In [1]:
```python
%matplotlib inline
import pandas as pd
import numpy as np ; np.random.seed(sum(map(ord, "aesthetics")))
import seaborn as sns
import matplotlib.pyplot as plt
from collections import defaultdict
from sklearn.preprocessing import OneHotEncoder, LabelEncoder
from sklearn.metrics import classification_report,confusion_matrix, roc_curve, roc_auc_score, auc, accu
racy_score
from sklearn.datasets import make_classification
from sklearn.model_selection import learning_curve, GridSearchCV

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from boruta import BorutaPy
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from scipy import interp

# Ignoring Warnings
import warnings
warnings.filterwarnings("ignore")
```

Data Preprocessing

In [2]:
```python
credit_data = pd.read_csv("C://Users//Imtiyaz//Desktop//Task//credit.csv")
```

In [3]:
```python
credit_data.head()
```

Out[3]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	employment_duration	percent_of
0	< 0 DM	6	critical	furniture/appliances	1169	unknown	> 7 years	
1	1 - 200 DM	48	good	furniture/appliances	5951	< 100 DM	1 - 4 years	
2	unknown	12	critical	education	2096	< 100 DM	4 - 7 years	
3	< 0 DM	42	good	furniture/appliances	7882	< 100 DM	4 - 7 years	
4	< 0 DM	24	poor	car	4870	< 100 DM	1 - 4 years	

In [4]:
```python
credit_data.columns
```

Out[4]:
```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

In [5]:
```python
#Encoding numerical variables
num_var = ['months_loan_duration','amount', 'percent_of_income', 'years_at_residence', 'age', 'existing
_loans_count', 'dependents']
print(num_var)
```

```
['months_loan_duration', 'amount', 'percent_of_income', 'years_at_residence', 'age', 'existing_loans_
count', 'dependents']
```

In [6]:
```python
# Standardization
num_var_std = pd.DataFrame(StandardScaler().fit_transform(credit_data[num_var]))
```

In [7]:
```python
num_var_std
```

Out[7]:

	0	1	2	3	4	5	6
0	-1.236478	-0.745131	0.918477	1.046987	2.766456	1.027079	-0.428290
1	2.248194	0.949817	-0.870183	-0.765977	-1.191404	-0.704926	-0.428290
2	-0.738668	-0.416562	-0.870183	0.140505	1.183312	-0.704926	2.334869
3	1.750384	1.634247	-0.870183	1.046987	0.831502	-0.704926	2.334869
4	0.256953	0.566664	0.024147	1.046987	1.535122	1.027079	2.334869
...	...	...	...	...	...	...	...
995	-0.738668	-0.544162	0.024147	1.046987	-0.399832	-0.704926	-0.428290
996	0.754763	0.207612	0.918477	1.046987	0.391740	-0.704926	-0.428290
997	-0.738668	-0.874503	0.918477	1.046987	0.215835	-0.704926	-0.428290
998	1.999289	-0.505528	0.918477	1.046987	-1.103451	-0.704926	-0.428290
999	1.999289	0.462457	0.024147	1.046987	-0.751642	-0.704926	-0.428290

1000 rows × 7 columns

In [8]:
```python
#Encoding Categorical variables
cat_var = ['checking_balance', 'credit_history', 'purpose', 'savings_balance', 'employment_duration',
'other_credit', 'housing','job', 'phone', 'default']
```

In [9]:
```python
#creat default dictionary
def_dict = defaultdict(LabelEncoder)
type(def_dict)
```

Out[9]: collections.defaultdict

In [10]:
```python
# Encoding the categorical variable into numerical variable
Encoded_catvar = credit_data[cat_var].apply(lambda x:def_dict[x.name].fit_transform(x))
```

In [11]:
```python
# print transformations
for x in range(len(cat_var)):
    print(cat_var[x],": ", credit_data[cat_var[x]].unique())
    print(cat_var[x],": ", Encoded_catvar[cat_var[x]].unique())
```

