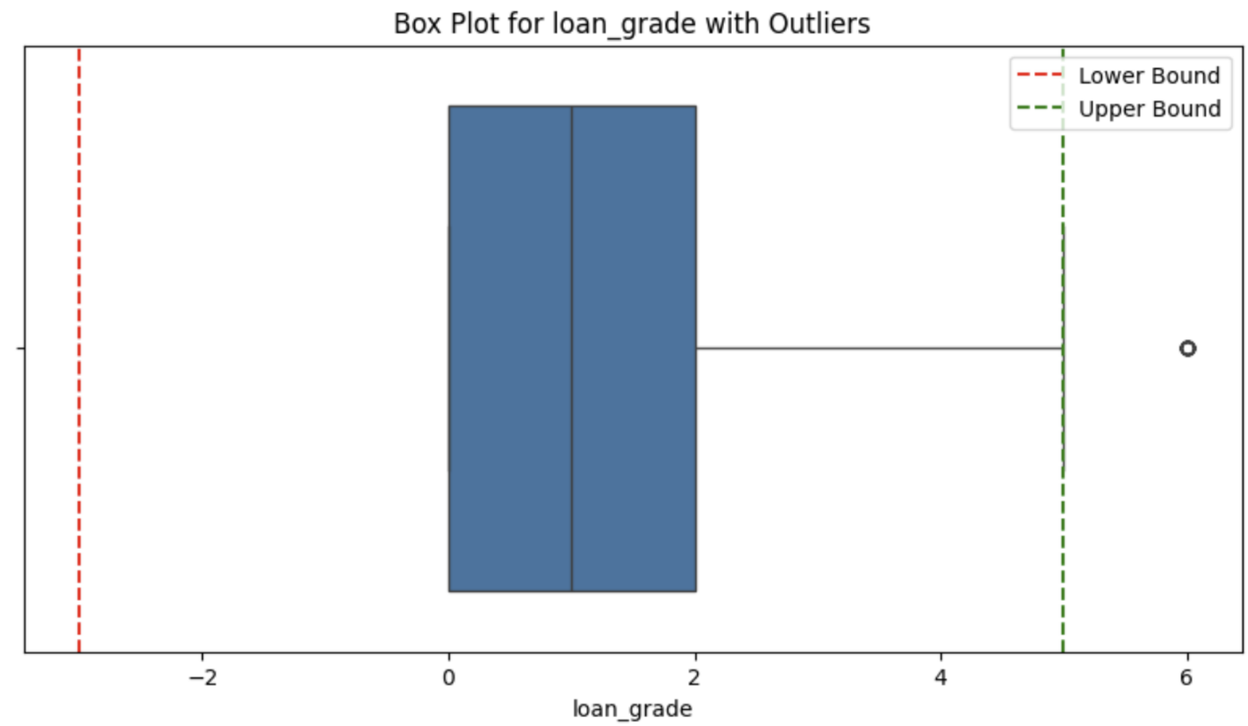


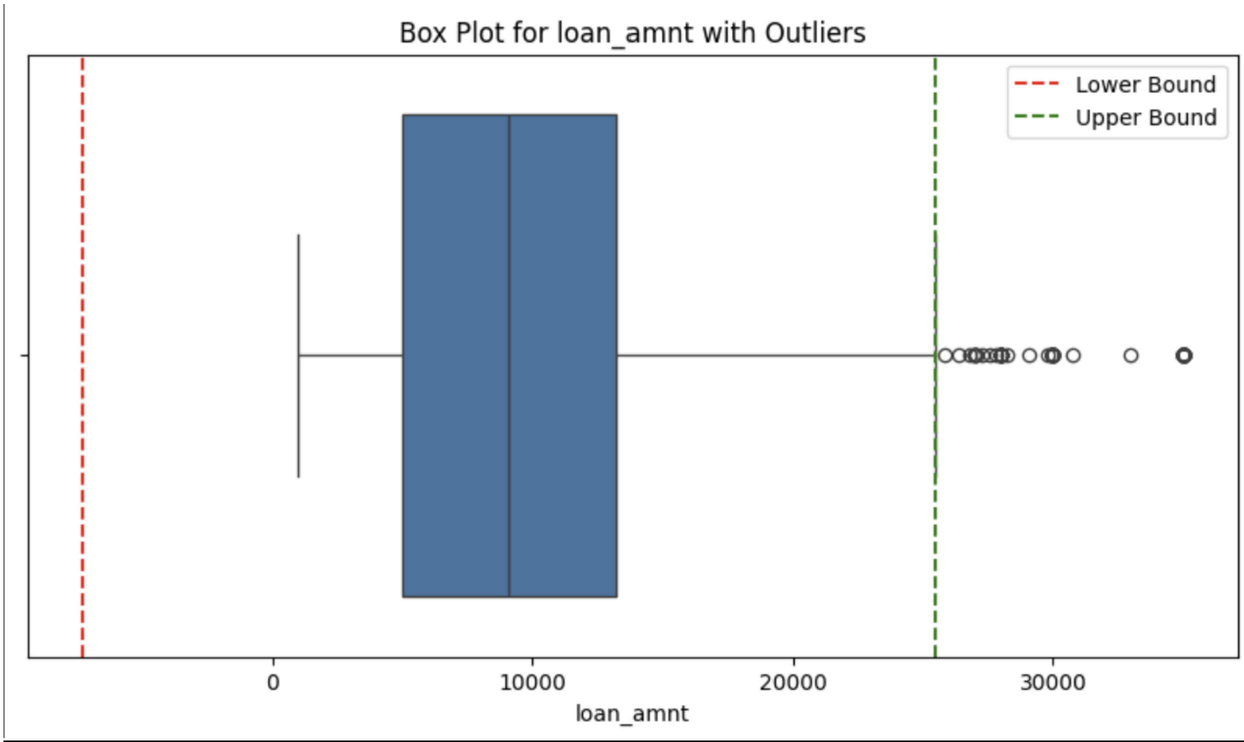

Box Plot for person_income with Outliers
Lower Bound
Upper Bound
0
100000
200000
300000
400000
500000
