Importing Libraries

In [1]:
```
import pyforest #for importing all libraries at once
import pygwalker as pyg
import pycaret
```

In [2]:
```
from sklearn.model_selection import cross_val_score,GridSearchCV, RandomizedSearchC
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
#to ignore all the warnings
import warnings
warnings.filterwarnings("ignore")
```

Loading the Dataset and play with the data

In [3]:
```
# Load the dataset
df1 = pd.read_csv("C&T train dataset.csv")
df2 = pd.read_csv("C&T test dataset.csv")
copy=pd.read_csv('C&T test dataset.csv')
#keep a copy of the test beacuse we will be later dropping the sno column but it wi
```

In [4]:
```
#We can get the whole info about our data plus we can perform some visualisations
gwalker = pyg.walk(df1)
```

```
Box(children=(HTML(value='<div id="ifr-pyg-000616d423f52488kOv3qTUAo4wuSg6V" style
="height: auto">\n    <head>…
```

In [5]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 800 entries, 0 to 799
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   sno               800 non-null    int64  
 1   acc_info          800 non-null    object 
 2   duration_month    800 non-null    int64  
 3   credit_history    800 non-null    object 
 4   purpose           800 non-null    object 
 5   savings_acc       800 non-null    object 
 6   employment_st     792 non-null    object 
 7   poi               788 non-null    float64
 8   personal_status   800 non-null    object 
 9   gurantors         792 non-null    object 
 10  resident_since    800 non-null    int64  
 11  property_type     800 non-null    object 
 12  age               796 non-null    float64
 13  installment_type  800 non-null    object 
 14  housing_type      793 non-null    object 
 15  credits_no        800 non-null    int64  
 16  job_type          800 non-null    object 
 17  liables           800 non-null    int64  
 18  telephone         800 non-null    object 
 19  foreigner         800 non-null    object 
 20  Group_no          800 non-null    int64  
dtypes: float64(2), int64(6), object(13)
memory usage: 131.4+ KB
```

EDA and Preprocessing

Handling null values and duplicates

Train dataset

In [6]:
```
df1.duplicated().sum()   #no duplicate values
```

Out[6]: 0

In [7]:
```
# Display the number of missing values in each column
print("Number of missing values in each column:")
print(df1.isnull().sum())

# Numeric Columns: Impute missing values with mean
numeric_columns = ['poi', 'age']
for col in numeric_columns:
    df1[col].fillna(df1[col].mean(), inplace=True)

# Categorical Columns: Impute missing values with mode
categorical_columns = ['employment_st','housing_type','gurantors']
for col in categorical_columns:
```

```python
    df1[col].fillna(df1[col].mode()[0], inplace=True)

# Display the number of missing values after handling
print("\nNumber of missing values after handling:")
print(df1.isnull().sum())
```

```
Number of missing values in each column:
sno                  0
acc_info             0
duration_month       0
credit_history       0
purpose              0
savings_acc          0
employment_st        8
poi                 12
personal_status      0
gurantors            8
resident_since       0
property_type        0
age                  4
installment_type     0
housing_type         7
credits_no           0
job_type             0
liables              0
telephone            0
foreigner            0
Group_no             0
dtype: int64

Number of missing values after handling:
sno                 0
acc_info            0
duration_month      0
credit_history      0
purpose             0
savings_acc         0
employment_st       0
poi                 0
personal_status     0
gurantors           0
resident_since      0
property_type       0
age                 0
installment_type    0
housing_type        0
credits_no          0
job_type            0
liables             0
telephone           0
foreigner           0
Group_no            0
dtype: int64
```

Test Dataset

In [8]:
```
df2.duplicated().sum()   #no duplicate values
```

Out[8]: 0

In [9]:
```
# Display the number of missing values in each column
print("Number of missing values in each column:")
print(df2.isnull().sum())

# Numeric Columns: Impute missing values with mean
numeric_columns = ['poi', 'age']
for col in numeric_columns:
    df2[col].fillna(df2[col].mean(), inplace=True)

# Categorical Columns: Impute missing values with mode
categorical_columns = ['employment_st','housing_type','gurantors']
for col in categorical_columns:
    df2[col].fillna(df2[col].mode()[0], inplace=True)

# Display the number of missing values after handling
print("\nNumber of missing values after handling:")
print(df2.isnull().sum())
```

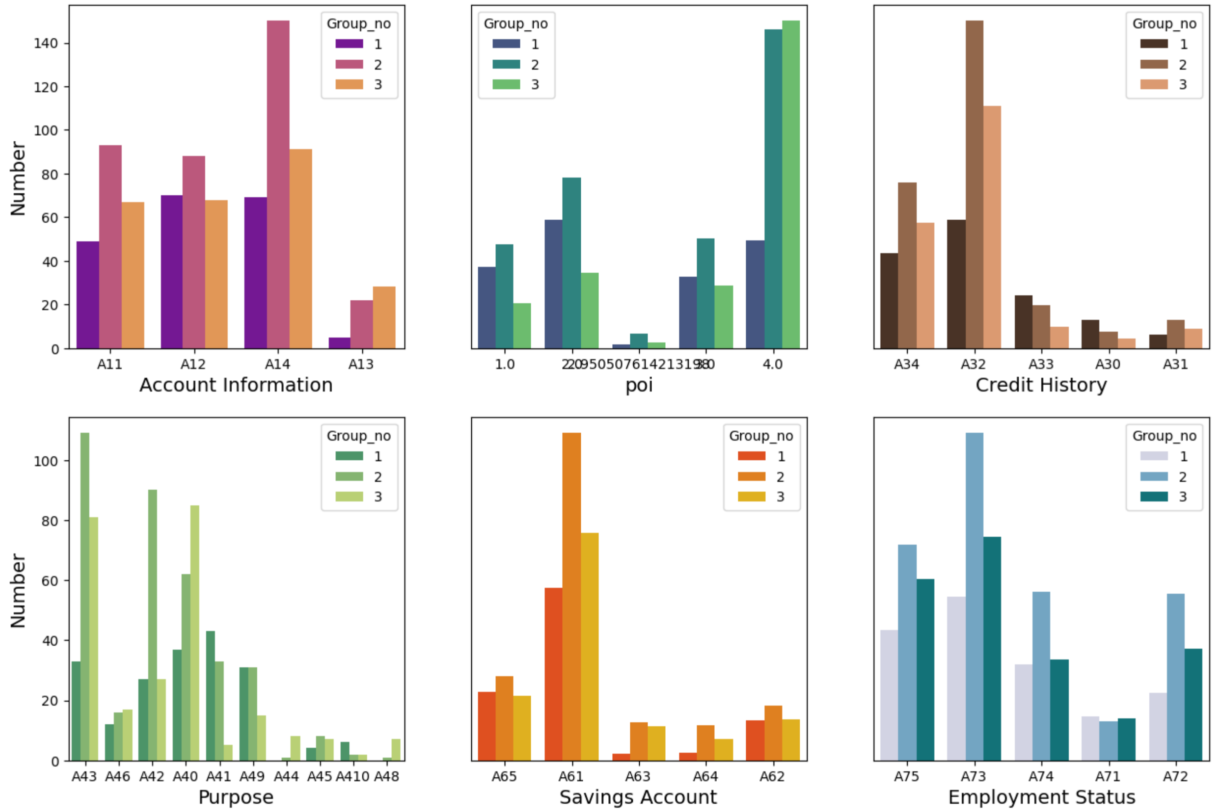
```
Number of missing values in each column:
sno                 0
acc_info            0
duration_month      0
credit_history      0
purpose             0
savings_acc         0
employment_st       5
poi                 5
personal_status     0
gurantors           0
resident_since      0
property_type       0
age                 2
installment_type    0
housing_type        8
credits_no          0
job_type            0
liables             0
telephone           0
foreigner           0
dtype: int64

Number of missing values after handling:
sno                 0
acc_info            0
duration_month      0
credit_history      0
purpose             0
savings_acc         0
employment_st       0
poi                 0
personal_status     0
gurantors           0
resident_since      0
property_type       0
age                 0
installment_type    0
housing_type        0
credits_no          0
job_type            0
liables             0
telephone           0
foreigner           0
dtype: int64
```

Data Visualisation

In [10]:
```
#Deleting the columns which are not necessary for our classification problem
#df1=df1.drop(['sno','telephone','personal_status'],axis=1)
#df2=df2.drop(['sno','telephone','personal_status'],axis=1)
```

In [11]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.figure(figsize=(15,10))

plt.subplot(2,3,1)
sns.countplot(x='acc_info', hue='Group_no', data=df1, palette='plasma')
plt.xlabel('Account Information', fontsize=14)
plt.ylabel('Number', fontsize=14)

plt.subplot(2,3,2)
sns.countplot(x='poi', hue='Group_no', data=df1, palette='viridis')
plt.xlabel('poi', fontsize=14)
plt.ylabel(' ')
plt.yticks([])

plt.subplot(2,3,3)
sns.countplot(x='credit_history', hue='Group_no', data=df1, palette='copper')
plt.xlabel('Credit History', fontsize=14)
plt.ylabel(' ')
plt.yticks([])

plt.subplot(2,3,4)
sns.countplot(x='purpose', hue='Group_no', data=df1, palette='summer')
plt.xlabel('Purpose', fontsize=14)
plt.ylabel('Number', fontsize=14)

plt.subplot(2,3,5)
sns.countplot(x='savings_acc', hue='Group_no', data=df1, palette='autumn')
plt.xlabel('Savings Account', fontsize=14)
plt.ylabel(' ')
plt.yticks([])

plt.subplot(2,3,6)
sns.countplot(x='employment_st', hue='Group_no', data=df1, palette='PuBuGn')
plt.xlabel('Employment Status', fontsize=14)
plt.ylabel(' ')
plt.yticks([])
plt.show()
```