```
checking_balance :  ['< 0 DM' '1 - 200 DM' 'unknown' '> 200 DM']
checking_balance :  [1 0 3 2]
credit_history :  ['critical' 'good' 'poor' 'perfect' 'very good']
credit_history :  [0 1 3 2 4]
purpose :  ['furniture/appliances' 'education' 'car' 'business' 'renovations' 'car0']
purpose :  [4 3 1 0 5 2]
savings_balance :  ['unknown' '< 100 DM' '500 - 1000 DM' '> 1000 DM' '100 - 500 DM']
savings_balance :  [4 2 1 3 0]
employment_duration :  ['> 7 years' '1 - 4 years' '4 - 7 years' 'unemployed' '< 1 year']
employment_duration :  [3 0 1 4 2]
other_credit :  ['none' 'bank' 'store']
other_credit :  [1 0 2]
housing :  ['own' 'other' 'rent']
housing :  [1 0 2]
job :  ['skilled' 'unskilled' 'management' 'unemployed']
job :  [1 3 0 2]
phone :  ['yes' 'no']
phone :  [1 0]
default :  ['no' 'yes']
default :  [0 1]
```

In [12]:
```python
dummy_vars = pd.get_dummies(credit_data[cat_var])
```

In [13]:
```python
dummy_vars
```

Out[13]:

	checking_balance_1 - 200 DM	checking_balance_< 0 DM	checking_balance_> 200 DM	checking_balance_unknown	credit_history_critical	credit_history_good
0	0	1	0	0	1	0
1	1	0	0	0	0	1
2	0	0	0	1	1	0
3	0	1	0	0	0	1
4	0	1	0	0	0	0
...	...	...	...	...	...	...
995	0	0	0	1	0	1
996	0	1	0	0	0	1
997	0	0	0	1	0	1
998	0	1	0	0	0	1
999	1	0	0	0	1	0

1000 rows × 39 columns

In [14]:
```python
clean_credit_data = pd.concat([credit_data[num_var], dummy_vars], axis = 1)
print(clean_credit_data.shape)
```