600000
700000
800000
person_income

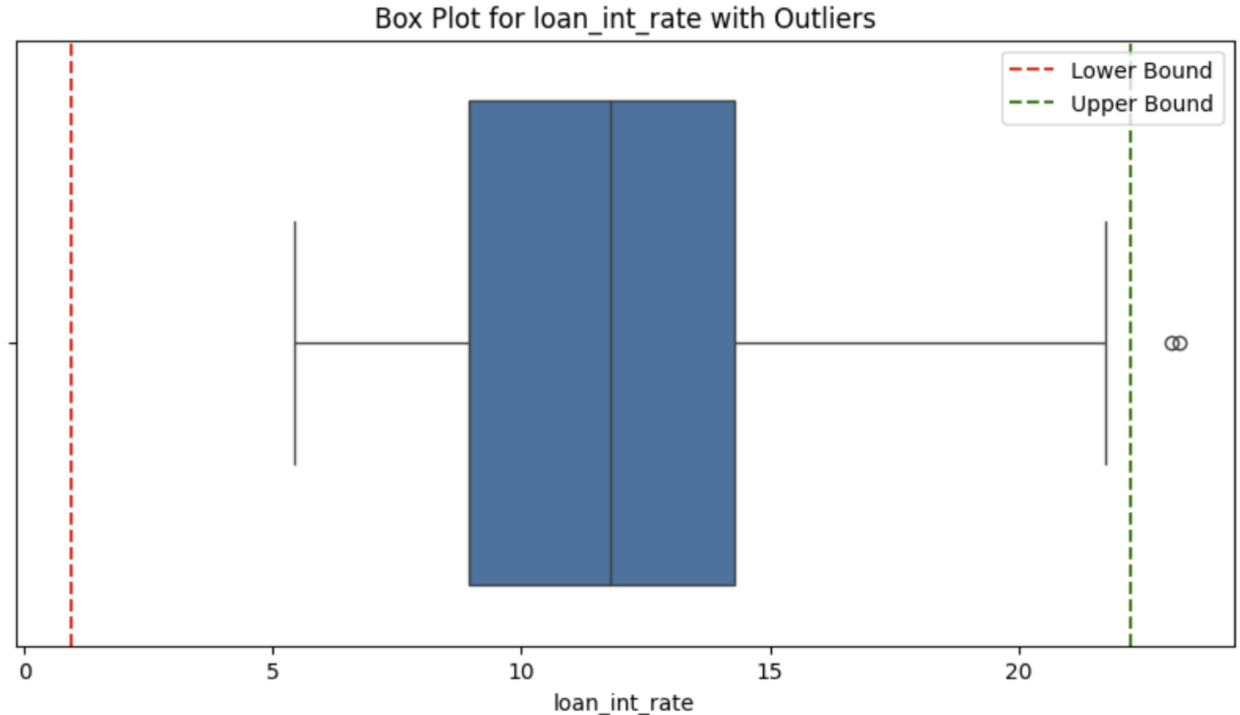

Box Plot for person_home_ownership with Outliers
Lower Bound
Upper Bound
−4
−2
0
2
4
6
8
person_home_ownership