Univariate Analysis

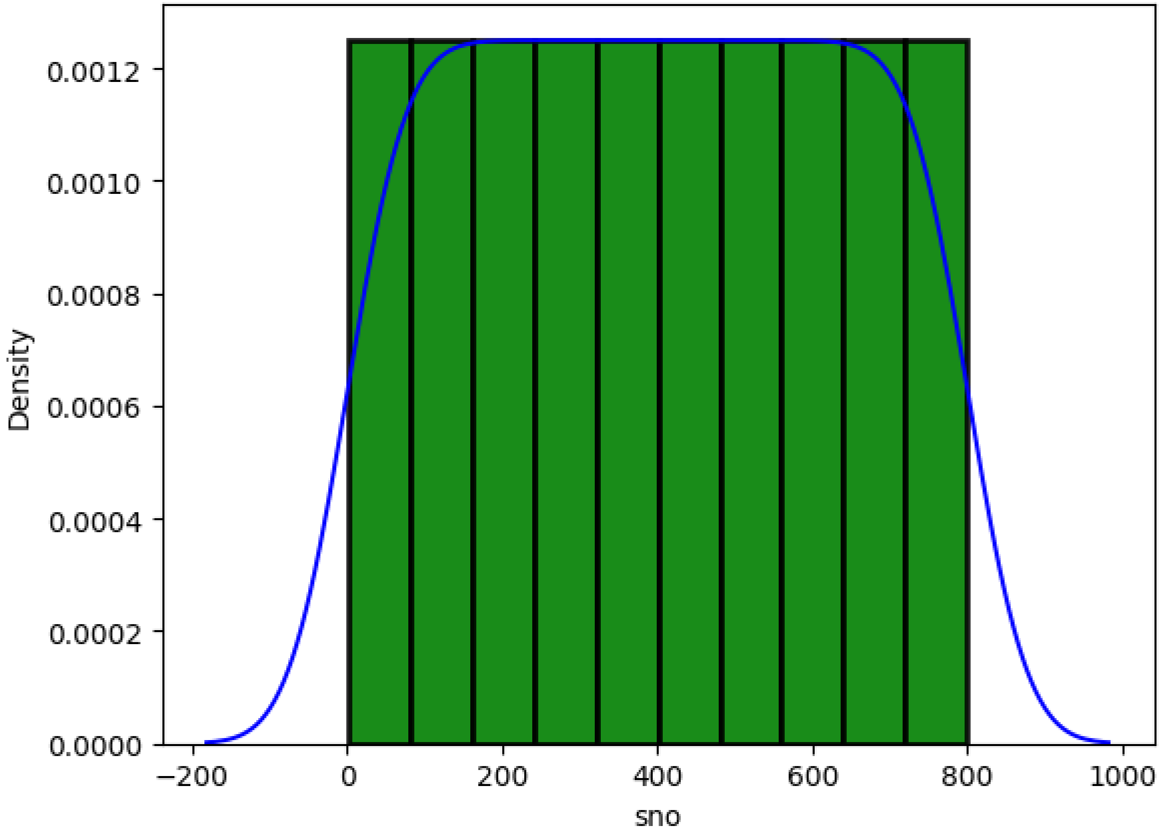
In [12]:
```
# Define categorical and numerical features
categorical_features = ['acc_info', 'credit_history', 'purpose', 'savings_acc', 'em
                        'gurantors', 'property_type', 'installment_type',
                        'housing_type', 'job_type', 'foreigner']
numerical_features = ['duration_month', 'poi', 'resident_since', 'age', 'credits_no
```

In [13]:
```
# Select only numeric columns
df1_numeric = df1.select_dtypes(include=['number'])
df1_categorical = df1.select_dtypes(exclude=['number'])

# Now calculate correlations
#correlation_matrix = df1_numeric.corr()
```

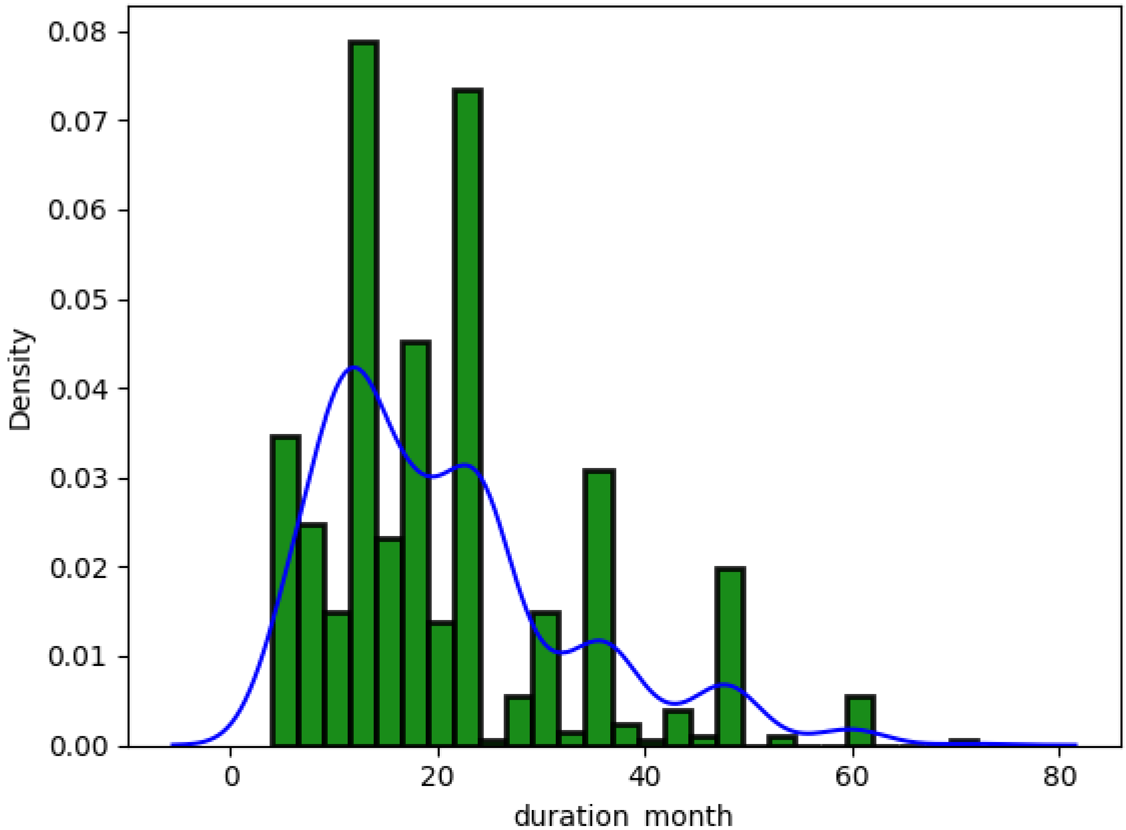
In [14]:
```
from scipy.stats import skew,kurtosis
key =[]
skewval =[]
kurtval =[]
for i  in df1_numeric :
    print('The skewness of the ',i, '=',skew(df1_numeric [i]))
    print('The kurtosis of the ',i, '=',kurtosis(df1_numeric [i]))
    plt.figure()
    sns.distplot(df1_numeric[i],color='blue',kde=True,
                 hist_kws = {'color':'green', 'edgecolor':'black','linewidth':2,'al
    plt.show()
    key.append(i)
```

```
    skewval.append(skew(df1_numeric [i]))
    kurtval.append(kurtosis(df1_numeric [i]))
```

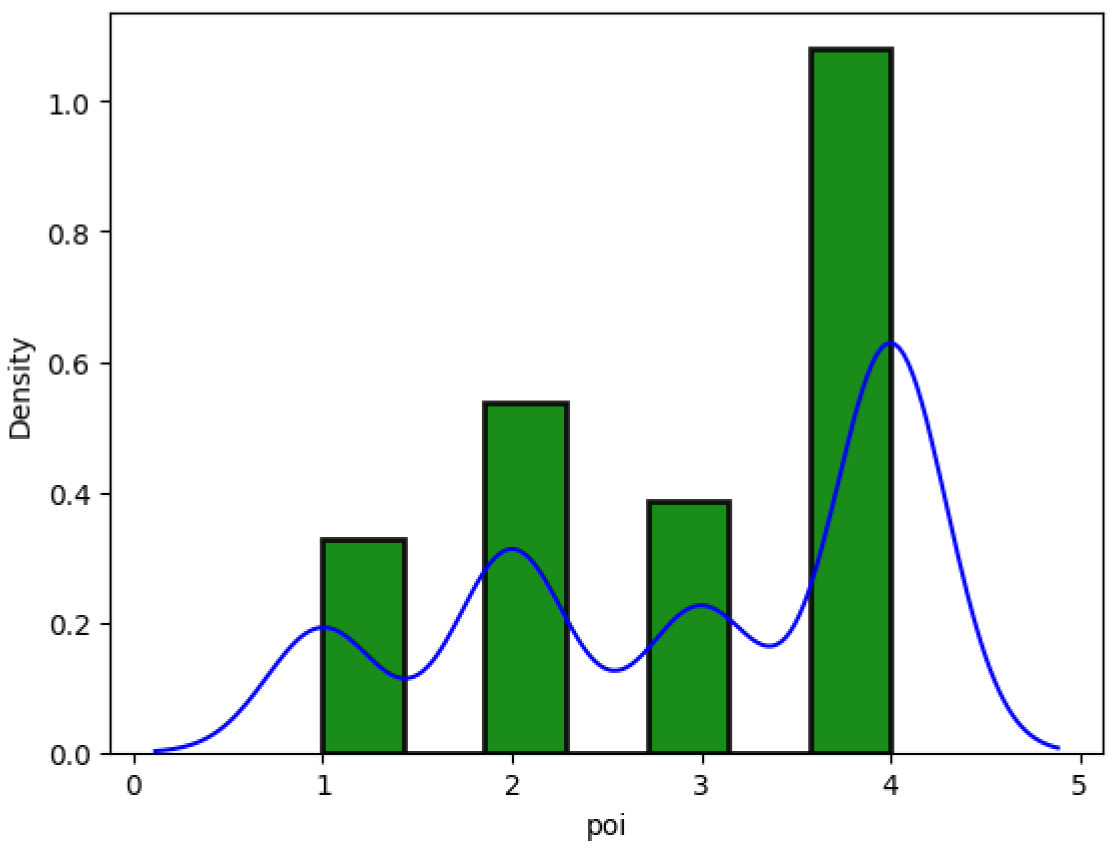
```
The skewness of the  sno = 0.0
The kurtosis of the  sno = -1.2000037500058593
```



