1.) Imports

In [64]:

```
import numpy as np
from numpy import mean
from numpy import std
import pandas as pd

from scipy import stats
from scipy.stats import boxcox

import math

import matplotlib.pyplot as plt
import seaborn as sns

#from sklearn.experimental import enable_iterative_imputer
#from sklearn.impute import IterativeImputer

from sklearn import preprocessing
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler

from sklearn.pipeline import Pipeline

from sklearn.model_selection import train_test_split
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import cross_val_score

from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.feature_selection import RFE

from sklearn.linear_model import BayesianRidge
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Perceptron

from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

2.) EDA

In [2]:

```
df = pd.read_csv('cibil_test.csv')
```

Viewing sample of the data set

In [3]:

```
df.head()
```

Out[3]:

	ID	Target	empType	education	age	gender	maritalStatus	netMonthlyIncome	loanType	loanAmt	...	bureauScoreCardVer
0	843731420000192	Declined	Self Employed	OTHERS	30	2	Married	NaN	TW	92000	...	10.0
1	843731420000191	Declined	NaN	NaN	30	2	NaN	NaN	TW	92000	...	10.0
2	843731420000193	Declined	NaN	NaN	54	2	NaN	NaN	TW	90000	...	10.0
3	864141420000025	Declined	NaN	NaN	25	1	NaN	NaN	TW	76585	...	10.0
4	864141420000024	Declined	Self Employed	GRADUATE	25	1	Married	NaN	TW	76585	...	10.0

5 rows × 24 columns

Info of the data set

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 922 entries, 0 to 921
Data columns (total 24 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   ID                   922 non-null    object 
 1   Target               922 non-null    object 
 2   empType              531 non-null    object 
 3   education            530 non-null    object 
 4   age                  922 non-null    int64  
 5   gender               922 non-null    int64  
 6   maritalStatus        683 non-null    object 
 7   netMonthlyIncome     580 non-null    float64
 8   loanType             922 non-null    object 
 9   loanAmt              922 non-null    int64  
 10  loanTenure           922 non-null    int64  
 11  bureauTrackId        922 non-null    int64  
 12  bureauName           922 non-null    object 
 13  bureauScore          920 non-null    float64
 14  bureauScoreCardVer   920 non-null    float64
 15  bureauScoreCardName  920 non-null    float64
 16  bureauScoreCardDate  920 non-null    float64
 17  bureauScoreName      920 non-null    object 
 18  addrsCategory        922 non-null    int64  
 19  pinCode              922 non-null    int64  
 20  addrsDateReported    920 non-null    float64
 21  addrsStateCode       922 non-null    int64  
 22  phoneType            909 non-null    float64
 23  idType               884 non-null    object 
dtypes: float64(7), int64(8), object(9)
memory usage: 173.0+ KB
```

Shape of the data set

In [5]:

```
df.shape
```

Out[5]:

```
(922, 24)
```

Dropping id column

In [6]:

```
to_drop = ['ID', 'bureauTrackId', 'bureauScoreCardDate', 'addrsDateReported']

df.drop(to_drop, inplace = True, axis = 1)
df.head(5)
```

Out[6]:

	Target	empType	education	age	gender	maritalStatus	netMonthlyIncome	loanType	loanAmt	loanTenure	bureauName	bureauScore	bure
0	Declined	Self Employed	OTHERS	30	2	Married	NaN	TW	92000	0	CIBIL	0.0	
1	Declined	NaN	NaN	30	2	NaN	NaN	TW	92000	0	CIBIL	557.0	
2	Declined	NaN	NaN	54	2	NaN	NaN	TW	90000	0	CIBIL	0.0	
3	Declined	NaN	NaN	25	1	NaN	NaN	TW	76585	0	CIBIL	603.0	
4	Declined	Self Employed	GRADUATE	25	1	Married	NaN	TW	76585	0	CIBIL	603.0	

Describe

In [7]:

```
df.describe().T
```

Out[7]:

	count	mean	std	min	25%	50%	75%	max
age	922.0	33.300434	10.902279	0.0	26.0	32.0	40.00	67.0
gender	922.0	1.804772	0.396591	1.0	2.0	2.0	2.00	2.0
netMonthlyIncome	580.0	17492.175862	17355.600766	0.0	0.0	17500.0	25000.00	200000.0
loanAmt	922.0	36600.067245	38618.169939	50.0	20000.0	30000.0	45000.00	1016000.0
loanTenure	922.0	6.222343	13.400918	0.0	0.0	1.5	11.00	365.0
bureauScore	920.0	450.542391	348.737197	0.0	0.0	675.0	741.25	819.0
bureauScoreCardVer	920.0	10.000000	0.000000	10.0	10.0	10.0	10.00	10.0
bureauScoreCardName	920.0	8.000000	0.000000	8.0	8.0	8.0	8.00	8.0
addrsCategory	922.0	2.300434	0.922502	1.0	2.0	2.0	3.00	4.0
pinCode	922.0	581734.913232	130674.410888	110041.0	560043.5	608201.0	638657.00	852221.0
addrsStateCode	922.0	27.828633	7.203941	2.0	27.0	29.0	33.00	36.0
phoneType	909.0	1.067107	0.781748	0.0	1.0	1.0	1.00	3.0

No. of unique values in each column

In [8]:

```
nunq_df = df.nunique().to_frame()
nunq_df
```

Out[8]:

	0
Target	2
empType	4
education	8
age	45
gender	2
maritalStatus	2
netMonthlyIncome	76
loanType	4
loanAmt	126
loanTenure	19
bureauName	1
bureauScore	183
bureauScoreCardVer	1
bureauScoreCardName	1
bureauScoreName	1
addrsCategory	4
pinCode	777
addrsStateCode	22
phoneType	4
idType	7

In [9]:

```
nunq_df.shape
```

Out[9]:

```
(20, 1)
```

Removing columns having only 1 unique value

In [10]:

```
nunq_df[nunq_df[0] == 1].index
```

Out[10]:

```
Index(['bureauName', 'bureauScoreCardVer', 'bureauScoreCardName',
       'bureauScoreName'],
      dtype='object')
```