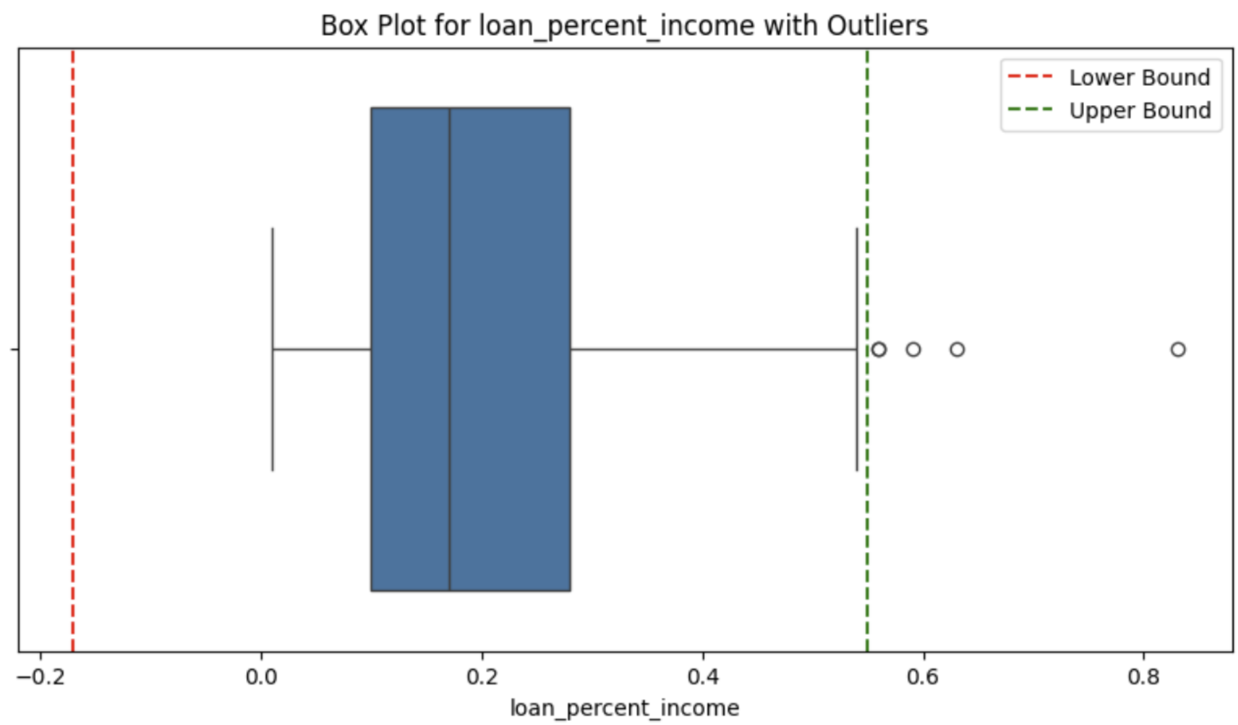

Box Plot for person_emp_length with Outliers
Lower Bound
Upper Bound
0
20
40
60
80
100
120
person_emp_length