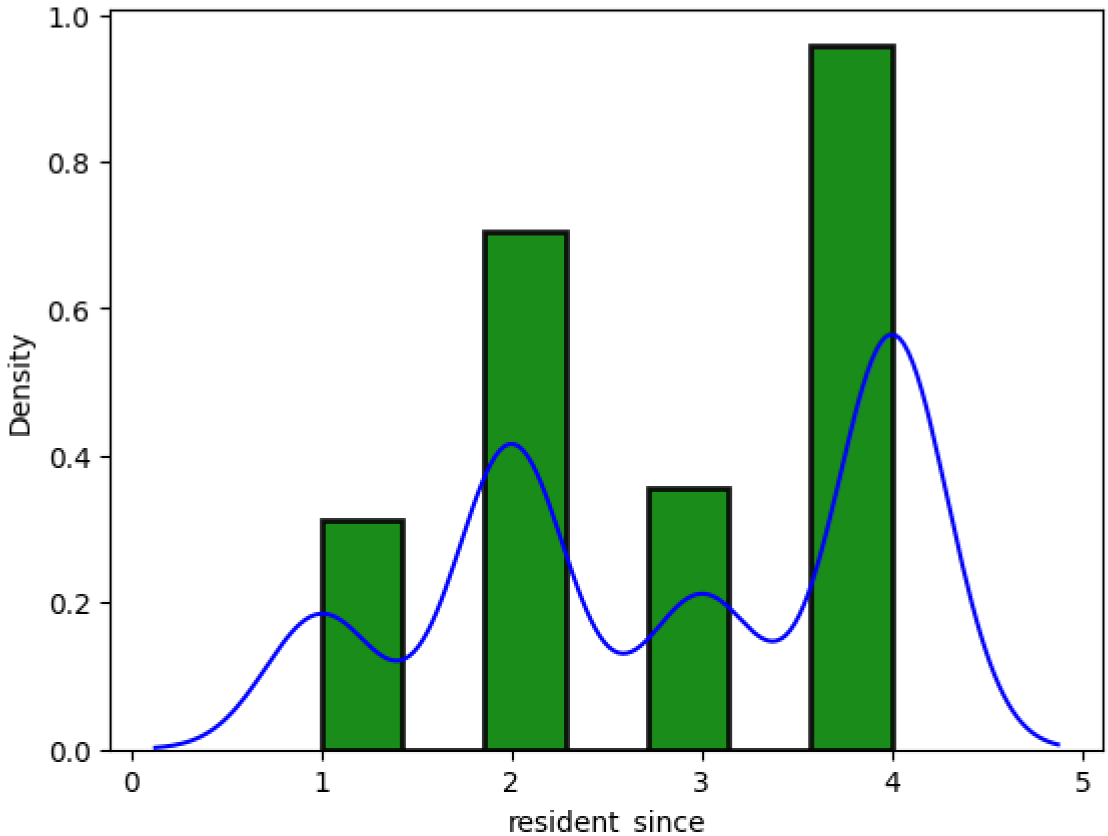

```
The skewness of the  duration_month = 1.1470027029543717
The kurtosis of the  duration_month = 1.0529676149863754
```



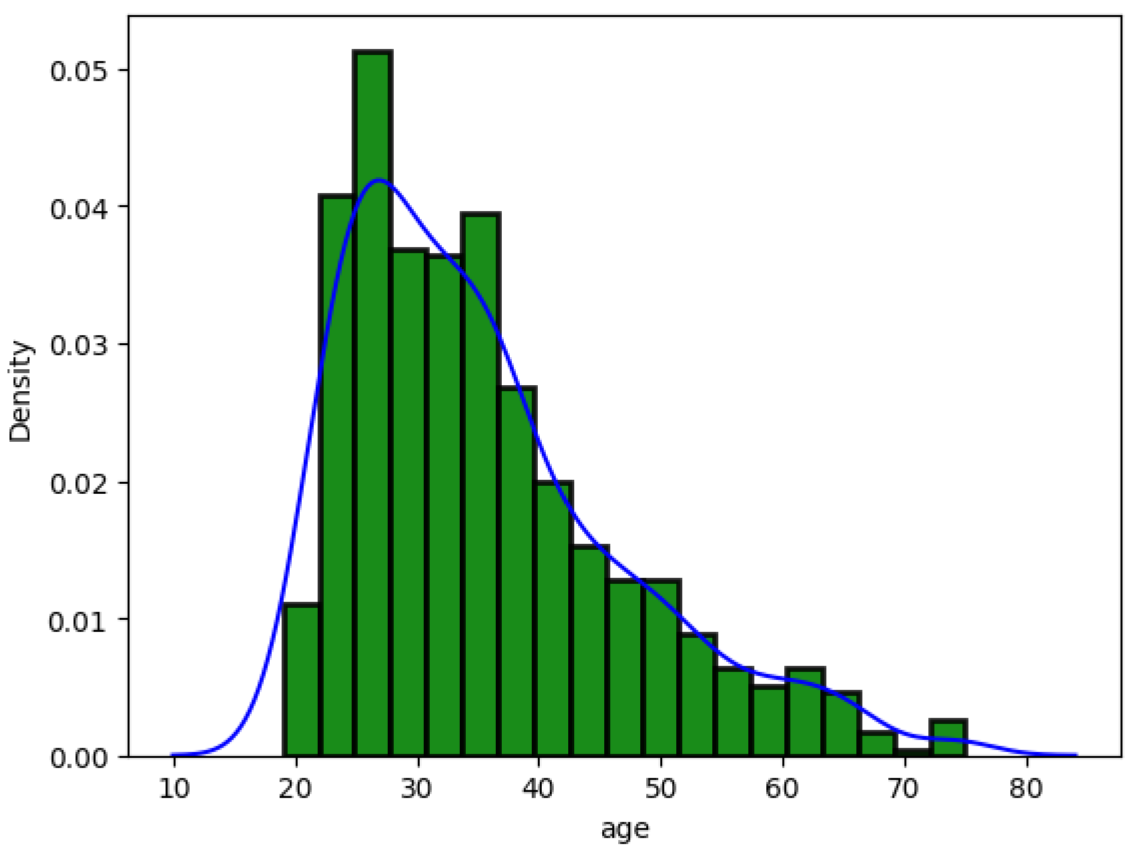

```
The skewness of the  poi = -0.5039066555236439
The kurtosis of the  poi = -1.2293262165867966
```



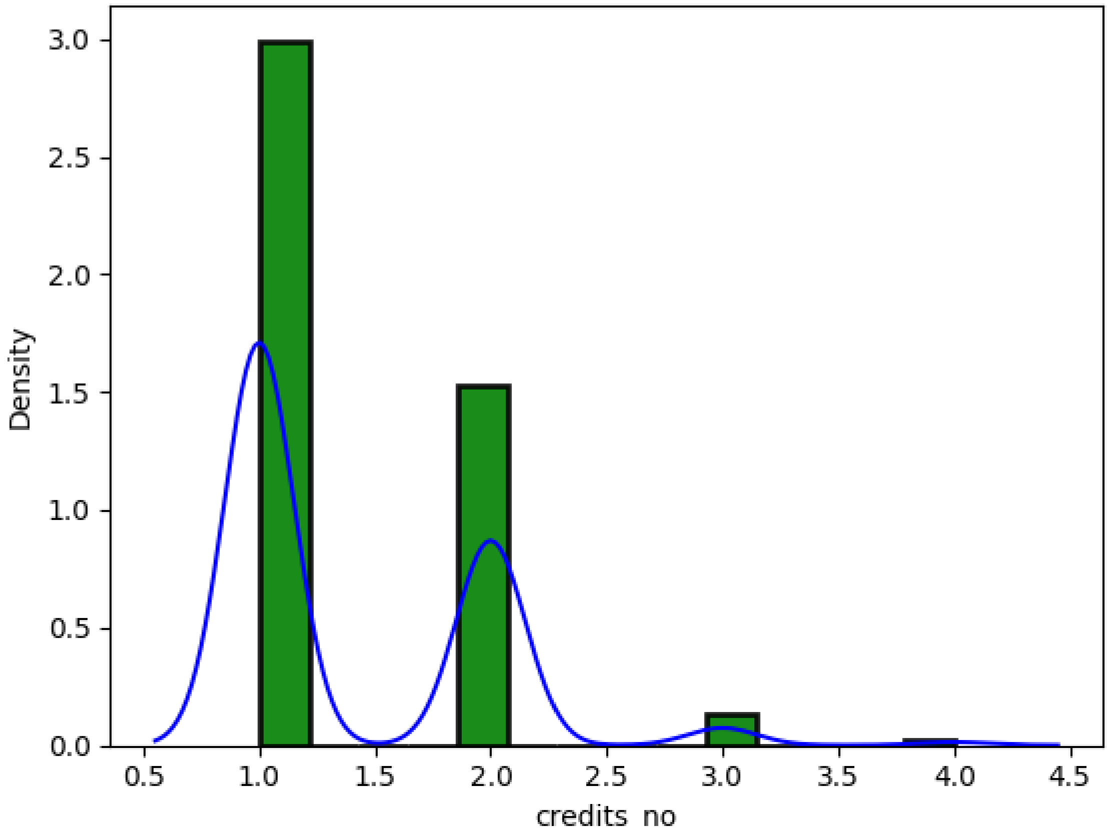

```
The skewness of the  resident_since = -0.2767323024835812
The kurtosis of the  resident_since = -1.3766474938843547
```



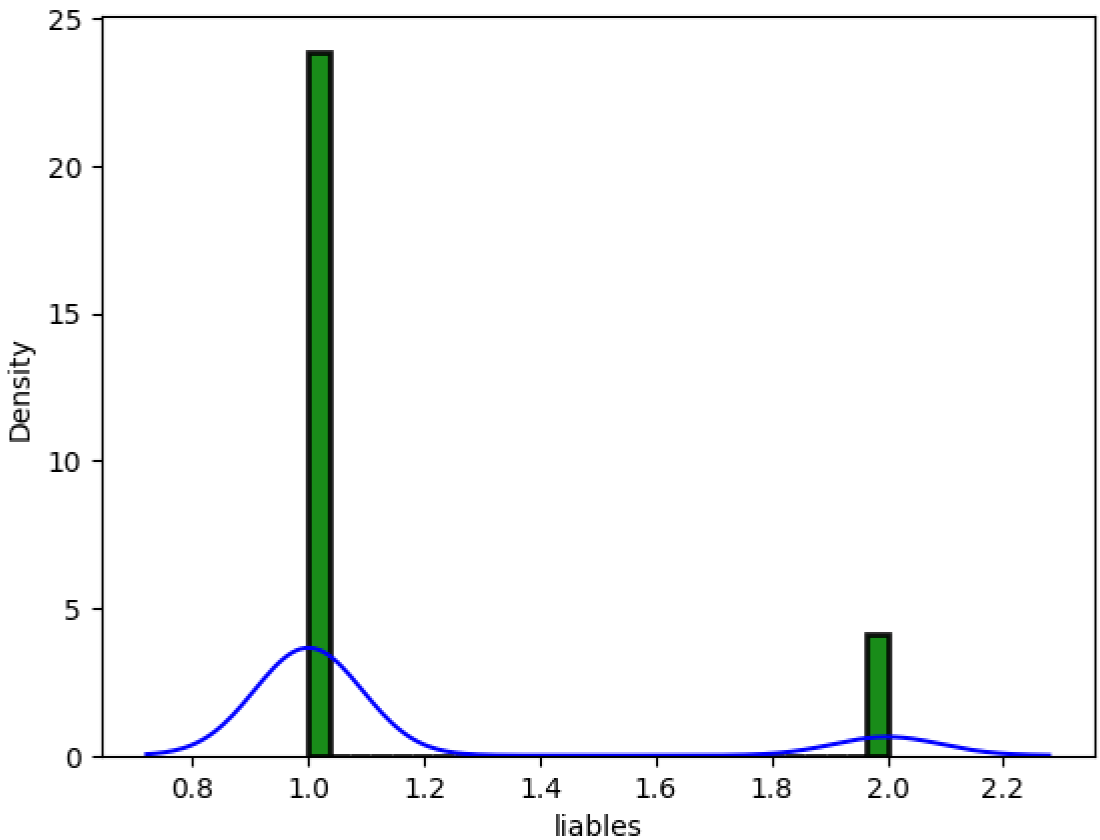

```
The skewness of the  age = 1.060425705742385
The kurtosis of the  age = 0.6927026736563899
```