In [11]:

```
to_drop = nunq_df[nunq_df[0] == 1].index

df.drop(to_drop, inplace = True, axis = 1)
df.shape
```

Out[11]:

```
(922, 16)
```

In [12]:

```
nunq_df[nunq_df[0] == 2].index
```

Out[12]:

```
Index(['Target', 'gender', 'maritalStatus'], dtype='object')
```

Value counts for specific columns

In [13]:

```
df['Target'].value_counts()
```

Out[13]:

```
Declined    530
Approved    392
Name: Target, dtype: int64
```

In [14]:

```
df['empType'].value_counts()
```

Out[14]:

```
Self Employed     339
Salaried          174
Non-Government     12
Others              6
Name: empType, dtype: int64
```

In [15]:

```
df['education'].value_counts()
```

Out[15]:

```
12TH              153
GRADUATE          149
SSC                72
UNDER GRADUATE     71
OTHERS             62
POST-GRADUATE      16
PROFESSIONAL        6
DOCTORATE           1
Name: education, dtype: int64
```

In [16]:

```
df['gender'].value_counts()
```

Out[16]:

```
2    742
1    180
Name: gender, dtype: int64
```

In [17]:

```
df['maritalStatus'].value_counts()
```

Out[17]:

```
Married    376
Single     307
Name: maritalStatus, dtype: int64
```

In [18]:

```
df['loanType'].value_counts()
```

Out[18]:

```
CDL    524
DPL    281
TW     116
AL       1
Name: loanType, dtype: int64
```

In [19]:

```
df['addrsCategory'].value_counts()
```

Out[19]:

```
2    508
4    151
1    144
3    119
Name: addrsCategory, dtype: int64
```

In [20]:

```
df['addrsStateCode'].value_counts()
```

Out[20]:

```
33    367
29    143
19    113
27     92
32     58
28     39
36     16
9      15
23     11
6      10
20      9
3       9
24      7
21      7
8       5
10      4
5       4
34      3
7       3
4       3
18      2
2       2
Name: addrsStateCode, dtype: int64
```

In [21]:

```
df['phoneType'].value_counts()
```

Out[21]:

```
1.0    635
0.0    154
3.0     95
2.0     25
Name: phoneType, dtype: int64
```

In [22]:

```
df['idType'].value_counts()
```

Out[22]:

```
1        700
3        132
6         28
4         18
5          4
2          1
EMAIL      1
Name: idType, dtype: int64
```

Dropping duplicate rows

In [23]:

```
df.count()     # Used to count the number of rows
```

Out[23]:

```
Target              922
empType             531
education           530
age                 922
gender              922
maritalStatus       683
netMonthlyIncome    580
loanType            922
loanAmt             922
loanTenure          922
bureauScore         920
addrsCategory       922
pinCode             922
addrsStateCode      922
phoneType           909
idType              884
dtype: int64
```

In [24]:

```
duplicate_rows_df = df[df.duplicated()]
print("number of duplicate rows: ", duplicate_rows_df.shape)
```

```
number of duplicate rows:  (1, 16)
```

In [25]:

```
df = df.drop_duplicates()
```

In [26]:

```
df.count()     # Used to count the number of rows
```

Out[26]:

```
Target              921
empType             531
education           530
age                 921
gender              921
maritalStatus       682
netMonthlyIncome    579
loanType            921
loanAmt             921
loanTenure          921
bureauScore         919
addrsCategory       921
pinCode             921
addrsStateCode      921
phoneType           908
idType              883
dtype: int64
```

Checking for any null values

In [27]:

```
df.isnull().sum()
```

Out[27]:

```
Target                0
empType             390
education           391
age                   0
gender                0
maritalStatus       239
netMonthlyIncome    342
loanType              0
loanAmt               0
loanTenure            0
bureauScore           2
addrsCategory         0
pinCode               0
addrsStateCode        0
phoneType            13
idType               38
dtype: int64
```

Handling Missing or Null values

In [28]:

```
df['empType'].fillna('OTHERS', inplace=True)
df['education'].fillna('OTHERS', inplace=True)
```

```
df['maritalStatus'].fillna('missing', inplace=True)
df['idType'].fillna('missing', inplace=True)

df['bureauScore'].fillna(value=-1, inplace=True)
df['phoneType'].fillna(value=0.0, inplace=True)
df['netMonthlyIncome'].fillna(df['netMonthlyIncome'].median(), inplace=True)

df.tail()
```

Out[28]:

	Target	empType	education	age	gender	maritalStatus	netMonthlyIncome	loanType	loanAmt	loanTenure	bureauScore	addrsCatego
917	Declined	OTHERS	OTHERS	21	2	missing	17500.0	TW	80000	0	724.0	
918	Declined	OTHERS	OTHERS	36	2	missing	17500.0	TW	74000	0	627.0	
919	Declined	OTHERS	OTHERS	46	2	missing	17500.0	TW	70000	0	786.0	
920	Declined	OTHERS	PROFESSIONAL	0	2	Married	0.0	AL	1016000	0	-1.0	
921	Declined	OTHERS	OTHERS	29	2	Single	0.0	DPL	30000	0	-1.0	

In [29]:

```
df.isnull().sum()
```

Out[29]:

```
Target              0
empType             0
education           0
age                 0
gender              0
maritalStatus       0
netMonthlyIncome    0
loanType            0
loanAmt             0
loanTenure          0
bureauScore         0
addrsCategory       0
pinCode             0
addrsStateCode      0
phoneType           0
idType              0
dtype: int64
```