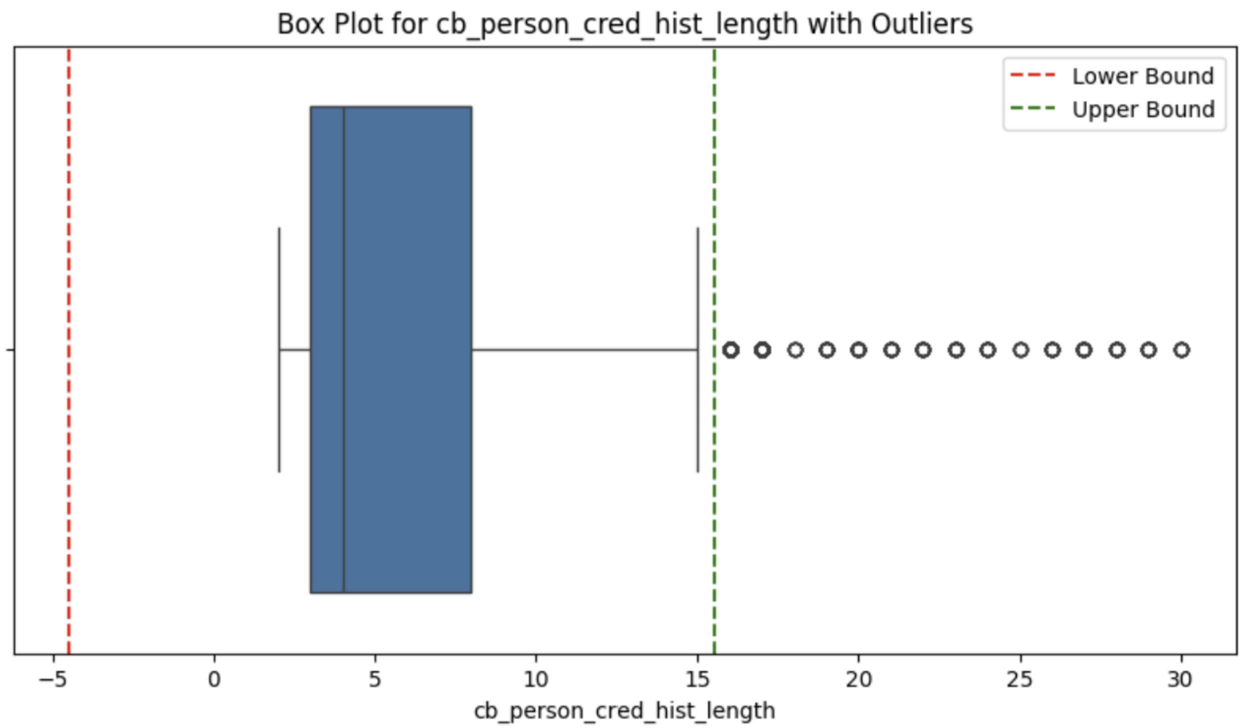

Box Plot for loan_intent with Outliers
Lower Bound
Upper Bound
−4
−2
0
2
4
6
8
loan_intent

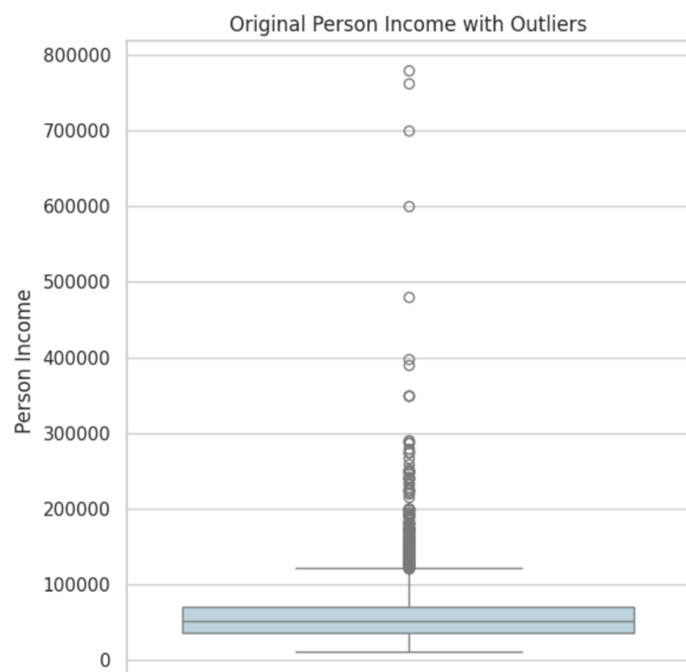

Box Plot for loan_grade with Outliers
Lower Bound
Upper Bound
−2
0
2
4
6
loan_grade


Box Plot for loan_amnt with Outliers
Lower Bound
Upper Bound
0
10000
20000
30000
loan_amnt


Box Plot for loan_int_rate with Outliers
Lower Bound
Upper Bound
0
5
10
15
20
loan_int_rate


Box Plot for loan_percent_income with Outliers
Lower Bound
Upper Bound
−0.2
0.0
0.2
0.4
0.6
0.8
loan_percent_income


Box Plot for cb_person_cred_hist_length with Outliers
Lower Bound
Upper Bound
−5
0
5
10
15
20
25
30
cb_person_cred_hist_length