```
(1000, 46)
```

In [15]:
```python
clean_credit_data = clean_credit_data.drop(['default_no', 'phone_no'], axis=1)
```

In [16]:
```python
#renaming the columns
clean_credit_data = clean_credit_data.rename(columns={"phone_yes": "phone", "default_yes": "default"})
```

In [17]:
```python
#find correlation of each variable
#clean_credit_data.corr(method ='kendall')
```

Data Exporatory Analysis

In [18]:
```python
#Data Exporatory Analysis
clean_credit_data.describe()
```

Out[18]:

	months_loan_duration	amount	percent_of_income	years_at_residence	age	existing_loans_count	dependents	chec
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	
mean	20.903000	3271.258000	2.973000	2.845000	35.546000	1.407000	1.155000	
std	12.058814	2822.736876	1.118715	1.103718	11.375469	0.577654	0.362086	
min	4.000000	250.000000	1.000000	1.000000	19.000000	1.000000	1.000000	
25%	12.000000	1365.500000	2.000000	2.000000	27.000000	1.000000	1.000000	
50%	18.000000	2319.500000	3.000000	3.000000	33.000000	1.000000	1.000000	
75%	24.000000	3972.250000	4.000000	4.000000	42.000000	2.000000	1.000000	
max	72.000000	18424.000000	4.000000	4.000000	75.000000	4.000000	2.000000	

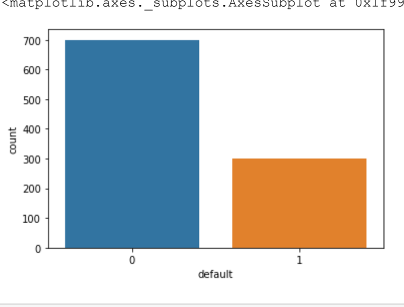
8 rows × 44 columns

In [19]:
```python
# checking null values
clean_credit_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 44 columns):
 #   Column                           Non-Null Count  Dtype
---  ------                           --------------  -----
 0   months_loan_duration             1000 non-null   int64
 1   amount                           1000 non-null   int64
 2   percent_of_income                1000 non-null   int64
 3   years_at_residence               1000 non-null   int64
 4   age                              1000 non-null   int64
 5   existing_loans_count             1000 non-null   int64
 6   dependents                       1000 non-null   int64
 7   checking_balance_1 - 200 DM      1000 non-null   uint8
 8   checking_balance_< 0 DM          1000 non-null   uint8
 9   checking_balance_> 200 DM        1000 non-null   uint8
 10  checking_balance_unknown         1000 non-null   uint8
 11  credit_history_critical          1000 non-null   uint8
 12  credit_history_good              1000 non-null   uint8
 13  credit_history_perfect           1000 non-null   uint8
 14  credit_history_poor              1000 non-null   uint8
 15  credit_history_very good         1000 non-null   uint8
 16  purpose_business                 1000 non-null   uint8
 17  purpose_car                      1000 non-null   uint8
 18  purpose_car0                     1000 non-null   uint8
 19  purpose_education                1000 non-null   uint8
 20  purpose_furniture/appliances     1000 non-null   uint8
 21  purpose_renovations              1000 non-null   uint8
 22  savings_balance_100 - 500 DM     1000 non-null   uint8
 23  savings_balance_500 - 1000 DM    1000 non-null   uint8
 24  savings_balance_< 100 DM         1000 non-null   uint8
 25  savings_balance_> 1000 DM        1000 non-null   uint8
 26  savings_balance_unknown          1000 non-null   uint8
 27  employment_duration_1 - 4 years  1000 non-null   uint8
 28  employment_duration_4 - 7 years  1000 non-null   uint8
 29  employment_duration_< 1 year     1000 non-null   uint8
 30  employment_duration_> 7 years    1000 non-null   uint8
 31  employment_duration_unemployed   1000 non-null   uint8
 32  other_credit_bank                1000 non-null   uint8
 33  other_credit_none                1000 non-null   uint8
 34  other_credit_store               1000 non-null   uint8
 35  housing_other                    1000 non-null   uint8
 36  housing_own                      1000 non-null   uint8
 37  housing_rent                     1000 non-null   uint8
 38  job_management                   1000 non-null   uint8
 39  job_skilled                      1000 non-null   uint8
 40  job_unemployed                   1000 non-null   uint8
 41  job_unskilled                    1000 non-null   uint8
 42  phone                            1000 non-null   uint8
 43  default                          1000 non-null   uint8
dtypes: int64(7), uint8(37)
memory usage: 90.9 KB
```

In [20]:
```python
sns.countplot(x='default', data=clean_credit_data)
```