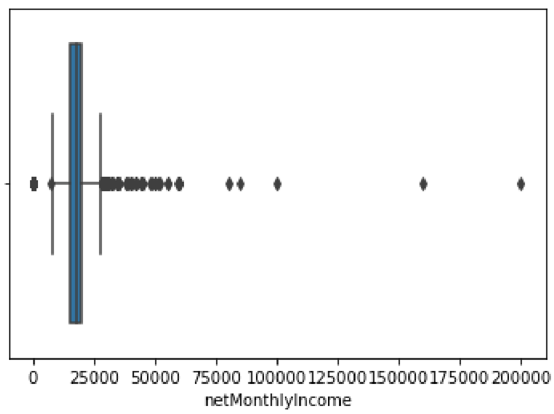
Detecting and handling outliers

In [30]:

```
sns.boxplot(x=df['netMonthlyIncome'])
```

Out[30]:

```
<AxesSubplot:xlabel='netMonthlyIncome'>
```

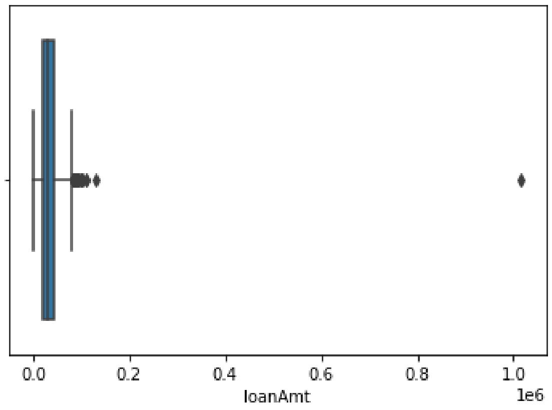


In [31]:

```
sns.boxplot(x=df['loanAmt'])
```

Out[31]:

```
<AxesSubplot:xlabel='loanAmt'>
```

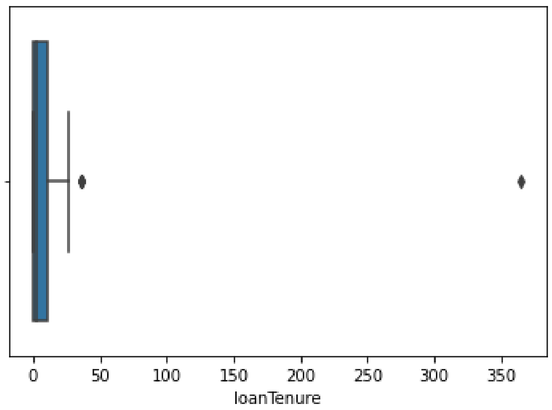


In [32]:

```
sns.boxplot(x=df['loanTenure'])
```

Out[32]:

```
<AxesSubplot:xlabel='loanTenure'>
```



In [33]:

```
df.shape
```

Out[33]:

```
(921, 16)
```

In [34]:

```
out_df = df[['netMonthlyIncome', 'loanAmt']]
out_df.head()
```

Out[34]:

	netMonthlyIncome	loanAmt
0	17500.0	92000
1	17500.0	92000
2	17500.0	90000
3	17500.0	76585
4	17500.0	76585

Q1 = out_df.quantile(0.25) Q3 = out_df.quantile(0.75) IQR = Q3 - Q1 print(IQR)out_df = out_df[~((out_df < (Q1 - 1.5 * IQR)) |(out_df > (Q3 + 1.5 * IQR))).any(axis=1)] out_df.shape

3.) Feature Engineering

Separating variables

In [35]:

```
df.shape
```

Out[35]:

```
(921, 16)
```

In [36]:

```
nominal_cols = ['empType', 'maritalStatus', 'loanType', 'idType']
print('NOMINAL COLUMNS:', nominal_cols)
```

```
num_cols = ['age', 'netMonthlyIncome', 'loanAmt', 'loanTenure', 'bureauScore']
print('NUMERICAL COLUMNS:', num_cols)

# one-hot encoding > gender, maritalStatus, idType, loanType, empType
# ordinal > education
```

```
NOMINAL COLUMNS: ['empType', 'maritalStatus', 'loanType', 'idType']
NUMERICAL COLUMNS: ['age', 'netMonthlyIncome', 'loanAmt', 'loanTenure', 'bureauScore']
```

Label encoding target variable

In [37]:

```
replace_map = {'Target': {'Approved': 1, 'Declined': 0}}
df.replace(replace_map, inplace=True)
```

Label encoding ordinal variable

In [38]:

```
replace_map = {'education': {'OTHERS':1, 'SSC':2, '12TH':3, 'UNDER GRADUATE':4, 'GRADUATE':5, 'POST-GRADUA
df.replace(replace_map, inplace=True)
```

One-hot encoding nominal variables

In [39]:

```
final_df = pd.get_dummies(df, columns=nominal_cols, drop_first=False)
```

In [40]:

```
final_df.head()
```

Out[40]:

	Target	education	age	gender	netMonthlyIncome	loanAmt	loanTenure	bureauScore	addrsCategory	pinCode	...	loanType_DPL	loanType_T
0	0	1	30	2	17500.0	92000	0	0.0	2	852221	...	0	
1	0	1	30	2	17500.0	92000	0	557.0	2	852221	...	0	
2	0	1	54	2	17500.0	90000	0	0.0	2	852106	...	0	
3	0	1	25	1	17500.0	76585	0	603.0	2	833201	...	0	
4	0	5	25	1	17500.0	76585	0	603.0	2	833201	...	0	

5 rows × 32 columns

In [41]:

```
final_df.shape
```

Out[41]:

```
(921, 32)
```

Log transforming skew variables

In [42]:

```
num_cols = final_df.select_dtypes('number').columns.values
print(num_cols)
```

```
['Target' 'education' 'age' 'gender' 'netMonthlyIncome' 'loanAmt'
 'loanTenure' 'bureauScore' 'addrsCategory' 'pinCode' 'addrsStateCode'
 'phoneType' 'empType_Non-Government' 'empType_OTHERS' 'empType_Others'
 'empType_Salaried' 'empType_Self Employed' 'maritalStatus_Married'
 'maritalStatus_Single' 'maritalStatus_missing' 'loanType_AL'
 'loanType_CDL' 'loanType_DPL' 'loanType_TW' 'idType_1' 'idType_2'
 'idType_3' 'idType_4' 'idType_5' 'idType_6' 'idType_EMAIL'
 'idType_missing']
```

In [43]:

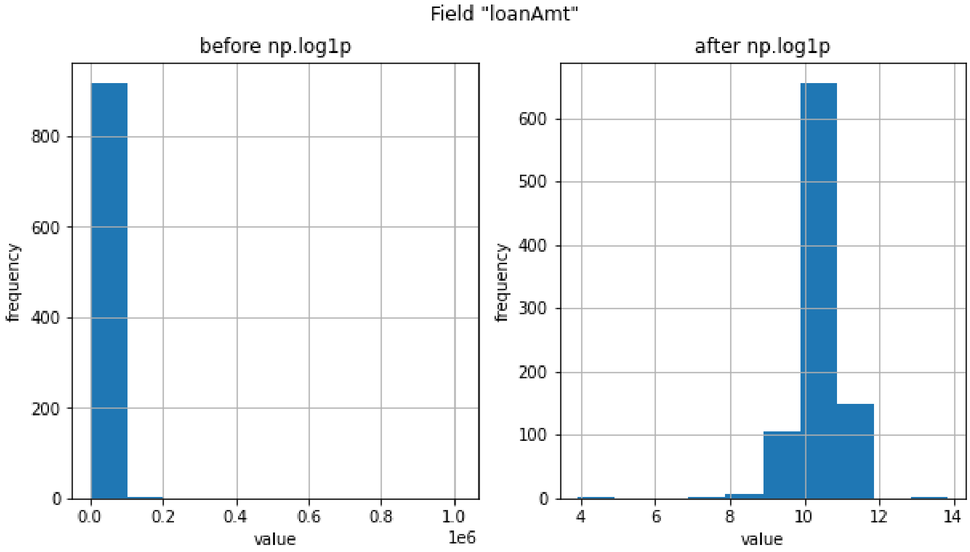
```
skew_vals = final_df[num_cols].skew()

skew_limit = 0.75
skew_cols = (skew_vals.
             sort_values(ascending=False)
             .to_frame()
             .rename(columns={0:'Skew'})
             .query('abs(Skew) > {}'.format(skew_limit)))
skew_cols
```

Out[43]:

	Skew
loanType_AL	30.347982
idType_2	30.347982
idType_EMAIL	30.347982
loanTenure	20.923492
loanAmt	17.956410
idType_5	15.099561
empType_Others	12.288134
empType_Non-Government	8.602568
idType_4	6.952986
idType_6	5.479231
idType_missing	4.620540
netMonthlyIncome	4.402677
loanType_TW	2.258398
idType_3	2.039145
empType_Salaried	1.591944
phoneType	1.182806
maritalStatus_missing	1.099058
loanType_DPL	0.853508
education	0.793274
idType_1	-1.212862
pinCode	-1.438734
gender	-1.538600
addrsStateCode	-1.676700

In [44]:

```
# Choose a field
field = "loanAmt"

# Create two "subplots" and a "figure" using matplotlib
fig, (ax_before, ax_after) = plt.subplots(1, 2, figsize=(10, 5))

# Create a histogram on the "ax_before" subplot
df[field].hist(ax=ax_before)

# Apply a log transformation (numpy syntax) to this column
df[field].apply(np.log1p).hist(ax=ax_after)

# Formatting of titles etc. for each subplot
ax_before.set(title='before np.log1p', ylabel='frequency', xlabel='value')
ax_after.set(title='after np.log1p', ylabel='frequency', xlabel='value')
fig.suptitle('Field "{}"'.format(field));
```



In [45]:

```
# Perform the skew transformation:

for col in skew_cols.index.values:
    final_df[col] = final_df[col].apply(np.log1p)
```

for col in skew_cols.index.values: final_df[col] = boxcox(final_df[col])

In [46]:

```
final_df.head()
```

Out[46]:

	Target	education	age	gender	netMonthlyIncome	loanAmt	loanTenure	bureauScore	addrsCategory	pinCode	...	loanType_DPL	loanT
0	0	0.693147	30	1.098612	9.770013	11.429555	0.0	0.0	2	13.655602	...	0.0	(
1	0	0.693147	30	1.098612	9.770013	11.429555	0.0	557.0	2	13.655602	...	0.0	(
2	0	0.693147	54	1.098612	9.770013	11.407576	0.0	0.0	2	13.655467	...	0.0	(
3	0	0.693147	25	0.693147	9.770013	11.246170	0.0	603.0	2	13.633031	...	0.0	(
4	0	1.791759	25	0.693147	9.770013	11.246170	0.0	603.0	2	13.633031	...	0.0	(

5 rows × 32 columns

Separating target and features

In [47]:

```
y_col = "Target"

X = final_df.drop(y_col, axis=1)
y = final_df[y_col]
```

Normalization of features

normalized_X = preprocessing.normalize(X)

Restructuring the dataframe

index_values = X.index.values column_values = X.columns.values X = pd.DataFrame(data = normalized_X, index = index_values, columns = column_values)

MinMaxScaler

X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0)) X_scaled = X_std * (max - min) + min

In [69]:

```
scaler = MinMaxScaler()
minMax_X = scaler.fit_transform(X)
```

Restructuring the dataframe

In [70]:

```
index_values = X.index.values
column_values = X.columns.values
```

```
X = pd.DataFrame(data = minMax_X,
                 index = index_values,
                 columns = column_values)
```

In [71]:

```
X.head()
```

Out[71]:

	education	age	gender	netMonthlyIncome	loanAmt	loanTenure	bureauScore	addrsCategory	pinCode	addrsStateCode	...	loanType_D
0	0.000000	0.447761	1.0	0.800422	0.757380	0.0	0.001220	0.333333	1.000000	0.517168	...	
1	0.000000	0.447761	1.0	0.800422	0.757380	0.0	0.680488	0.333333	1.000000	0.517168	...	
2	0.000000	0.805970	1.0	0.800422	0.755160	0.0	0.001220	0.333333	0.999934	0.517168	...	
3	0.000000	0.373134	0.0	0.800422	0.738856	0.0	0.736585	0.333333	0.988974	0.774552	...	
4	0.730423	0.373134	0.0	0.800422	0.738856	0.0	0.736585	0.333333	0.988974	0.774552	...	