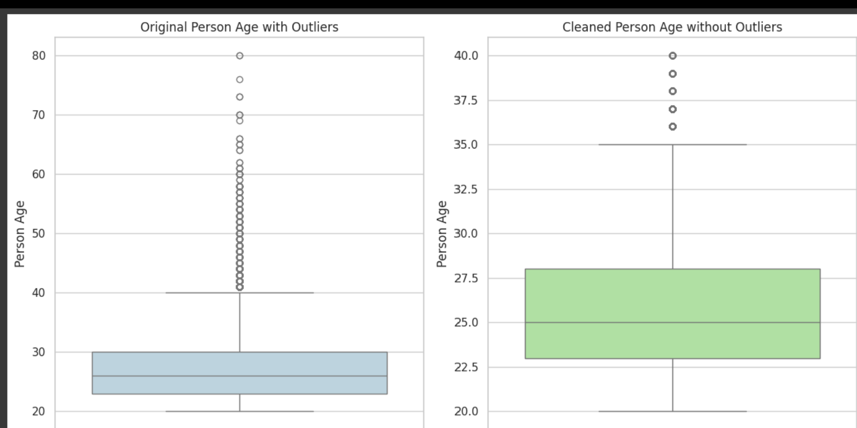

Original Person Income with Outliers
800000
700000
600000
500000
400000
300000
200000
100000
0
Person Income

## Remove Outliers

```
[51]
import pandas as pd
import numpy as np

# Assuming df is your DataFrame
# 1. Check for outliers using the IQR method
Q1 = df.describe().loc['25%']
Q3 = df.describe().loc['75%']
IQR = Q3 - Q1

# Define outlier bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# 2. Remove outliers
# Keep only the rows that are not outliers
df_cleaned = df[~((df < lower_bound) | (df > upper_bound)).any(axis=1)]

# Display the shape of the original and cleaned DataFrame
print(f"Original shape: {df.shape}")
print(f"Cleaned shape: {df_cleaned.shape}")
```

```
Original shape: (16700, 12)
Cleaned shape: (11540, 12)
```

```
[54]
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for person_age
plt.subplot(1, 2, 1)
sns.boxplot(y=df['person_age'], color='lightblue')
plt.title("Original Person Age with Outliers")
plt.ylabel("Person Age")

# Box plot for cleaned DataFrame for person_age
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['person_age'], color='lightgreen')
plt.title("Cleaned Person Age without Outliers")
plt.ylabel("Person Age")

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

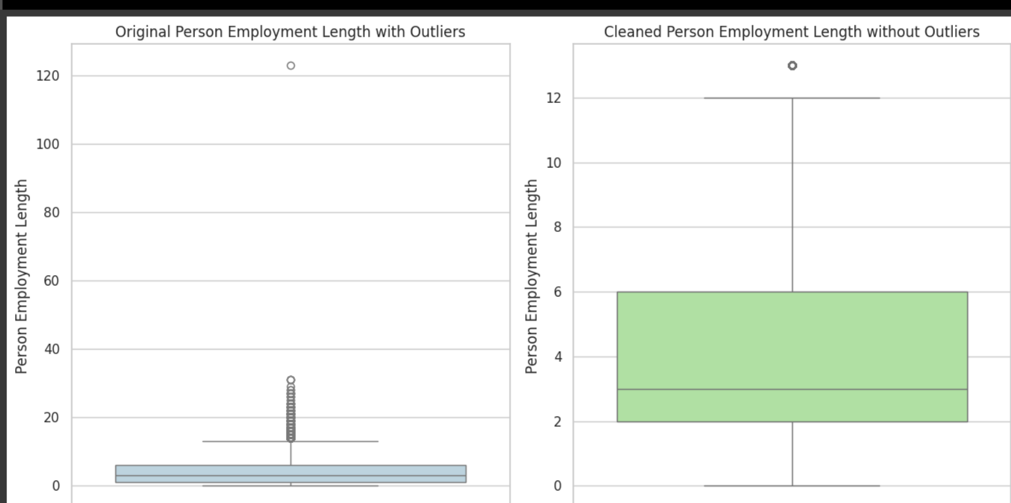



- The left plot ("Original Person Age with Outliers") shows the data before removing outliers, with several data points outside the typical range, especially above 40 years.
- The right plot ("Cleaned Person Age without Outliers") displays the data after outlier removal, resulting in a tighter interquartile range (IQR) and fewer extreme values.

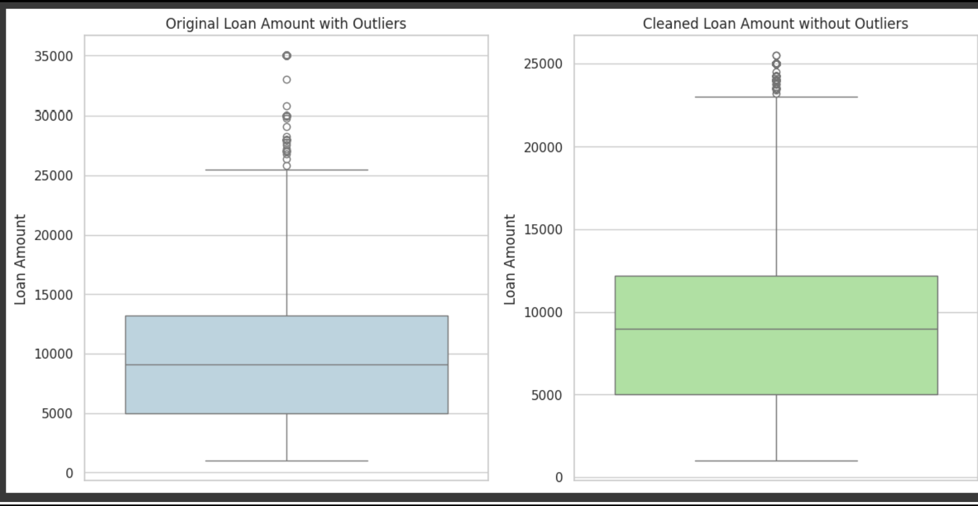
The right plot should ideally have removed the most significant outliers while retaining the core distribution of the data. If this is what you intended, then the visualization effectively demonstrates the cleaned distribution of ages.