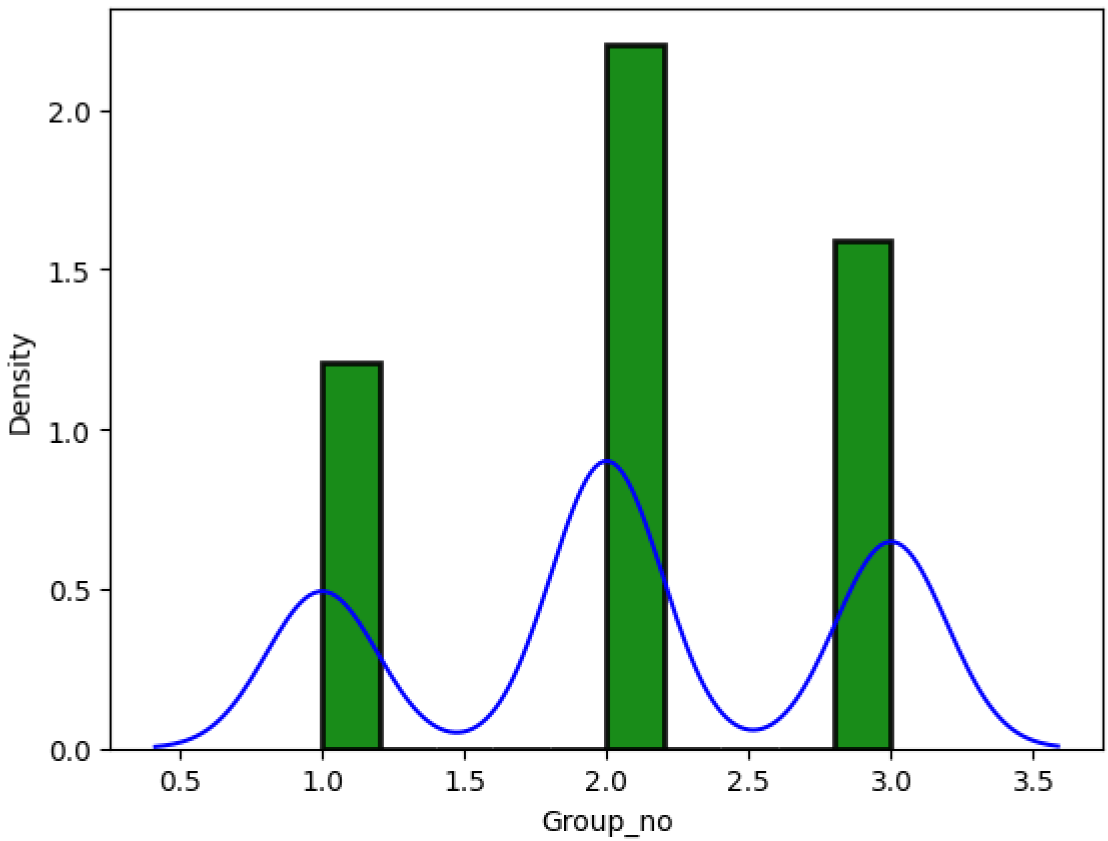

```
The skewness of the  credits_no = 1.2652514728538597
The kurtosis of the  credits_no = 1.4791338504629952
```




```
The skewness of the  liables = 1.9881351927895627
The kurtosis of the  liables = 1.9526815448083923
```




```
The skewness of the  Group_no = -0.12325434930243198
The kurtosis of the  Group_no = -1.1851006072150474
```



In [15]:
```
dict = {'variables':key,'skew_values':skewval,'kurtosis_values':kurtval}
```

In [16]:
```
show_val = pd.DataFrame(dict)
```

In [17]:
```
show_val
```

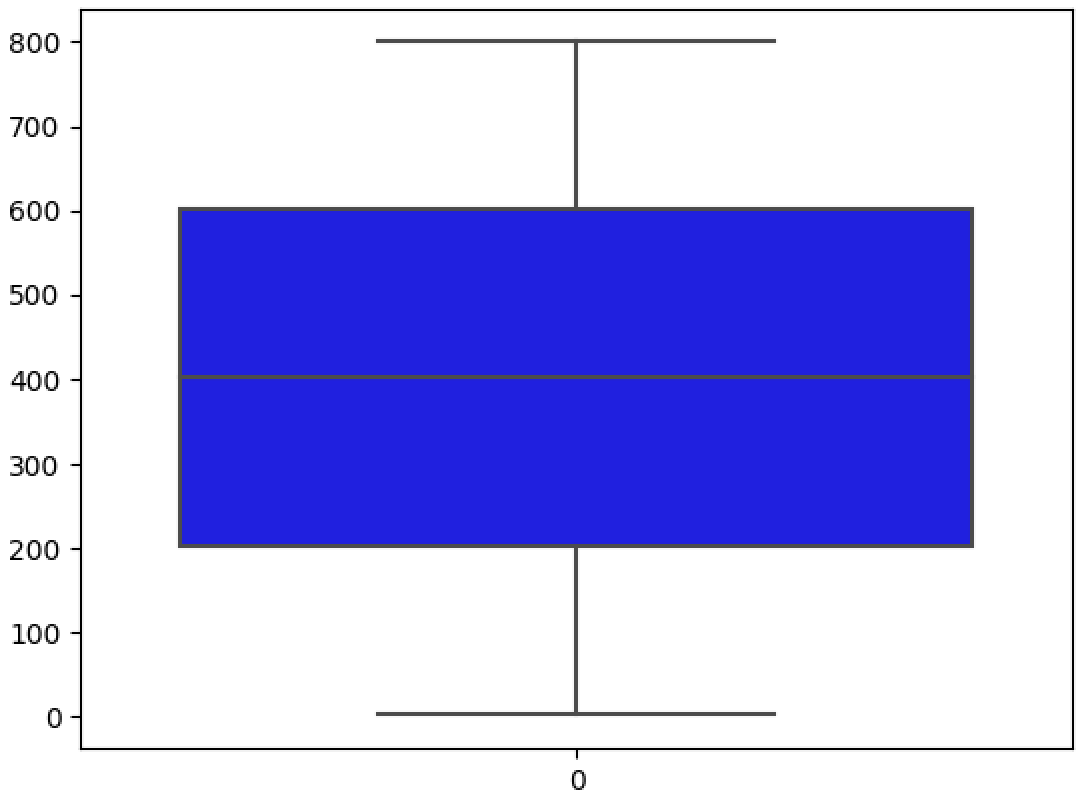
Out[17]:

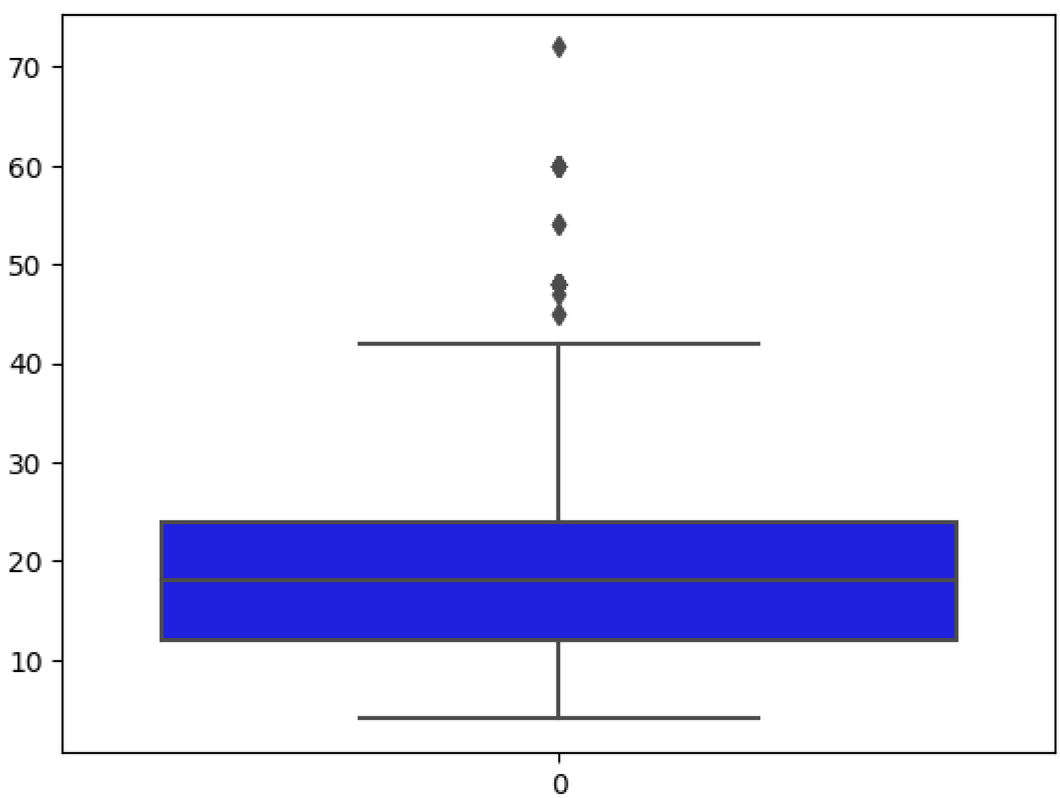
	variables	skew_values	kurtosis_values
0	sno	0.000000	-1.200004
1	duration_month	1.147003	1.052968
2	poi	-0.503907	-1.229326
3	resident_since	-0.276732	-1.376647
4	age	1.060426	0.692703
5	credits_no	1.265251	1.479134
6	liables	1.988135	1.952682
7	Group_no	-0.123254	-1.185101