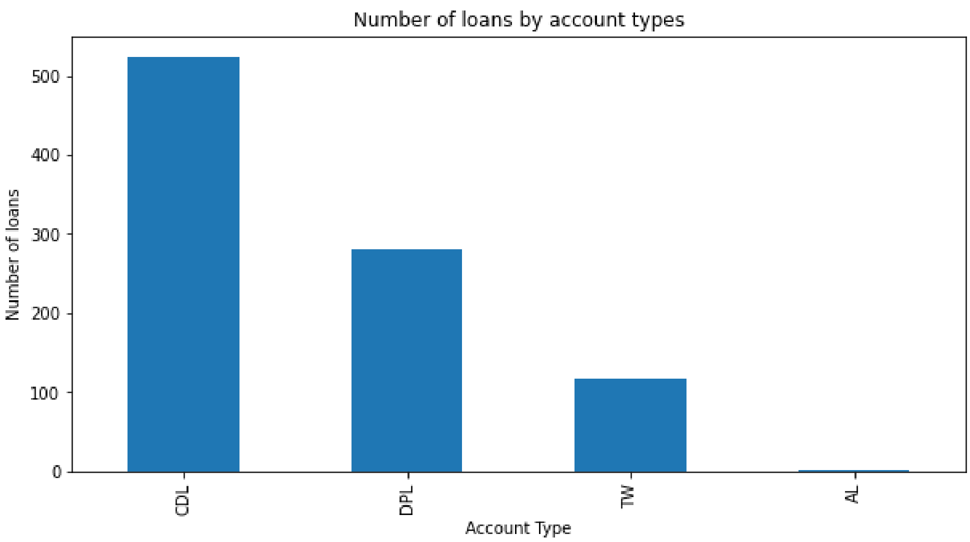
5 rows × 31 columns

4.) Data Visualization

sns.pairplot(df)

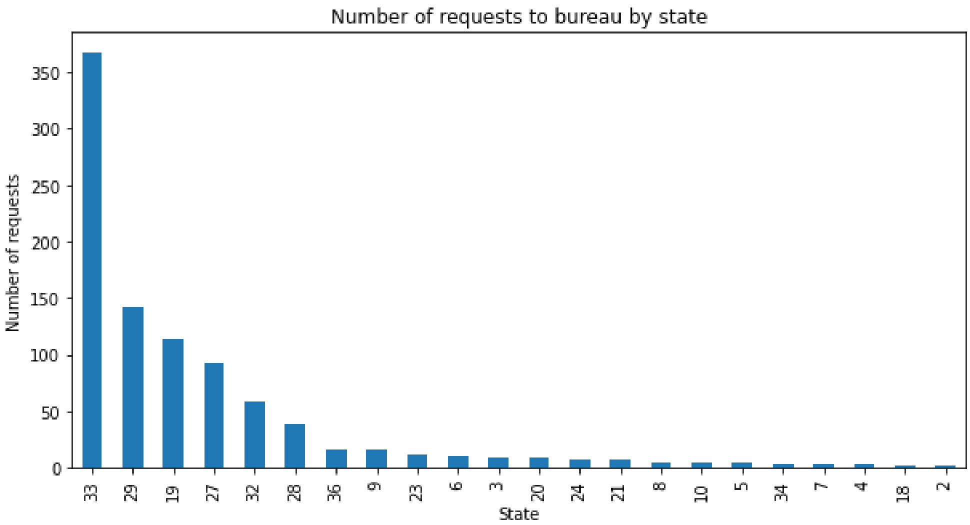
In [72]:

```
df.loanType.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of loans by account types")
plt.ylabel('Number of loans')
plt.xlabel('Account Type');
```



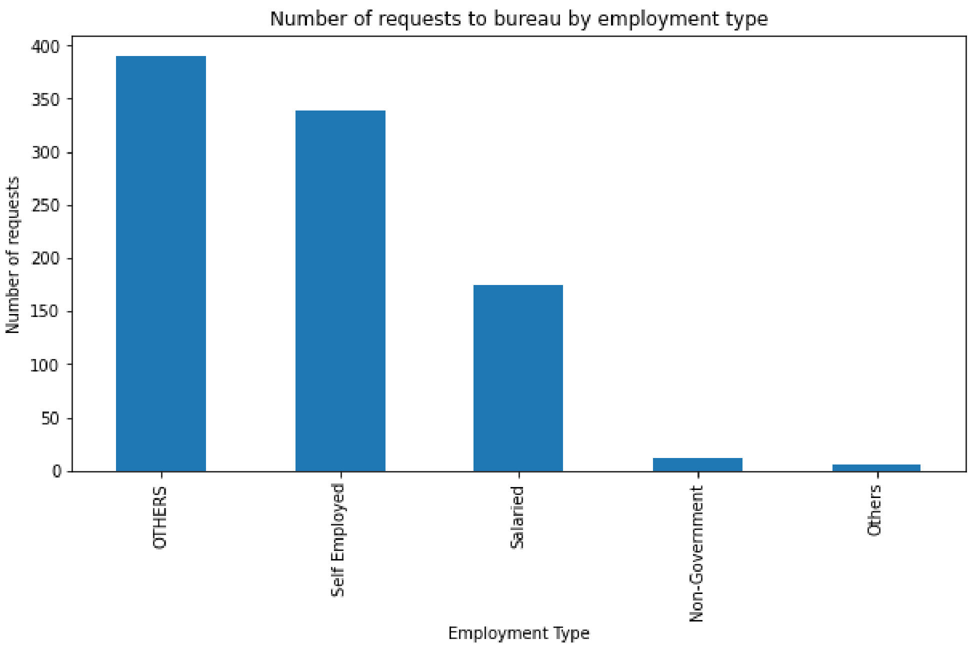
In [73]:

```
df.addrsStateCode.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of requests to bureau by state")
plt.ylabel('Number of requests')
plt.xlabel('State');
```



In [74]:

```
df.empType.value_counts().nlargest(40).plot(kind='bar', figsize=(10,5))
plt.title("Number of requests to bureau by employment type")
plt.ylabel('Number of requests')
plt.xlabel('Employment Type');
```