```python
import matplotlib.pyplot as plt
import seaborn as sns

# Set the style for the plots
sns.set(style="whitegrid")

# Create a figure
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for person_income
plt.subplot(1, 2, 1)
sns.boxplot(y=df['person_income'], color='lightblue')
plt.title("Original Person Income with Outliers")
plt.ylabel("Person Income")

# Box plot for cleaned DataFrame for person_income
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['person_income'], color='lightgreen')
plt.title("Cleaned Person Income without Outliers")
plt.ylabel("Person Income")

# Show the plots
plt.tight_layout()
plt.show()
```

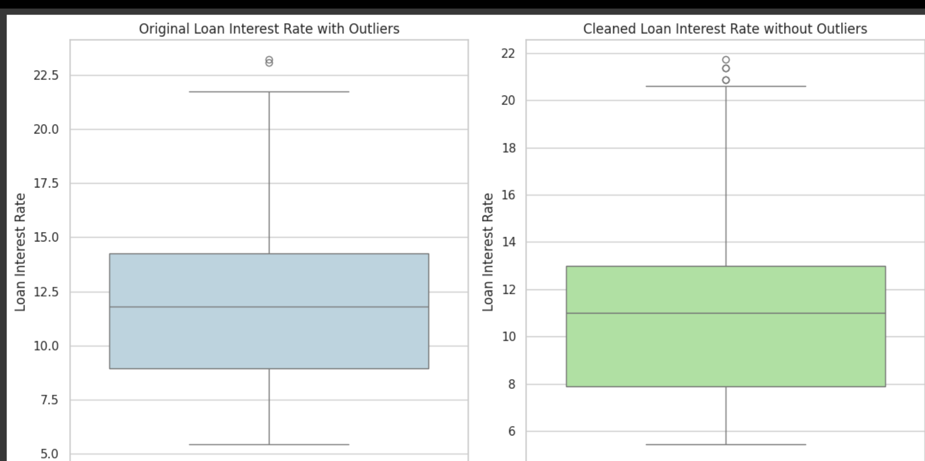

```python
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for person_emp_length
plt.subplot(1, 2, 1)
sns.boxplot(y=df['person_emp_length'], color='lightblue')
plt.title("Original Person Employment Length with Outliers")
plt.ylabel("Person Employment Length")

# Box plot for cleaned DataFrame for person_emp_length
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['person_emp_length'], color='lightgreen')
plt.title("Cleaned Person Employment Length without Outliers")
plt.ylabel("Person Employment Length")

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



[56]
```
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for loan_amnt
plt.subplot(1, 2, 1)
sns.boxplot(y=df['loan_amnt'], color='lightblue')
plt.title("Original Loan Amount with Outliers")
plt.ylabel("Loan Amount")

# Box plot for cleaned DataFrame for loan_amnt
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['loan_amnt'], color='lightgreen')
plt.title("Cleaned Loan Amount without Outliers")
plt.ylabel("Loan Amount")

plt.tight_layout()
plt.show()