Out[20]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99caefc08>



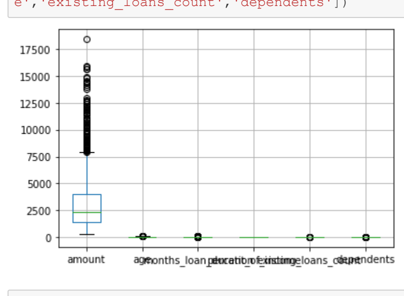
In [21]:
```python
# Correlation Martix
clean_credit_data.corr()
```

Out[21]:

	months_loan_duration	amount	percent_of_income	years_at_residence	age	existing_loans_cou
months_loan_duration	1.000000	0.624984	0.074749	0.034067	-0.036136	-0.01128
amount	0.624984	1.000000	-0.271316	0.028926	0.032716	0.02079
percent_of_income	0.074749	-0.271316	1.000000	0.049302	0.058266	0.02166
years_at_residence	0.034067	0.028926	0.049302	1.000000	0.266419	0.08962
age	-0.036136	0.032716	0.058266	0.266419	1.000000	0.14925
existing_loans_count	-0.011284	0.020795	0.021669	0.089625	0.149254	1.00000
dependents	-0.023834	0.017142	-0.071207	0.042643	0.118201	0.10966
checking_balance_1 - 200 DM	0.089452	0.119612	-0.051906	-0.055817	-0.078121	-0.05266
checking_balance_< 0 DM	0.022244	-0.020912	0.046917	0.088350	-0.011162	-0.02919
checking_balance_> 200 DM	-0.076455	-0.100510	-0.041591	-0.064303	0.037504	-0.04021
checking_balance_unknown	-0.063467	-0.039485	0.024961	0.001985	0.062436	0.09443
credit_history_critical	-0.075575	-0.041807	0.041089	0.088460	0.163681	0.50136
credit_history_good	-0.069751	-0.086682	-0.020947	-0.081458	-0.155848	-0.54035
credit_history_perfect	0.118077	0.147191	-0.054401	0.000925	-0.022370	0.11242
credit_history_poor	0.136927	0.113552	-0.014597	-0.020351	0.016129	0.14174
credit_history_very good	0.033728	0.005923	0.030339	0.027694	0.014360	-0.09582
purpose_business	0.164113	0.103016	-0.025326	-0.048899	-0.001772	0.08498
purpose_car	-0.005320	0.125575	-0.101982	0.086749	0.099919	0.02873
purpose_car0	0.104516	0.192893	-0.030193	0.040460	0.042365	0.01775
purpose_education	-0.025450	-0.034796	0.055389	0.042876	0.060390	-0.01479
purpose_furniture/appliances	-0.096404	-0.197118	0.080197	-0.090209	-0.142913	-0.09543
purpose_renovations	-0.022549	-0.028875	0.040204	0.027253	0.039567	0.07139
savings_balance_100 - 500 DM	0.051587	0.013546	-0.015356	0.017789	-0.075302	-0.01094
savings_balance_500 - 1000 DM	-0.040257	-0.064256	-0.023186	0.032702	0.030988	-0.06159
savings_balance_< 100 DM	-0.047228	-0.036443	-0.008626	-0.089921	-0.044084	0.03390
savings_balance_> 1000 DM	-0.048261	-0.055542	0.034708	-0.002375	0.032007	0.03616
savings_balance_unknown	0.071185	0.106546	0.018367	0.080564	0.077811	-0.01559
employment_duration_1 - 4 years	-0.031920	-0.037052	-0.069614	-0.140663	-0.154975	-0.07307
employment_duration_4 - 7 years	0.079635	0.053755	-0.000712	-0.031156	-0.083520	0.03738
employment_duration_< 1 year	-0.056792	-0.051502	-0.034021	-0.164155	-0.209967	-0.09639
employment_duration_> 7 years	0.017471	-0.009619	0.133371	0.302796	0.362520	0.12362
employment_duration_unemployed	-0.005156	0.086159	-0.049420	0.036123	0.110562	0.01268
other_credit_bank	0.035851	0.039474	-0.016150	0.014530	0.046552	0.04219
other_credit_none	-0.067602	-0.048292	-0.016139	0.016704	-0.035362	-0.05029
other_credit_store	0.065688	0.024262	0.056071	-0.054460	-0.011080	0.02349
housing_other	0.189117	0.201643	0.040098	0.227044	0.253058	0.01140
housing_own	-0.075169	-0.117497	0.049922	-0.297547	0.006553	0.04138
housing_rent	-0.064417	-0.024611	-0.091373	0.167285	-0.212620	-0.05807
job_management	0.147515	0.319715	0.042805	0.004952	0.127605	-0.01090
job_skilled	0.055010	-0.092636	0.042623	-0.000657	-0.148283	-0.00147
job_unemployed	-0.044043	-0.027969	-0.087834	-0.034545	0.059954	0.05958
job_unskilled	-0.181203	-0.161757	-0.057237	0.009065	0.043712	-0.01039
phone	0.164718	0.276995	0.014413	0.095359	0.145259	0.06555
default	0.214927	0.154739	0.072404	0.002967	-0.091127	-0.04573

44 rows × 44 columns

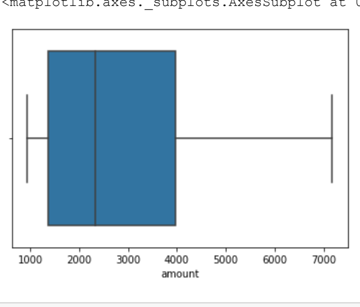
In [22]:
```python
#checking outliers
boxplot = clean_credit_data.boxplot(column=['amount', 'age', 'months_loan_duration', 'percent_of_incom
e','existing_loans_count','dependents'])
```



In [23]:
```python
# removing outliers from amount variable
print(clean_credit_data['amount'].quantile(0.10))
print(clean_credit_data['amount'].quantile(0.90))
clean_credit_data["amount"] = np.where(clean_credit_data["amount"] <932.0, 932.0,clean_credit_data['amo
unt'])
clean_credit_data["amount"] = np.where(clean_credit_data["amount"] >7179.400, 7179.400,clean_credit_dat
a['amount'])
sns.boxplot(x=clean_credit_data['amount'])
```