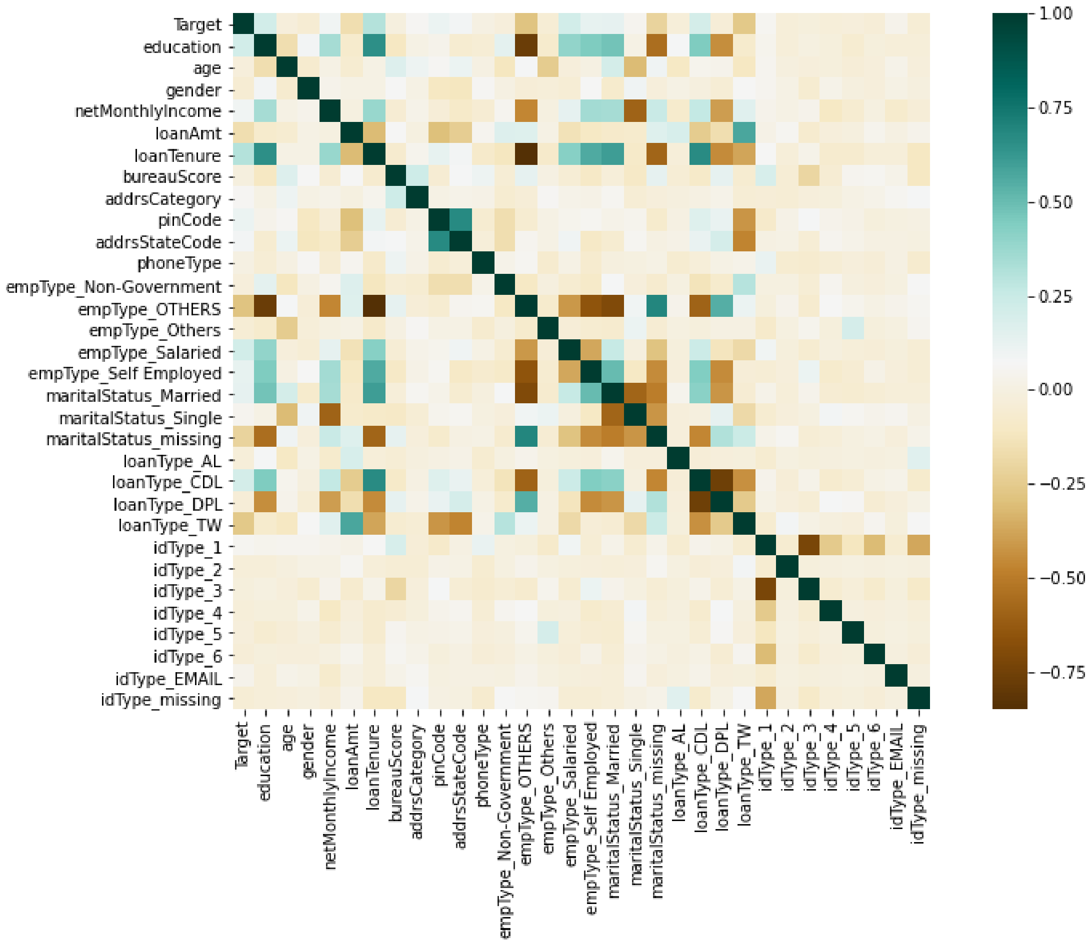


In [75]:

```
plt.figure(figsize=(14,8))
c= final_df.corr()
sns.heatmap(c,cmap="BrBG", square=True)
```

Out[75]:

```
<AxesSubplot:>
```



5.) Feature Selection

1. Univariate Selection

In [76]:

```python
#apply SelectKBest class to extract top 5 best features
bestfeatures = SelectKBest(score_func=chi2, k=6)
fit = bestfeatures.fit(X,y)
dfscores = pd.DataFrame(fit.scores_)
dfcolumns = pd.DataFrame(X.columns)
```

In [77]:

```python
#concat two dataframes for better visualization
featureScores = pd.concat([dfcolumns, dfscores], axis=1)
featureScores.columns = ['Specs', 'Score']  #naming the dataframe columns
print(featureScores.nlargest(10, 'Score'))  #print best features
```

```
                   Specs      Score
22           loanType_TW  54.664298
12        empType_OTHERS  42.952104
14      empType_Salaried  35.649273
18  maritalStatus_missing  29.795860
5             loanTenure  19.090505
20          loanType_CDL  17.224244
0              education  14.619018
16  maritalStatus_Married  10.431314
15  empType_Self Employed   9.267360
30          idType_missing   1.875173
```

In [78]:

```python
# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores

model_eval = LogisticRegression()
```

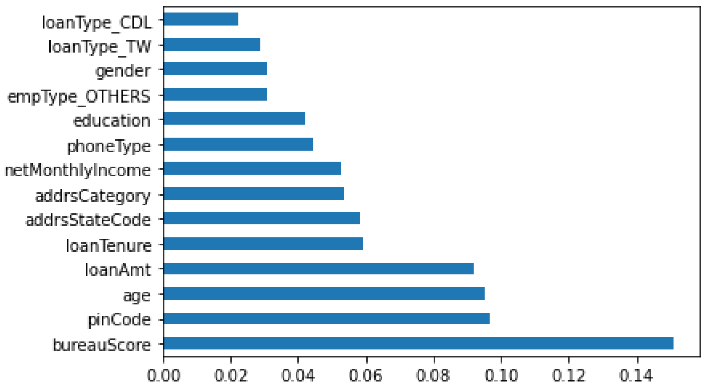
```
results = list()
for i in range(1,X.shape[1]+1):
        scores = evaluate_model(model_eval, X[featureScores.nlargest(i, 'Score')['Specs']], y)
        results.append(scores)
        print('> %s) %.3f (%.3f)' % (i, mean(scores), std(scores)))
```

```
> 1) 0.574 (0.004)
> 2) 0.660 (0.042)
> 3) 0.660 (0.042)
> 4) 0.660 (0.042)
> 5) 0.660 (0.042)
> 6) 0.660 (0.042)
> 7) 0.660 (0.042)
> 8) 0.648 (0.038)
> 9) 0.650 (0.038)
> 10) 0.650 (0.041)
> 11) 0.648 (0.038)
> 12) 0.648 (0.038)
> 13) 0.648 (0.038)
> 14) 0.646 (0.037)
> 15) 0.646 (0.041)
> 16) 0.657 (0.044)
> 17) 0.656 (0.043)
> 18) 0.657 (0.047)
> 19) 0.657 (0.047)
> 20) 0.657 (0.047)
> 21) 0.657 (0.044)
> 22) 0.656 (0.046)
> 23) 0.667 (0.033)
> 24) 0.667 (0.034)
> 25) 0.667 (0.037)
> 26) 0.669 (0.033)
> 27) 0.669 (0.033)
> 28) 0.668 (0.033)
> 29) 0.673 (0.033)
> 30) 0.668 (0.034)
> 31) 0.671 (0.035)
```