```

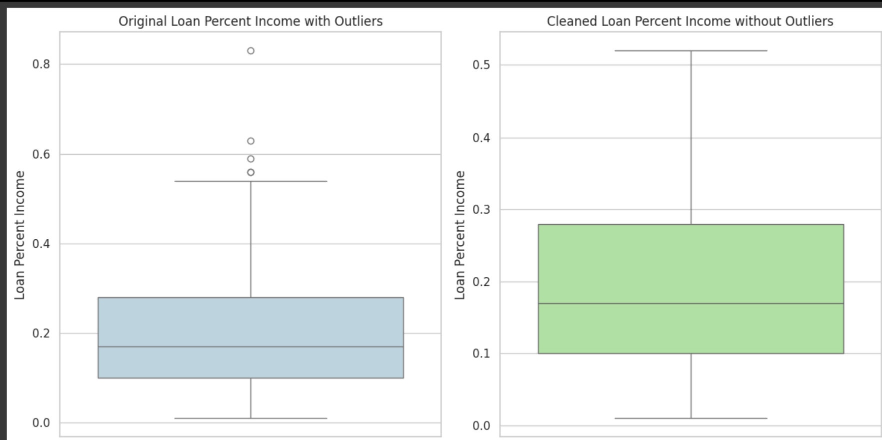



[57]
```
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for loan_int_rate
plt.subplot(1, 2, 1)
sns.boxplot(y=df['loan_int_rate'], color='lightblue')
plt.title("Original Loan Interest Rate with Outliers")
plt.ylabel("Loan Interest Rate")

# Box plot for cleaned DataFrame for loan_int_rate
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['loan_int_rate'], color='lightgreen')
plt.title("Cleaned Loan Interest Rate without Outliers")
plt.ylabel("Loan Interest Rate")

plt.tight_layout()
plt.show()

```



[58]

```python
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for loan_percent_income
plt.subplot(1, 2, 1)
sns.boxplot(y=df['loan_percent_income'], color='lightblue')
plt.title("Original Loan Percent Income with Outliers")
plt.ylabel("Loan Percent Income")

# Box plot for cleaned DataFrame for loan_percent_income
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['loan_percent_income'], color='lightgreen')
plt.title("Cleaned Loan Percent Income without Outliers")
plt.ylabel("Loan Percent Income")

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




[59]

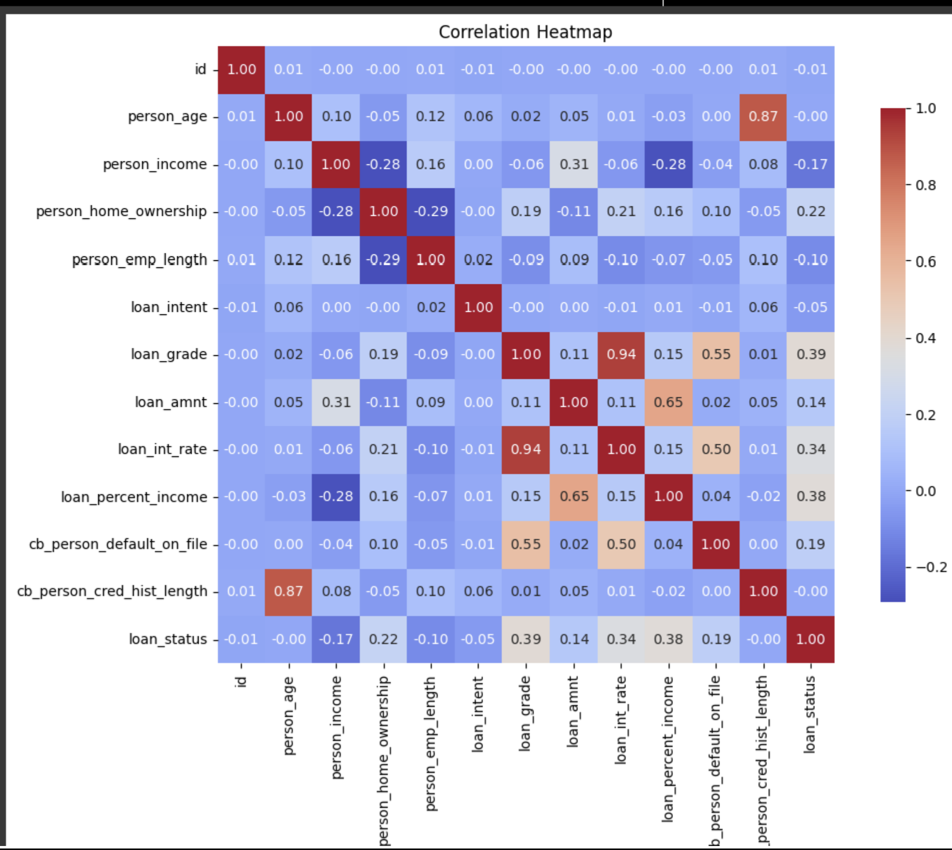
```python
plt.figure(figsize=(12, 6))