```
932.0
7179.4000000000015
```

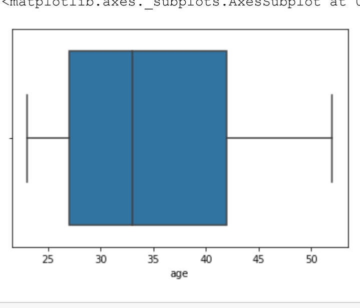
Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99eb5f9c8>



In [24]:
```python
#removing outliers from age variable
print(clean_credit_data['age'].quantile(0.10))
print(clean_credit_data['age'].quantile(0.90))
clean_credit_data["age"] = np.where(clean_credit_data["age"] <23.0, 23.0,clean_credit_data['age'])
clean_credit_data["age"] = np.where(clean_credit_data["age"] >52.0, 52.0,clean_credit_data['age'])
sns.boxplot(x=clean_credit_data['age'])
```

```
23.0
52.0
```

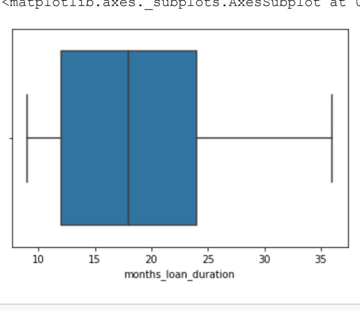
Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99ed8a188>



In [25]:
```python
#removing outliers from months_loan_duration variable
print(clean_credit_data['months_loan_duration'].quantile(0.10))
print(clean_credit_data['months_loan_duration'].quantile(0.90))
clean_credit_data["months_loan_duration"] = np.where(clean_credit_data["months_loan_duration"] <9.0, 9.
0,clean_credit_data['months_loan_duration'])
clean_credit_data["months_loan_duration"] = np.where(clean_credit_data["months_loan_duration"] >36.0, 3
6.0,clean_credit_data['months_loan_duration'])
sns.boxplot(x=clean_credit_data['months_loan_duration'])
```

```
9.0
36.0
```

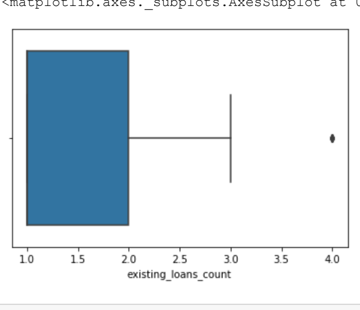
Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99ede3608>



In [26]:
```python
#removing outliers from existing_loans_count variable
print(clean_credit_data['existing_loans_count'].quantile(0.10))
print(clean_credit_data['existing_loans_count'].quantile(0.90))
#clean_credit_data["existing_loans_count"] = np.where(clean_credit_data["existing_loans_count"] <1.0,
 1.0,clean_credit_data['existing_loans_count'])
#clean_credit_data["existing_loans_count"] = np.where(clean_credit_data["existing_loans_count"] >2.0,
 2.0,clean_credit_data['existing_loans_count'])
sns.boxplot(x=clean_credit_data['existing_loans_count'])
```

```
1.0
2.0
```

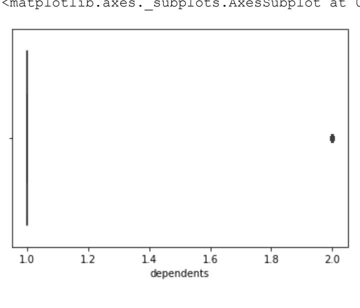
Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99ee32688>



In [27]:
```python
#removing outliers from dependents variable
print(clean_credit_data['dependents'].quantile(0.10))
print(clean_credit_data['dependents'].quantile(0.90))
#clean_credit_data["existing_loans_count"] = np.where(clean_credit_data["existing_loans_count"] <1.0,
 1.0,clean_credit_data['existing_loans_count'])
#clean_credit_data["existing_loans_count"] = np.where(clean_credit_data["existing_loans_count"] >2.0,
 2.0,clean_credit_data['existing_loans_count'])
sns.boxplot(x=clean_credit_data['dependents'])
```

```
1.0
2.0
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1f99eea8488>



Feature Selection

In [28]:
```python
# feature selection using boruta
features = [f for f in clean_credit_data.columns if f not in ['default']]
len(features)
```

Out[28]: 43

In [29]:
```python
X = clean_credit_data[features].values
Y = clean_credit_data['default'].values.ravel()
```

In [30]:
```python
rf = RandomForestClassifier(n_jobs=-1, class_weight='balanced', max_depth=5)
```