2. Feature Importance

In [79]:

```
from sklearn.ensemble import ExtraTreesClassifier

model_fi = ExtraTreesClassifier()
model_fi.fit(X,y)
#print(model.feature_importances_) #use inbuilt class feature_importances of tree based classifiers

#plot graph of feature importances for better visualization
feat_importances = pd.Series(model_fi.feature_importances_, index=X.columns)
feat_importances.sort_values(ascending=True).nlargest(14).plot(kind='barh')
plt.show()
```



In [80]:

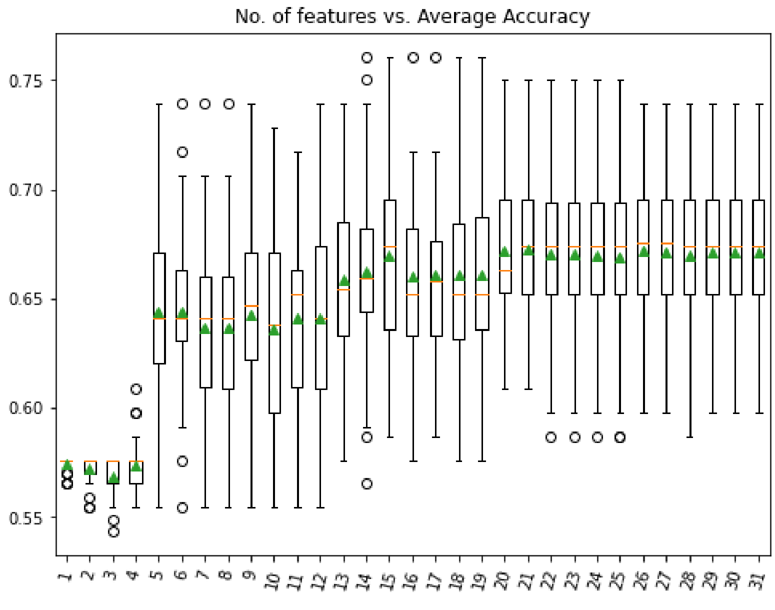
```
# evaluate a given model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores

model_eval = LogisticRegression()

results = list()
```

```
for i in range(1,X.shape[1]+1):
        scores = evaluate_model(model_eval, X[feat_importances.nlargest(i).index], y)
        results.append(scores)
        print('> %s) %.3f (%.3f)' % (i, mean(scores), std(scores)))
```

```
> 1) 0.574 (0.004)
> 2) 0.572 (0.007)
> 3) 0.568 (0.010)
> 4) 0.573 (0.013)
> 5) 0.644 (0.041)
> 6) 0.644 (0.040)
> 7) 0.637 (0.040)
> 8) 0.637 (0.041)
> 9) 0.642 (0.042)
> 10) 0.636 (0.045)
> 11) 0.641 (0.040)
> 12) 0.641 (0.044)
> 13) 0.659 (0.039)
> 14) 0.662 (0.043)
> 15) 0.670 (0.046)
> 16) 0.660 (0.040)
> 17) 0.661 (0.041)
> 18) 0.661 (0.042)
> 19) 0.661 (0.041)
> 20) 0.672 (0.037)
> 21) 0.673 (0.035)
> 22) 0.670 (0.037)
> 23) 0.670 (0.037)
> 24) 0.670 (0.037)
> 25) 0.669 (0.038)
> 26) 0.672 (0.037)
> 27) 0.671 (0.036)
> 28) 0.670 (0.036)
> 29) 0.671 (0.035)
> 30) 0.671 (0.035)
> 31) 0.671 (0.035)
```

In [81]:

```
no_of_features = [str(i) for i in range(1,X.shape[1]+1)]
# plot model performance for comparison
plt.figure(figsize=(8,6))
plt.boxplot(results, labels=no_of_features, showmeans=True)
plt.xticks(rotation=75)
plt.title('No. of features vs. Average Accuracy')
plt.show()
```



3. Correlation Matrix with Heatmap

#get correlations of each features in dataset corrmat = gcr_data.corr() top_corr_features = corrmat.index plt.figure(figsize=(20,20)) #plot heat map g=sns.heatmap(gcr_data[top_corr_features].corr(), annot=False, cmap="RdYlGn")

4. Recursive Feature Elimination (RFE)

In [82]:

```
# get a list of models to evaluate
```

```
def get_models():
    models = dict()
    # lr
    rfe = RFE(estimator=LogisticRegression(), n_features_to_select=5)
    model = LogisticRegression()
    models['lr'] = Pipeline(steps=[('s',rfe),('m',model)])
    # perceptron
    rfe = RFE(estimator=Perceptron(), n_features_to_select=5)
    model = LogisticRegression()
    models['per'] = Pipeline(steps=[('s',rfe),('m',model)])
    # cart
    rfe = RFE(estimator=DecisionTreeClassifier(), n_features_to_select=5)
    model = LogisticRegression()
    models['cart'] = Pipeline(steps=[('s',rfe),('m',model)])
    # rf
    rfe = RFE(estimator=RandomForestClassifier(), n_features_to_select=5)
    model = LogisticRegression()
    models['rf'] = Pipeline(steps=[('s',rfe),('m',model)])
    # gbm
    rfe = RFE(estimator=GradientBoostingClassifier(), n_features_to_select=5)
    model = LogisticRegression()
    models['gbm'] = Pipeline(steps=[('s',rfe),('m',model)])
    return models

# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
    return scores
```