# Box plot for original DataFrame for cb_person_cred_hist_length
plt.subplot(1, 2, 1)
sns.boxplot(y=df['cb_person_cred_hist_length'], color='lightblue')
plt.title("Original Credit History Length with Outliers")
plt.ylabel("Credit History Length")

# Box plot for cleaned DataFrame for cb_person_cred_hist_length
plt.subplot(1, 2, 2)
sns.boxplot(y=df_cleaned['cb_person_cred_hist_length'], color='lightgreen')
plt.title("Cleaned Credit History Length without Outliers")
plt.ylabel("Credit History Length")

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

Divide the dataset into target (X) and features (y)

```python
from sklearn.model_selection import train_test_split

X = data_to_train.drop('loan_status', axis=1)  # Replace 'target_column' with your actual target column name
y = data_to_train['loan_status']  # Replace 'target_column' with your actual target column name

# Split the dataset into training and testing sets (70% train, 30% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

```

# Feature Selection

```python
import seaborn as sns
import matplotlib.pyplot as plt

# Calculate the correlation matrix
correlation_matrix = df.corr()

# Set up the matplotlib figure
plt.figure(figsize=(12, 8))

# Generate a heatmap
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, cbar_kws={"shrink": .8})

# Set title and labels
plt.title('Correlation Heatmap')
plt.show()
```

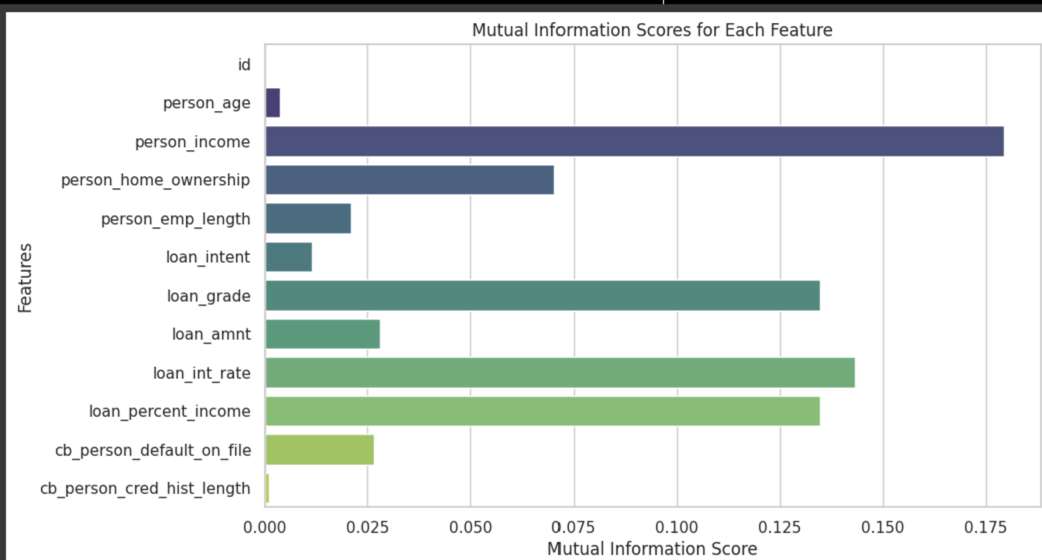


# Miscore

```python
from sklearn.feature_selection import mutual_info_classif, mutual_info_regression

# Use mutual_info_classif for classification tasks or mutual_info_regression for regression tasks
# Replace X and y with your actual feature matrix and target variable

# For classification tasks
mi_score = mutual_info_classif(X, y, random_state=0)

# For regression tasks
# mi_score = mutual_info_regression(X, y, random_state=0)

# Display MI scores
print(mi_score)

```

```
[0.         0.00373506 0.17936325 0.07044917 0.02101987 0.01156984
 0.13482432 0.02798012 0.14331738 0.13466079 0.02672561 0.00098097]
```

```python
[70]
# Assuming X is a DataFrame, get the feature names
feature_names = X.columns if hasattr(X, 'columns') else [f'Feature {i}' for i in range(len(mi_score))]

# Create a bar plot of MI scores
plt.figure(figsize=(10, 6))
sns.barplot(x=mi_score, y=feature_names, palette="viridis")
plt.xlabel('Mutual Information Score')
plt.ylabel('Features')
plt.title('Mutual Information Scores for Each Feature')
plt.show()
```



# Correlation with Target