In [31]:
```python
boruta_feature_selector = BorutaPy(rf, n_estimators='auto', verbose=2, random_state=4242, max_iter = 50
, perc = 90)
boruta_feature_selector.fit(X, Y)
```

```
Iteration: 	1 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	2 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	3 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	4 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	5 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	6 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	7 / 50
Confirmed: 	0
Tentative: 	43
Rejected: 	0
Iteration: 	8 / 50
Confirmed: 	14
Tentative: 	5
Rejected: 	24
Iteration: 	9 / 50
Confirmed: 	14
Tentative: 	5
Rejected: 	24
Iteration: 	10 / 50
Confirmed: 	14
Tentative: 	5
Rejected: 	24
Iteration: 	11 / 50
Confirmed: 	14
Tentative: 	5
Rejected: 	24
Iteration: 	12 / 50
Confirmed: 	14
Tentative: 	5
Rejected: 	24
Iteration: 	13 / 50
Confirmed: 	14
Tentative: 	4
Rejected: 	25
Iteration: 	14 / 50
Confirmed: 	14
Tentative: 	4
Rejected: 	25
Iteration: 	15 / 50
Confirmed: 	14
Tentative: 	4
Rejected: 	25
Iteration: 	16 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	17 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	18 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	19 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	20 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	21 / 50
Confirmed: 	14
Tentative: 	3
Rejected: 	26
Iteration: 	22 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	23 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	24 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	25 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	26 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	27 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	28 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	29 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	30 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	31 / 50
Confirmed: 	14
Tentative: 	2
Rejected: 	27
Iteration: 	32 / 50
Confirmed: 	14
Tentative: 	0
Rejected: 	29


BorutaPy finished running.

Iteration: 	33 / 50
Confirmed: 	14
Tentative: 	0
Rejected: 	29
```

Out[31]:
```
BorutaPy(alpha=0.05,
         estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                          class_weight='balanced',
                                          criterion='gini', max_depth=5,
                                          max_features='auto',
                                          max_leaf_nodes=None, max_samples=None,
                                          min_impurity_decrease=0.0,
                                          min_impurity_split=None,
                                          min_samples_leaf=1,
                                          min_samples_split=2,
                                          min_weight_fraction_leaf=0.0,
                                          n_estimators=113, n_jobs=-1,
                                          oob_score=False,
                                          random_state=RandomState(MT19937) at 0x1F99C991D08,
                                          verbose=0, warm_start=False),
         max_iter=50, n_estimators='auto', perc=90,
         random_state=RandomState(MT19937) at 0x1F99C991D08, two_step=True,
         verbose=2)
```

In [32]:
```python
X_filtered = boruta_feature_selector.transform(X)
X_filtered.shape
```

Out[32]: (1000, 14)

In [33]:
```python
final_features = list()
indexes = np.where(boruta_feature_selector.support_ == True)
for x in np.nditer(indexes):
    final_features.append(features[x])
print(final_features)
```

```
['months_loan_duration', 'amount', 'percent_of_income', 'age', 'checking_balance_1 - 200 DM', 'checki
ng_balance_< 0 DM', 'checking_balance_unknown', 'credit_history_critical', 'credit_history_perfect',
'credit_history_very good', 'savings_balance_< 100 DM', 'savings_balance_unknown', 'other_credit_non
e', 'housing_own']
```

Models with Unbalanced Data with all features

In [34]:
```python
X = clean_credit_data.drop('default', axis=1)
y = clean_credit_data['default']
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.2, random_state=1)
```

In [35]:
```python
#LogisticRegression with Unbalanced data  (all features)
model = LogisticRegression()
model.fit(X_train, y_train)
model.score(X_test, y_test)
```

Out[35]: 0.765

In [36]:
```python
#Confusion Matrix
y_pred = model.predict(X_test)
#Compute precision, recall, F-measure and support
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       141
           1       0.65      0.44      0.53        59

    accuracy                           0.77       200
   macro avg       0.72      0.67      0.68       200
weighted avg       0.75      0.77      0.75       200
```