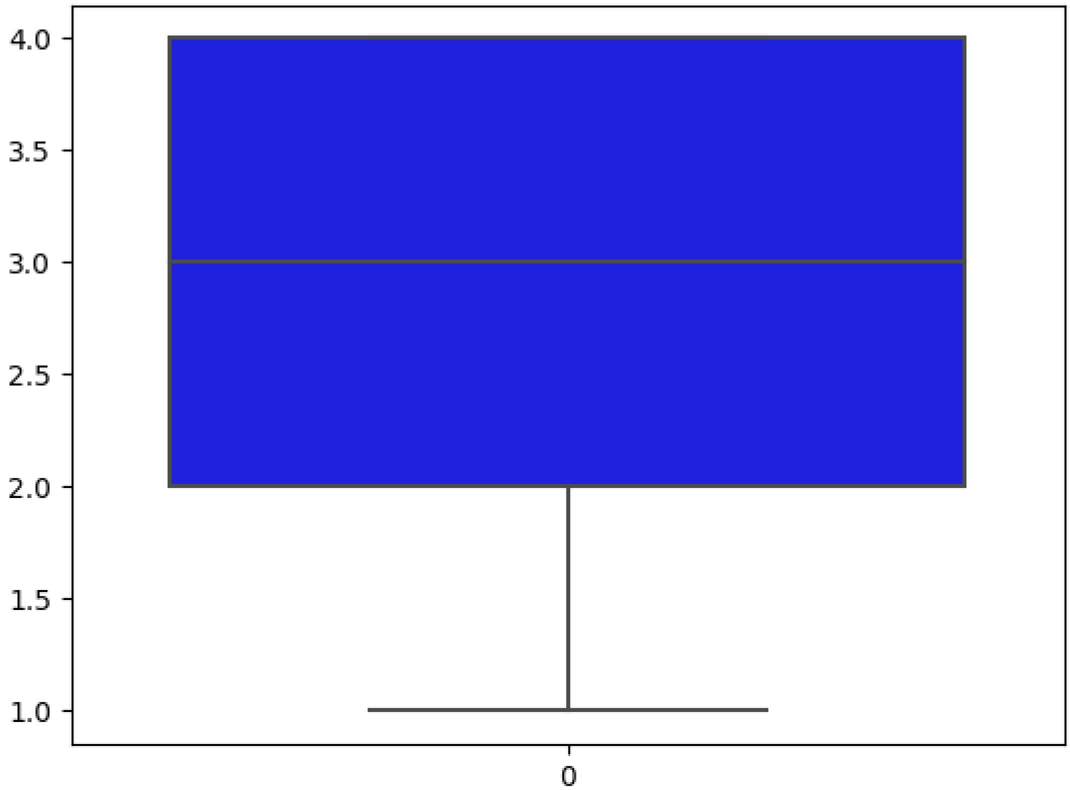
Handling outliers

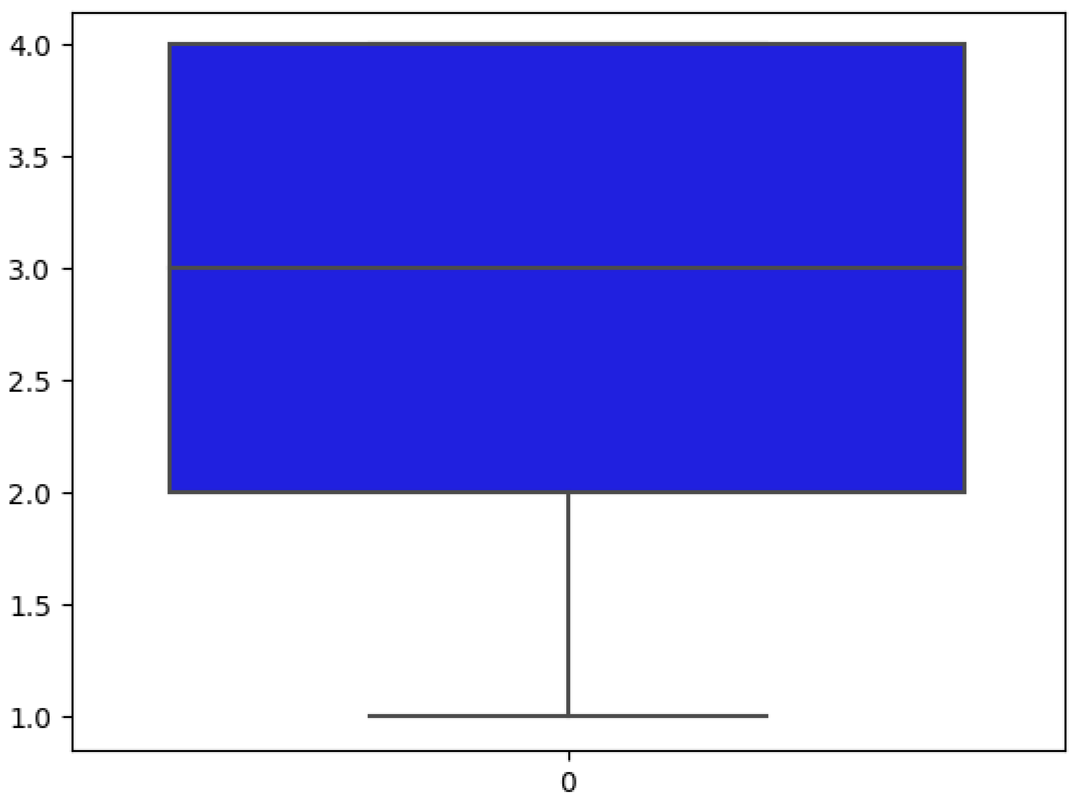
In [18]:
```
for i  in df1_numeric:
    plt.figure()
```

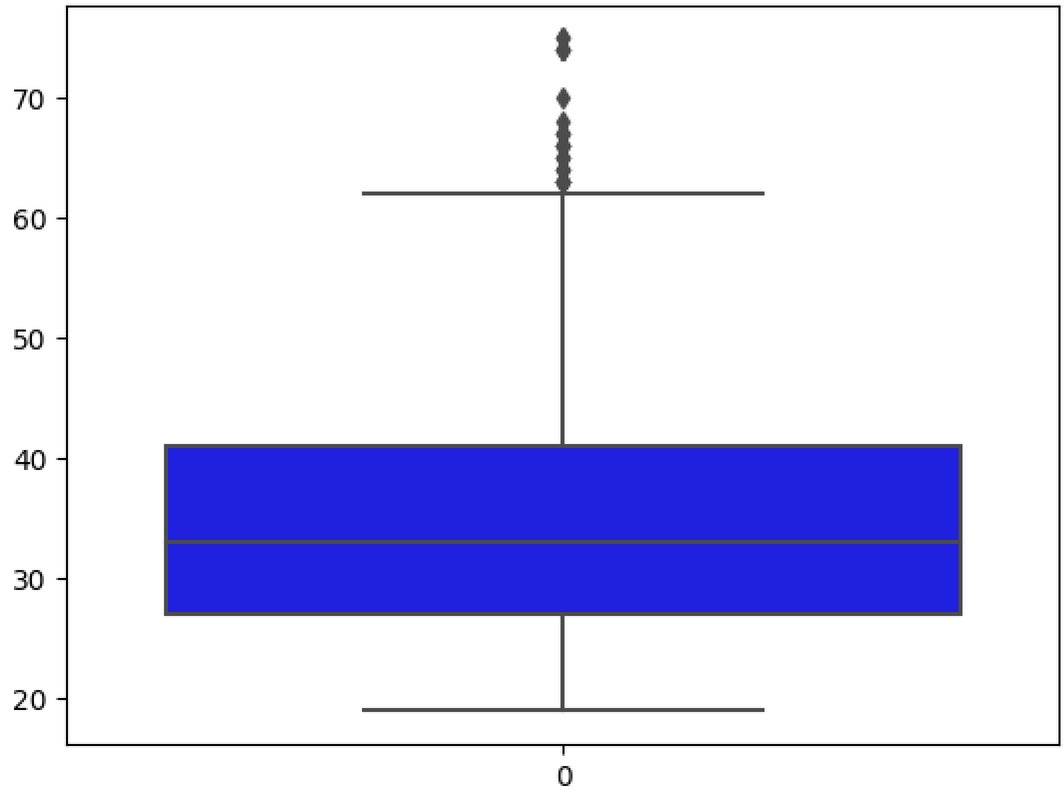
```
    sns.boxplot(df1_numeric[i],color='blue')
    plt.show()
```