In [37]:
```python
# Decision Tree Classifier using grid search with unbalanced data (all features)

param_grid = {'max_depth': np.arange(3, 10)}

tree = GridSearchCV(DecisionTreeClassifier(), param_grid)

tree.fit(X_train, y_train)
tree_preds = tree.predict_proba(X_test)[:, 1]
tree_performance = roc_auc_score(y_test, tree_preds)
print(classification_report(y_test, tree_preds.round()))
```

```
              precision    recall  f1-score   support

           0       0.78      0.79      0.78       141
           1       0.47      0.46      0.47        59

    accuracy                           0.69       200
   macro avg       0.62      0.62      0.62       200
weighted avg       0.69      0.69      0.69       200
```

In [38]:
```python
# RandomForest Classifier using grid search with unbalanced data (all features)

rfc=RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, y_train)
CV_rfc.best_params_
```

Out[38]:
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 200}
```

In [39]:
```python
rfc1=RandomForestClassifier(random_state=42, max_features='auto', n_estimators= 200, max_depth=8 , crit
erion='gini')
rfc1.fit(X_train, y_train)
```

Out[39]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=200,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [40]:
```python
rf_pred=rfc1.predict(X_test)
print(classification_report(y_test, rf_pred.round()))
```

```
              precision    recall  f1-score   support

           0       0.78      0.94      0.85       141
           1       0.71      0.37      0.49        59

    accuracy                           0.77       200
   macro avg       0.75      0.65      0.67       200
weighted avg       0.76      0.77      0.74       200
```

Models with Unbalanced Data with selected features

In [41]:
```python
# Spliting data into training nd testing for Unbalanced data
X_clean = clean_credit_data.filter(['months_loan_duration', 'amount', 'percent_of_income', 'age', 'chec
king_balance_1 - 200 DM', 'checking_balance_< 0 DM', 'checking_balance_unknown', 'credit_history_critic
al', 'credit_history_perfect', 'credit_history_very good', 'savings_balance_< 100 DM', 'savings_balance
_unknown', 'other_credit_none', 'housing_own'])
y_clean = clean_credit_data['default']
X_train_clean, X_test_clean, y_train_clean, y_test_clean = train_test_split(X_clean,y_clean,test_size=
0.2, random_state=1)
```

In [42]:
```python
#LogisticRegression with Unbalanced data
model = LogisticRegression()
model.fit(X_train_clean, y_train_clean)
model.score(X_test_clean, y_test_clean)
```

Out[42]: 0.75

In [43]:
```python
#Confusion Matrix
y_pred = model.predict(X_test_clean)
confusion_matrix = confusion_matrix(y_test_clean, y_pred)
print(confusion_matrix)
```

```
[[127  14]
 [ 36  23]]
```

In [44]:
```
#Compute precision, recall, F-measure and support
print(classification_report(y_test_clean, y_pred))
```
```
              precision    recall  f1-score   support

           0       0.78      0.90      0.84       141
           1       0.62      0.39      0.48        59

    accuracy                           0.75       200
   macro avg       0.70      0.65      0.66       200
weighted avg       0.73      0.75      0.73       200
```

In [45]:
```
# Decision Tree Classifier using grid search with unbalanced data

param_grid = {'max_depth': np.arange(3, 10)}

tree = GridSearchCV(DecisionTreeClassifier(), param_grid)

tree.fit(X_train_clean, y_train_clean)
tree_preds = tree.predict_proba(X_test_clean)[:, 1]
tree_performance = roc_auc_score(y_test_clean, tree_preds)

print(classification_report(y_test_clean, tree_preds.round()))
```
```
              precision    recall  f1-score   support

           0       0.79      0.83      0.81       141
           1       0.54      0.47      0.50        59

    accuracy                           0.73       200
   macro avg       0.66      0.65      0.66       200
weighted avg       0.72      0.72      0.72       200
```

In [46]:
```
# RandomForest Classifier using grid search with unbalanced data

rfc=RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train_clean, y_train_clean)
CV_rfc.best_params_
```
Out[46]:
```
{'criterion': 'gini',
 'max_depth': 7,
 'max_features': 'auto',
 'n_estimators': 500}
```

In [47]:
```
rfc1=RandomForestClassifier(random_state=42, max_features='auto', n_estimators= 500, max_depth=7 , crit
erion='gini')
rfc1.fit(X_train_clean, y_train_clean)
```
Out[47]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=7, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [48]:
```
rf_pred=rfc1.predict(X_test_clean)
print(classification_report(y_test_clean, rf_pred.round()))
```
```
              precision    recall  f1-score   support