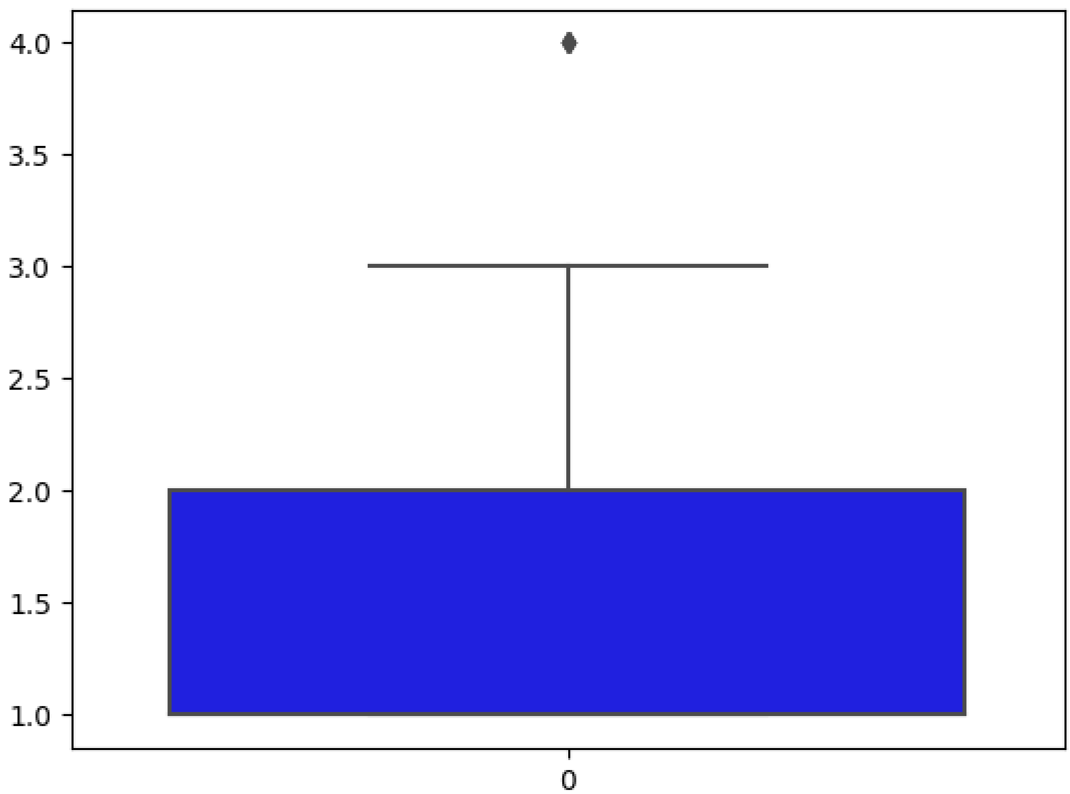


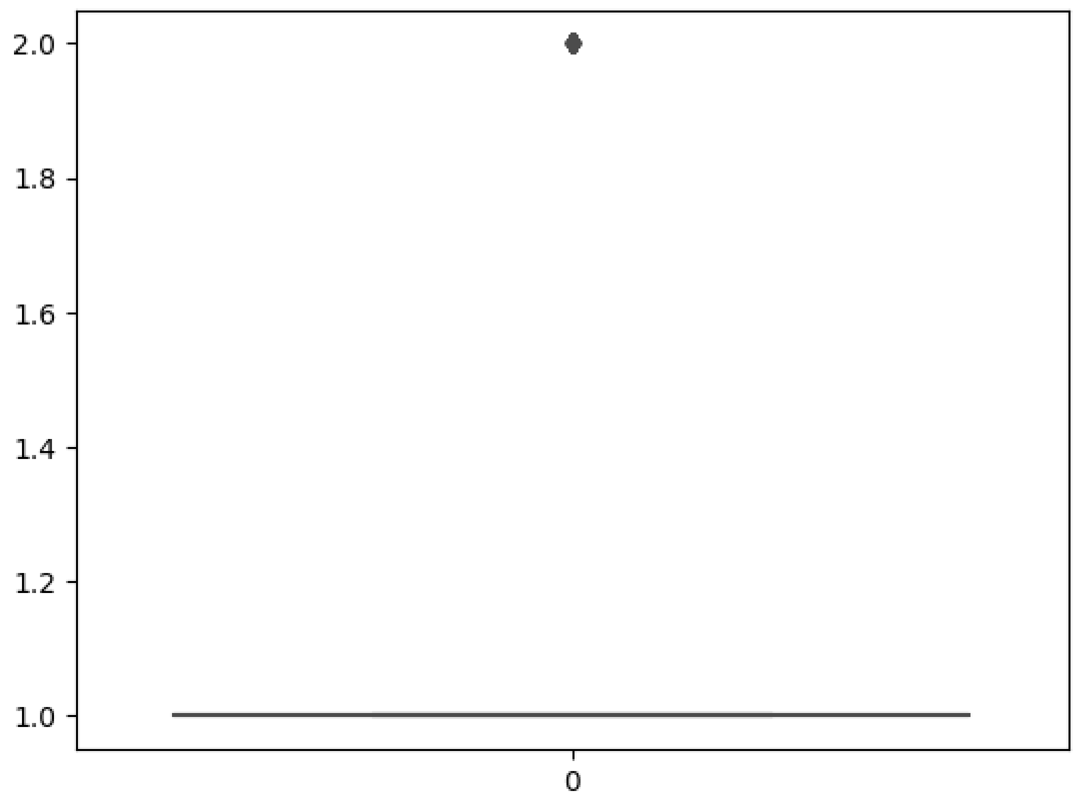


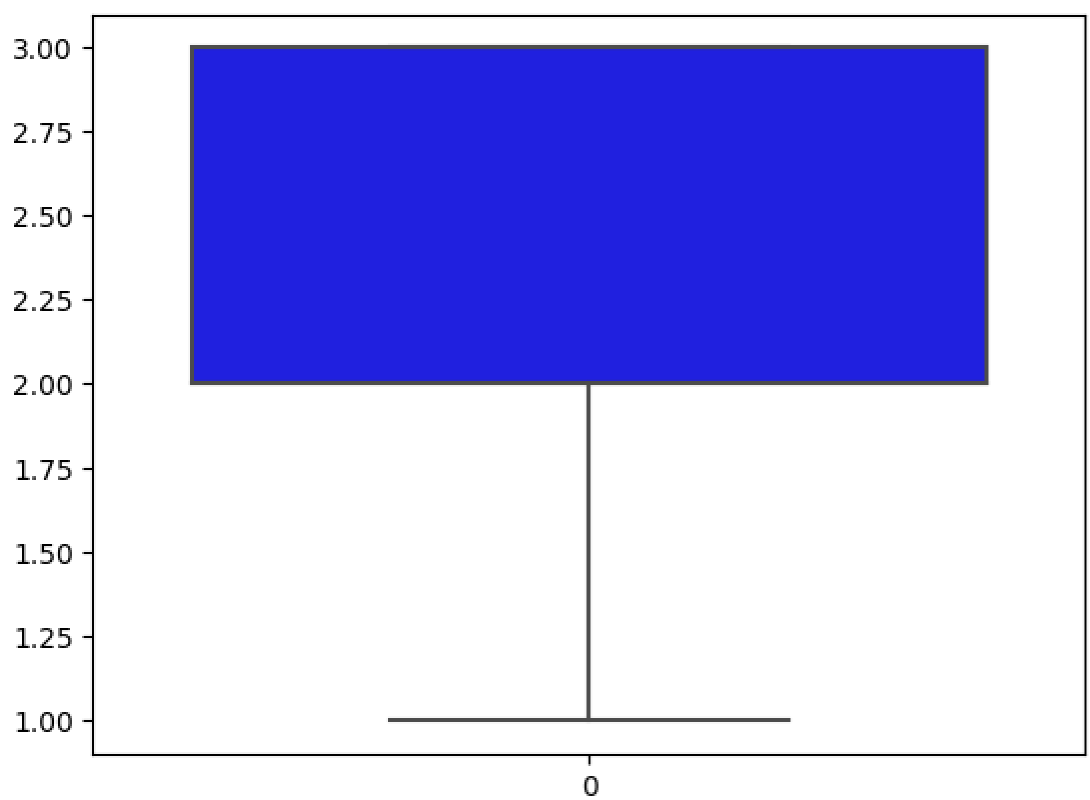

4.0
3.5
3.0
2.5
2.0
1.5
1.0
0


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0


70
60
50
40
30
20
0


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0





In [19]:
```
#Impute the outliers from the data
# Calculate the first quartile (Q1) and third quartile (Q3)
Q1 = df1_numeric.quantile(0.25)
Q3 = df1_numeric.quantile(0.75)

# Calculate the IQR
```

```
IQR = Q3 - Q1

# Define the threshold for outliers (e.g., 1.5 times the IQR)
threshold = 1.5


# Impute outliers with the median
# The clip() method limits the values in the DataFrame to be within the specified l
df1_imputed_median = df1_numeric.clip(lower=Q1 - threshold * IQR, upper=Q3 + thresh
```

In [20]:
```
# Combine imputed numeric columns with original categorical columns
df1_new = pd.concat([df1_imputed_median, df1_categorical], axis=1)
```

In [21]:
```
df1_new
```

Out[21]:

	sno	duration_month	poi	resident_since	age	credits_no	liables	Group_no	acc_inf
0	1	6	4.0	4	62.0	2.0	1	3	A1
1	2	42	2.0	2	22.0	1.0	1	1	A1
2	3	12	2.0	3	49.0	1.0	1	2	A1
3	4	42	2.0	4	45.0	1.0	1	1	A1
4	5	24	3.0	4	53.0	2.0	1	1	A1
...	...	...	...	...	...	...	...	...	.
795	796	9	2.0	4	22.0	1.0	1	2	A1
796	797	18	1.0	4	51.0	1.0	1	1	A1
797	798	12	2.0	4	22.0	2.0	1	3	A1
798	799	24	4.0	4	54.0	2.0	1	3	A1
799	800	9	4.0	2	35.0	1.0	1	2	A1

800 rows × 21 columns

Checking whether our dataset is imbalanced

In [22]:
```
# Assuming 'Group_no' is the target variable
class_distribution = df1_new['Group_no'].value_counts()

print("Class Distribution:")
print(class_distribution)

# Calculate the imbalance ratio
imbalance_ratio = class_distribution.min() / class_distribution.max()

print("Imbalance Ratio:", imbalance_ratio)
```

```
Class Distribution:
Group_no
2    353
3    254
1    193
Name: count, dtype: int64
Imbalance Ratio: 0.546742209631728
```

So our dataset is Balanced

Bivariate Analysis

In [23]:
```
# Now calculate correlations
correlation_matrix = df1_numeric.corr()
```

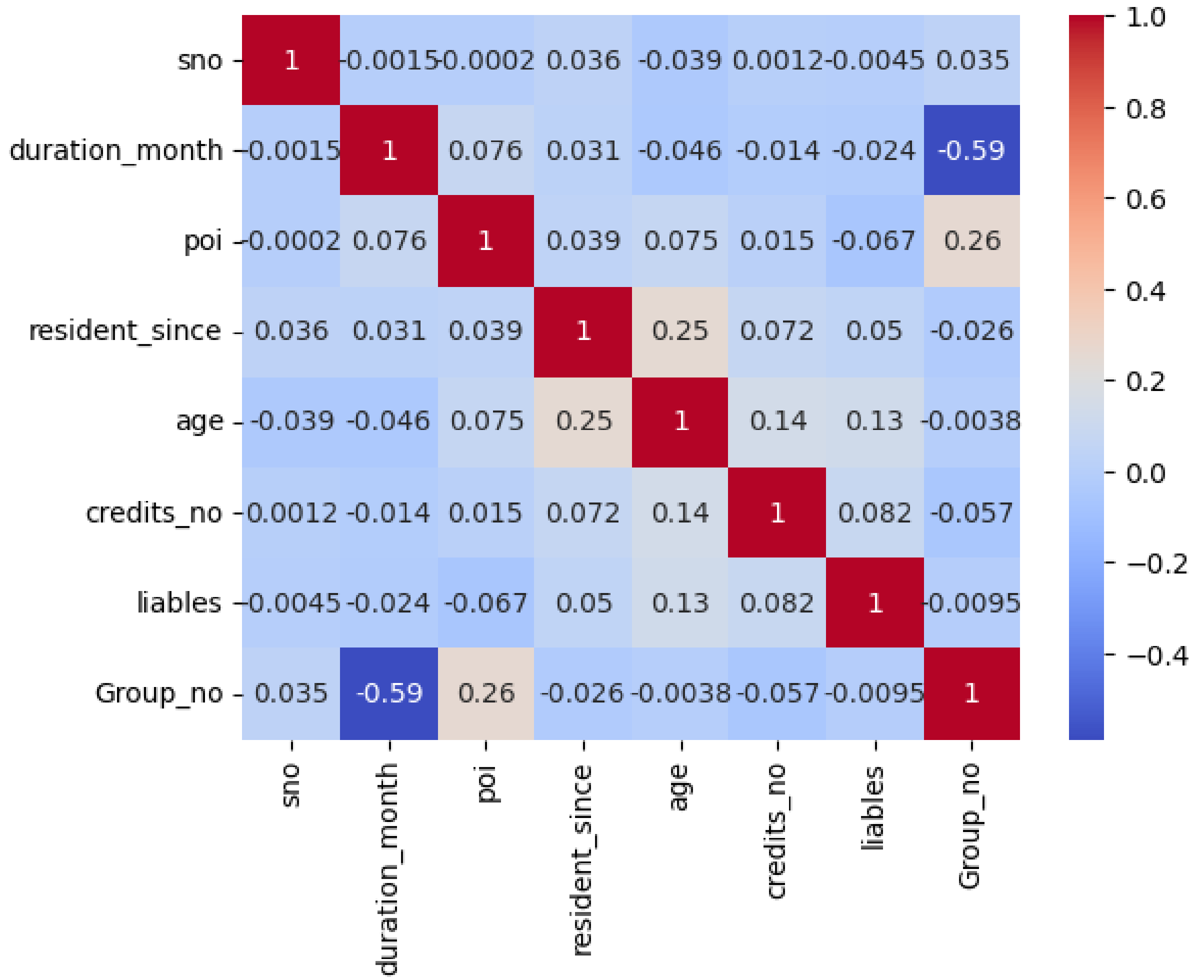
In [24]:
```
correlation_matrix
```

Out[24]:

	sno	duration_month	poi	resident_since	age	credits_nc
sno	1.000000	-0.001513	-0.000196	0.035694	-0.039311	0.00115
duration_month	-0.001513	1.000000	0.076066	0.030762	-0.045680	-0.01381
poi	-0.000196	0.076066	1.000000	0.039182	0.075055	0.01457
resident_since	0.035694	0.030762	0.039182	1.000000	0.251351	0.07208
age	-0.039311	-0.045680	0.075055	0.251351	1.000000	0.13834
credits_no	0.001155	-0.013814	0.014575	0.072087	0.138340	1.00000
liables	-0.004457	-0.023749	-0.066547	0.050136	0.129534	0.08198
Group_no	0.035358	-0.588725	0.259411	-0.026316	-0.003826	-0.05659

In [25]:
```
sns.heatmap(correlation_matrix,annot=True,cmap='coolwarm')
```

Out[25]: <Axes: >



Label encoding

In [26]:

```
# Create a label encoder object
label_encoder = LabelEncoder()

# Iterate over columns and encode categorical columns
for column in df1_new.columns:
    if df1_new[column].dtype == 'object':  # Check if column contains categorical d
        df1_new[column] = label_encoder.fit_transform(df1_new[column])

# Display the encoded DataFrame
df1_new
```

Out[26]:

	sno	duration_month	poi	resident_since	age	credits_no	liables	Group_no	acc_inf
0	1	6	4.0	4	62.0	2.0	1	3	
1	2	42	2.0	2	22.0	1.0	1	1	
2	3	12	2.0	3	49.0	1.0	1	2	
3	4	42	2.0	4	45.0	1.0	1	1	
4	5	24	3.0	4	53.0	2.0	1	1	
...	...	...	...	...	...	...	...	...	.
795	796	9	2.0	4	22.0	1.0	1	2	
796	797	18	1.0	4	51.0	1.0	1	1	
797	798	12	2.0	4	22.0	2.0	1	3	
798	799	24	4.0	4	54.0	2.0	1	3	
799	800	9	4.0	2	35.0	1.0	1	2	

800 rows × 21 columns

In [27]:

```
df2
```

Out[27]:

	sno	acc_info	duration_month	credit_history	purpose	savings_acc	employment_st
0	1	A14	24	A34	A46	A61	A75
1	2	A12	18	A34	A43	A61	A75
2	3	A11	20	A34	A42	A61	A75
3	4	A14	12	A34	A43	A65	A75
4	5	A12	12	A32	A40	A65	A71
...	...	...	...	...	...	...	...
195	196	A14	12	A32	A42	A61	A74
196	197	A11	30	A32	A41	A61	A73
197	198	A14	12	A32	A43	A61	A75
198	199	A11	45	A32	A43	A61	A73
199	200	A12	45	A34	A41	A62	A71

200 rows × 20 columns

In [28]:

```
# Create a label encoder object
label_encoder = LabelEncoder()

# Iterate over columns and encode categorical columns
```

```python
for column in df2.columns:
    if df2[column].dtype == 'object':  # Check if column contains categorical data
        df2[column] = label_encoder.fit_transform(df2[column])

# Display the encoded DataFrame
df2
```

Out[28]:

	sno	acc_info	duration_month	credit_history	purpose	savings_acc	employment_st
0	1	3	24	4	7	0	4
1	2	1	18	4	4	0	4
2	3	0	20	4	3	0	4
3	4	3	12	4	4	4	4
4	5	1	12	2	0	4	0
...	...	...	...	...	...	...	...
195	196	3	12	2	3	0	3
196	197	0	30	2	1	0	2
197	198	3	12	2	4	0	4
198	199	0	45	2	4	0	2
199	200	1	45	4	1	1	0

200 rows × 20 columns

In [29]:

```python
# Get the number of categories in each column
category_counts = {}
for column in df1_new.columns:
    category_counts[column] = len(df1_new[column].value_counts())

# Print or use the counts as needed
for column, count in category_counts.items():
    print(f"Column '{column}' has {count} categories.")
```

```
Column 'sno' has 800 categories.
Column 'duration_month' has 27 categories.
Column 'poi' has 5 categories.
Column 'resident_since' has 4 categories.
Column 'age' has 45 categories.
Column 'credits_no' has 4 categories.
Column 'liables' has 1 categories.
Column 'Group_no' has 3 categories.
Column 'acc_info' has 4 categories.
Column 'credit_history' has 5 categories.
Column 'purpose' has 10 categories.
Column 'savings_acc' has 5 categories.
Column 'employment_st' has 5 categories.
Column 'personal_status' has 4 categories.
Column 'gurantors' has 3 categories.
Column 'property_type' has 4 categories.
Column 'installment_type' has 3 categories.
Column 'housing_type' has 3 categories.
Column 'job_type' has 4 categories.
Column 'telephone' has 2 categories.
Column 'foreigner' has 2 categories.
```

Modelling

In [30]:
```
# Split data into features (X) and target variable (y)
X = df1_new.drop(columns=['Group_no'])  # Features
y = df1_new['Group_no']  # Target variable
```

In [31]:
```
# Mapping classes from [1, 2, 3] to [0, 1, 2]
class_mapping = {1: 0, 2: 1, 3: 2}
y= y.map(class_mapping)
```

In [32]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3,random_state=123)
```

In [33]:
```
# Define classifiers
classifiers = {
    "Random Forest": RandomForestClassifier(n_estimators=100, random_state=0),
    "Decision Tree": DecisionTreeClassifier(random_state=42),
    "Logistic Regression": LogisticRegression(),
    "NB": GaussianNB(),
    "MLP": MLPClassifier(),
    "Support Vector Machine": SVC(kernel='linear', random_state=42),
    "Gradient Boosting": GradientBoostingClassifier(n_estimators=100, learning_rate
}
```

In [34]:
```
# Define an empty dictionary to store accuracies
accuracies = {}

# Train and evaluate each classifier
for name, clf in classifiers.items():
    print(f"Training {name}...")
    # Train the classifier
    clf.fit(X_train, y_train)
```

```
# Make predictions on the testing set
y_pred = clf.predict(X_test)

# Calculate accuracy
print(" ")

accuracy = accuracy_score(y_test, y_pred)
print(f"                Accuracy for {name}: {accuracy*100:.2f}%")
accuracies[name] = round(accuracy*100,2)

# Print classification report
print(" ")

print(f"Classification Report for {name}:")
print(classification_report(y_test, y_pred))

# Print confusion matrix
print(f"Confusion Matrix for {name}:")
print(confusion_matrix(y_test, y_pred))
print("----------------------------------------------")
```

```
Training Random Forest...

                Accuracy for Random Forest: 67.50%

Classification Report for Random Forest:
              precision    recall  f1-score   support

           0       0.71      0.63      0.67        59
           1       0.64      0.70      0.67       110
           2       0.71      0.68      0.69        71

    accuracy                           0.68       240
   macro avg       0.69      0.67      0.68       240
weighted avg       0.68      0.68      0.68       240

Confusion Matrix for Random Forest:
[[37 20  2]
 [15 77 18]
 [ 0 23 48]]
----------------------------------------------
Training Decision Tree...

                Accuracy for Decision Tree: 49.58%

Classification Report for Decision Tree:
              precision    recall  f1-score   support

           0       0.61      0.63      0.62        59
           1       0.49      0.45      0.47       110
           2       0.42      0.45      0.44        71

    accuracy                           0.50       240
   macro avg       0.50      0.51      0.51       240
weighted avg       0.50      0.50      0.50       240

Confusion Matrix for Decision Tree:
[[37 14  8]
 [24 50 36]
 [ 0 39 32]]
----------------------------------------------
Training Logistic Regression...

                Accuracy for Logistic Regression: 63.75%

Classification Report for Logistic Regression:
              precision    recall  f1-score   support

           0       0.71      0.68      0.70        59
           1       0.63      0.60      0.62       110
           2       0.59      0.66      0.62        71

    accuracy                           0.64       240
   macro avg       0.65      0.65      0.64       240
weighted avg       0.64      0.64      0.64       240

Confusion Matrix for Logistic Regression:
```

```
[[40 14  5]
 [16 66 28]
 [ 0 24 47]]
----------------------------------------------
Training NB...

                Accuracy for NB: 60.83%

Classification Report for NB:
              precision    recall  f1-score   support

           0       0.58      0.71      0.64        59
           1       0.59      0.60      0.60       110
           2       0.68      0.54      0.60        71

    accuracy                           0.61       240
   macro avg       0.62      0.62      0.61       240
weighted avg       0.61      0.61      0.61       240

Confusion Matrix for NB:
[[42 15  2]
 [28 66 16]
 [ 3 30 38]]
----------------------------------------------
Training MLP...

                Accuracy for MLP: 57.08%

Classification Report for MLP:
              precision    recall  f1-score   support

           0       0.55      0.78      0.65        59
           1       0.63      0.35      0.45       110
           2       0.55      0.75      0.63        71

    accuracy                           0.57       240
   macro avg       0.58      0.62      0.58       240
weighted avg       0.59      0.57      0.55       240

Confusion Matrix for MLP:
[[46  7  6]
 [34 38 38]
 [ 3 15 53]]
----------------------------------------------
Training Support Vector Machine...

                Accuracy for Support Vector Machine: 65.00%

Classification Report for Support Vector Machine:
              precision    recall  f1-score   support

           0       0.68      0.61      0.64        59
           1       0.62      0.67      0.65       110
           2       0.68      0.65      0.66        71

    accuracy                           0.65       240
```

```
   macro avg       0.66      0.64      0.65       240
weighted avg       0.65      0.65      0.65       240

Confusion Matrix for Support Vector Machine:
[[36 20  3]
 [17 74 19]
 [ 0 25 46]]
----------------------------------------------
Training Gradient Boosting...