           0       0.77      0.91      0.83       141
           1       0.61      0.34      0.43        59

    accuracy                           0.74       200
   macro avg       0.69      0.62      0.63       200
weighted avg       0.72      0.74      0.71       200
```

Balanced Data with Majority Class

In [49]:
```
# Makimg balanced data for majority class
from sklearn.utils import resample
# Separate majority and minority classes
df_majority = clean_credit_data[clean_credit_data.default==0]
df_minority = clean_credit_data[clean_credit_data.default==1]

# Upsample minority class
df_minority_upsampled = resample(df_minority,
                                 replace=True,     # sample with replacement
                                 n_samples=700,    # to match majority class
                                 random_state=123) # reproducible results

# Combine majority class with upsampled minority class
df_upsampled = pd.concat([df_majority, df_minority_upsampled])

# Display new class counts
df_upsampled.default.value_counts()
```
Out[49]:
```
1    700
0    700
Name: default, dtype: int64
```

In [50]:
```
# Spliting data for Balanced Data with Majority Class
X_upsampled = df_upsampled.filter(['months_loan_duration', 'amount', 'percent_of_income', 'age', 'check
ing_balance_1 - 200 DM', 'checking_balance_< 0 DM', 'checking_balance_unknown', 'credit_history_critica
l', 'credit_history_perfect', 'credit_history_very good', 'savings_balance_< 100 DM', 'savings_balance_
unknown', 'other_credit_none', 'housing_own'])
y_upsampled = df_upsampled['default']
X_train_upsampled, X_test_upsampled, y_train_upsampled, y_test_upsampled = train_test_split(X_upsampled
,y_upsampled,test_size=0.2, random_state=1)
```

In [51]:
```
#LogisticRegression with Balanced Data with Majority Class

model = LogisticRegression()
model.fit(X_train_upsampled, y_train_upsampled)
model.score(X_test_upsampled, y_test_upsampled)
```
Out[51]:
```
0.7464285714285714
```

In [52]:
```
#Confusion Matrix
y_pred = model.predict(X_test_upsampled)
#confusion_matrix = confusion_matrix(y_test_upsampled, y_pred)
#print(confusion_matrix)

#Compute precision, recall, F-measure and support
print(classification_report(y_test_upsampled, y_pred))
```
```
              precision    recall  f1-score   support

           0       0.76      0.71      0.73       137
           1       0.74      0.78      0.76       143

    accuracy                           0.75       280
   macro avg       0.75      0.75      0.75       280
weighted avg       0.75      0.75      0.75       280
```

In [53]:
```
# Decision Tree Classifier using grid search with Majority balanced data

param_grid = {'max_depth': np.arange(3, 10)}

tree = GridSearchCV(DecisionTreeClassifier(), param_grid)

tree.fit(X_train_upsampled, y_train_upsampled)
tree_preds = tree.predict_proba(X_test_upsampled)[:, 1]
tree_performance = roc_auc_score(y_test_upsampled, tree_preds)

print(classification_report(y_test_upsampled, tree_preds.round()))
```
```
              precision    recall  f1-score   support

           0       0.83      0.71      0.76       137
           1       0.75      0.86      0.80       143

    accuracy                           0.79       280
   macro avg       0.79      0.78      0.78       280
weighted avg       0.79      0.79      0.78       280
```

In [54]:
```
# RandomForest Classifier using grid search with with Majority balanced data

rfc=RandomForestClassifier(random_state=42)

param_grid = {
    'n_estimators': [200, 500],
    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}

CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train_upsampled, y_train_upsampled)
CV_rfc.best_params_
```
Out[54]:
```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 500}
```

In [55]:
```
rfc1=RandomForestClassifier(random_state=42, max_features='auto', n_estimators= 500, max_depth=8, crite
rion='gini')
rfc1.fit(X_train_upsampled, y_train_upsampled)
```
Out[55]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=8, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=500,
                       n_jobs=None, oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

In [56]:
```
rf_pred=rfc1.predict(X_test_upsampled)
print(classification_report(y_test_upsampled, rf_pred.round()))
```
```
              precision    recall  f1-score   support

           0       0.90      0.81      0.85       137
           1       0.83      0.91      0.87       143

    accuracy                           0.86       280
   macro avg       0.86      0.86      0.86       280
weighted avg       0.86      0.86      0.86       280
```