                Accuracy for Gradient Boosting: 68.75%

Classification Report for Gradient Boosting:
              precision    recall  f1-score   support

           0       0.66      0.69      0.68        59
           1       0.69      0.65      0.67       110
           2       0.71      0.73      0.72        71

    accuracy                           0.69       240
   macro avg       0.69      0.69      0.69       240
weighted avg       0.69      0.69      0.69       240

Confusion Matrix for Gradient Boosting:
[[41 14  4]
 [21 72 17]
 [ 0 19 52]]
----------------------------------------------
```

In [35]:

```
accuracies
```

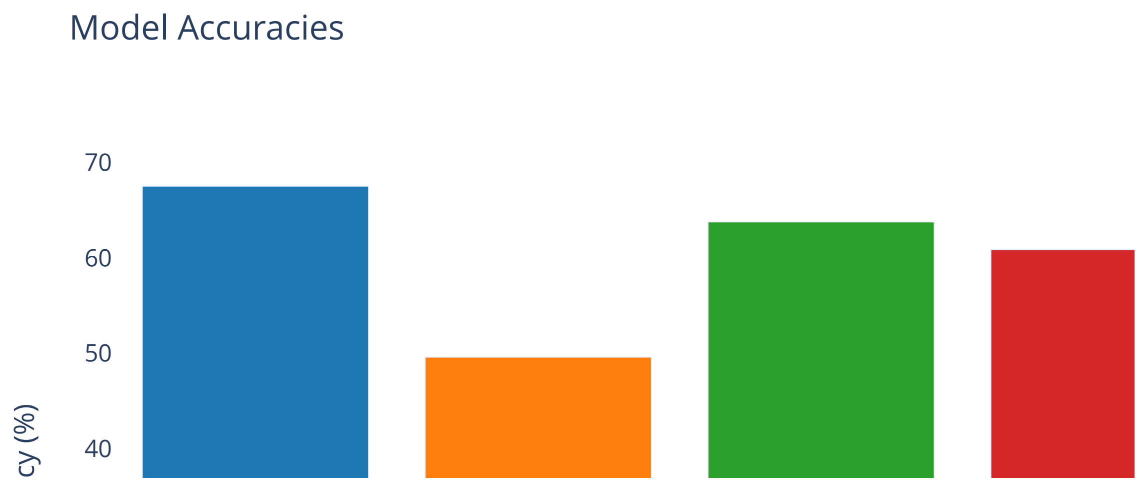
Out[35]:

```
{'Random Forest': 67.5,
 'Decision Tree': 49.58,
 'Logistic Regression': 63.75,
 'NB': 60.83,
 'MLP': 57.08,
 'Support Vector Machine': 65.0,
 'Gradient Boosting': 68.75}
```

In [36]:

```
# Define a color palette for the markers
color_palette = ['rgb(31, 119, 180)', 'rgb(255, 127, 14)', 'rgb(44, 160, 44)', 'rgb

# Create a trace for accuracy values with marker colors from the palette
trace = go.Bar(
    x=list(accuracies.keys()),
    y=list(accuracies.values()),
    marker_color=color_palette,
    name='Accuracy'
)

# Create the layout for the graph
layout = go.Layout(
    title='Model Accuracies',
    xaxis_title='Classifier',
    yaxis_title='Accuracy (%)',
    hovermode='closest',
```

```
    plot_bgcolor='rgba(0,0,0,0)'
)

# Create the figure
fig = go.Figure(data=[trace], layout=layout)

# Show the interactive graph
fig.show()
```



Selecting our best model

In [37]:

```
# Hyperparameter training for Random forest Classifier

# Define parameter grid for hyperparameter tuning
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Initialize and tune Random Forest Classifier
rfc = RandomForestClassifier(random_state=123)
```

```python
grid_search = GridSearchCV(rfc, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

# Get the best parameters
best_params = grid_search.best_params_

# Train RFC with best parameters
rfc_best = RandomForestClassifier(**best_params)
rfc_best.fit(X_train, y_train)
```

Out[37]:

```
RandomForestClassifier

RandomForestClassifier(min_samples_leaf=4, min_samples_split=10,
                       n_estimators=200)
```

In [38]:

```python
# Initialize and train Gradient Boosting Classifier
gbc = GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1,
gbc.fit(X_train, y_train)

# Predict on the test set
y_pred_gb = gbc.predict(X_test)

# Evaluate model performance
accuracy_gb = accuracy_score(y_test, y_pred_gb)
print("Accuracy for Gradient Boosting Classifier:", round(accuracy_gb * 100, 2))

# Classification report
print("Classification Report for Gradient Boosting Classifier:")
print(classification_report(y_test, y_pred_gb))
```

```
Accuracy for Gradient Boosting Classifier: 68.75
Classification Report for Gradient Boosting Classifier:
              precision    recall  f1-score   support

           0       0.66      0.69      0.68        59
           1       0.69      0.65      0.67       110
           2       0.71      0.73      0.72        71

    accuracy                           0.69       240
   macro avg       0.69      0.69      0.69       240
weighted avg       0.69      0.69      0.69       240
```

Even After hyperparameter tuning Gradiant boosting works best for us

In [39]:

```python
y_pred_gb
```

Out[39]:
```
array([0, 1, 1, 0, 2, 1, 2, 1, 2, 0, 0, 0, 1, 2, 1, 0, 2, 2, 0, 0, 1, 2,
       1, 2, 0, 0, 2, 2, 0, 1, 1, 1, 2, 0, 2, 1, 2, 1, 1, 2, 0, 2, 1, 2,
       0, 0, 1, 2, 2, 1, 2, 2, 1, 0, 1, 0, 2, 2, 2, 1, 1, 1, 0, 0, 0, 1,
       2, 0, 2, 2, 2, 0, 0, 1, 2, 0, 2, 1, 1, 1, 1, 2, 0, 2, 0, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 2, 2, 2, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1,
       1, 0, 2, 2, 1, 1, 2, 1, 1, 1, 0, 1, 0, 1, 1, 2, 0, 2, 1, 1, 2, 1,
       1, 2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 0, 1, 0, 2, 1, 1, 0, 0, 1, 1, 0,
       1, 0, 0, 1, 1, 1, 2, 1, 1, 0, 0, 2, 0, 0, 1, 1, 1, 0, 2, 1, 1, 1,
       1, 0, 2, 2, 0, 0, 0, 2, 0, 2, 2, 2, 1, 2, 2, 1, 2, 1, 1, 1, 1, 0,
       1, 1, 2, 2, 1, 2, 1, 1, 0, 1, 2, 0, 1, 1, 2, 2, 1, 2, 2, 1, 1, 0,
       1, 2, 2, 1, 1, 1, 1, 2, 1, 0, 2, 1, 1, 2, 2, 2, 2, 1, 0, 1],
      dtype=int64)
```

Save Model

In [40]:
```
import pickle

# Save the model to a file
with open('Credit Classification Problem Model C&T Bank.pkl', 'wb') as file:
    pickle.dump(GradientBoostingClassifier, file)

# Load the model from the file
with open('Credit Classification Problem Model C&T Bank.pkl', 'rb') as file:
    loaded_model = pickle.load(file)
```

Submission file

In [47]:
```
print("Training Data Columns:", X_train.columns)
print("Prediction Data Columns:", df2.columns)
```

```
Training Data Columns: Index(['sno', 'duration_month', 'poi', 'resident_since', 'ag
e', 'credits_no',
       'liables', 'acc_info', 'credit_history', 'purpose', 'savings_acc',
       'employment_st', 'personal_status', 'gurantors', 'property_type',
       'installment_type', 'housing_type', 'job_type', 'telephone',
       'foreigner'],
      dtype='object')
Prediction Data Columns: Index(['sno', 'acc_info', 'duration_month', 'credit_histor
y', 'purpose',
       'savings_acc', 'employment_st', 'poi', 'personal_status', 'gurantors',
       'resident_since', 'property_type', 'age', 'installment_type',
       'housing_type', 'credits_no', 'job_type', 'liables', 'telephone',
       'foreigner'],
      dtype='object')
```

In [48]:
```
# Reorder columns in the prediction data to match training data
df2 = df2[X_train.columns]
```

In [49]:
```
submission_pred=gbc.predict(df2)
```

In [50]:
```
submission_pred
```

Out[50]:
```
array([1, 1, 0, 2, 2, 0, 2, 2, 0, 2, 2, 2, 0, 0, 0, 0, 1, 1, 0, 1, 2, 1,
       1, 2, 1, 2, 0, 1, 0, 0, 1, 2, 0, 0, 2, 2, 2, 1, 0, 1, 1, 1, 2, 1,
       0, 1, 1, 2, 2, 2, 2, 0, 2, 1, 0, 2, 2, 1, 2, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 2, 0, 2, 1, 2, 1, 2, 1, 1, 2, 0, 0, 0, 0, 2, 1, 2, 1, 0,
       0, 0, 0, 2, 2, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 2, 1, 1, 1, 0, 0, 2,
       1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 2, 1, 1, 2, 1, 0, 1, 2, 1, 2,
       2, 2, 2, 0, 2, 2, 0, 0, 0, 2, 1, 1, 1, 0, 0, 2, 1, 0, 2, 0, 1, 0,
       2, 0, 0, 2, 1, 1, 2, 1, 1, 1, 2, 1, 1, 1, 0, 1, 2, 1, 1, 0, 1, 1,
       2, 2, 1, 2, 0, 0, 0, 0, 1, 2, 0, 1, 0, 1, 1, 2, 1, 0, 2, 1, 0, 2,
       0, 0], dtype=int64)
```

In [51]:
```
copy.head()
```

Out[51]:

	sno	acc_info	duration_month	credit_history	purpose	savings_acc	employment_st	po
0	1	A14	24	A34	A46	A61	A75	4.
1	2	A12	18	A34	A43	A61	A75	3.
2	3	A11	20	A34	A42	A61	A75	1.
3	4	A14	12	A34	A43	A65	A75	4.
4	5	A12	12	A32	A40	A65	A71	1.

In [52]:
```
final=pd.DataFrame({'serial number':copy.sno.values,'Group_no':submission_pred})
```

In [53]:
```
final
```

Out[53]:

	serial number	Group_no
0	1	1
1	2	1
2	3	0
3	4	2
4	5	2
...	...	...
195	196	1
196	197	0
197	198	2
198	199	0
199	200	0

200 rows × 2 columns

In [54]:
```
# Mapping classes from [0, 1, 2] to [1, 2, 3]
class_mapping = {0: 1, 1: 2, 2: 3}
final['Group_no']= final['Group_no'].map(class_mapping)
```

In [55]:
```
final
```

Out[55]:

	serial number	Group_no
0	1	2
1	2	2
2	3	1
3	4	3
4	5	3
...	...	...
195	196	2
196	197	1
197	198	3
198	199	1
199	200	1

200 rows × 2 columns

In [56]:
```
final.to_csv('submission.csv',index=False)  #by default index is true
```