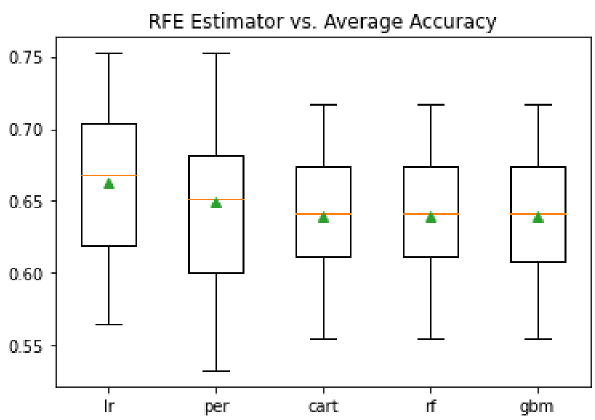
In [83]:

```
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
>lr 0.663 (0.051)
>per 0.649 (0.057)
>cart 0.640 (0.044)
>rf 0.640 (0.044)
>gbm 0.640 (0.044)
```

In [84]:

```
# plot model performance for comparison
plt.boxplot(results, labels=names, showmeans=True)
plt.title('RFE Estimator vs. Average Accuracy')
plt.show()
```



In [85]:

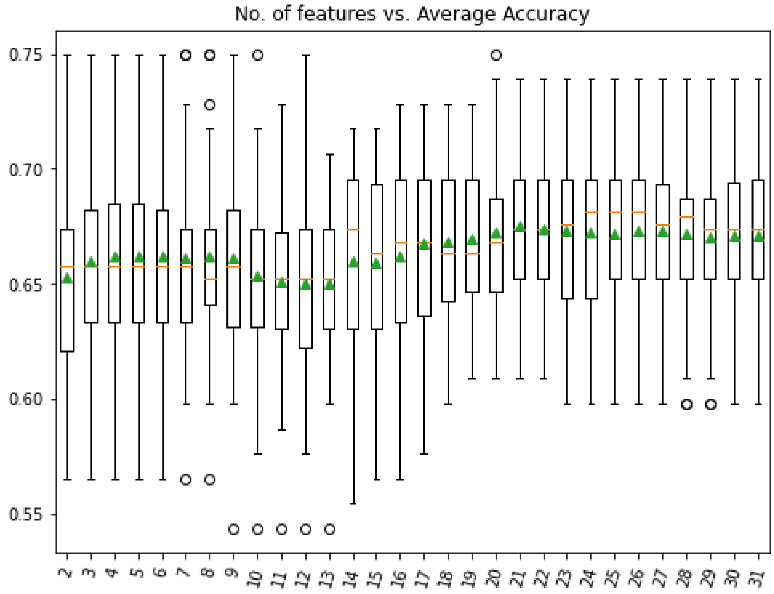
```
# get a list of models to evaluate
def get_models():
    models = dict()
    for i in range(2, X.shape[1]+1):
        rfe = RFE(estimator=LogisticRegression(), n_features_to_select=i)
        model = LogisticRegression()
        models[str(i)] = Pipeline(steps=[('s',rfe),('m',model)])
    return models
```

```python
# evaluate a give model using cross-validation
def evaluate_model(model, X, y):
    cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=42)
    scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1, error_score='raise')
    return scores

# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('> %s) %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
> 2) 0.653 (0.049)
> 3) 0.660 (0.042)
> 4) 0.662 (0.043)
> 5) 0.662 (0.043)
> 6) 0.662 (0.042)
> 7) 0.661 (0.042)
> 8) 0.662 (0.041)
> 9) 0.661 (0.044)
> 10) 0.654 (0.043)
> 11) 0.651 (0.041)
> 12) 0.650 (0.043)
> 13) 0.650 (0.037)
> 14) 0.660 (0.040)
> 15) 0.659 (0.039)
> 16) 0.662 (0.040)
> 17) 0.667 (0.036)
> 18) 0.668 (0.034)
> 19) 0.670 (0.030)
> 20) 0.672 (0.033)
> 21) 0.675 (0.033)
> 22) 0.674 (0.033)
> 23) 0.673 (0.034)
> 24) 0.672 (0.033)
> 25) 0.671 (0.034)
> 26) 0.673 (0.035)
> 27) 0.673 (0.036)
> 28) 0.672 (0.037)
> 29) 0.670 (0.035)
> 30) 0.671 (0.035)
> 31) 0.671 (0.035)
```

In [86]:

```python
# plot model performance for comparison
plt.figure(figsize=(8,6))
plt.boxplot(results, labels=names, showmeans=True)
plt.xticks(rotation=75)
plt.title('No. of features vs. Average Accuracy')
plt.show()
```



Suppressing any warnings

In [87]:

```
# Suppress warnings about too few trees from the early models
import warnings
warnings.filterwarnings("ignore", category=UserWarning)
warnings.filterwarnings("ignore", category=RuntimeWarning)
```

In [62]:

```
# define RFE
rfe = RFE(estimator=LogisticRegression(), n_features_to_select=22)
# fit RFE
rfe.fit(X, y)
# summarize all features
for i in range(X.shape[1]):
    print('Column: %d, Selected %s, Rank: %.3f' % (i, rfe.support_[i], rfe.ranking_[i]))
```

```
Column: 0, Selected True, Rank: 1.000
Column: 1, Selected False, Rank: 7.000
Column: 2, Selected True, Rank: 1.000
Column: 3, Selected False, Rank: 6.000
Column: 4, Selected True, Rank: 1.000
Column: 5, Selected True, Rank: 1.000
Column: 6, Selected False, Rank: 10.000
Column: 7, Selected True, Rank: 1.000
Column: 8, Selected False, Rank: 8.000
Column: 9, Selected True, Rank: 1.000
Column: 10, Selected True, Rank: 1.000
Column: 11, Selected True, Rank: 1.000
Column: 12, Selected True, Rank: 1.000
Column: 13, Selected True, Rank: 1.000
Column: 14, Selected True, Rank: 1.000
Column: 15, Selected True, Rank: 1.000
Column: 16, Selected True, Rank: 1.000
Column: 17, Selected False, Rank: 2.000
Column: 18, Selected True, Rank: 1.000
Column: 19, Selected False, Rank: 3.000
Column: 20, Selected True, Rank: 1.000
Column: 21, Selected True, Rank: 1.000
Column: 22, Selected True, Rank: 1.000
Column: 23, Selected True, Rank: 1.000
Column: 24, Selected False, Rank: 4.000
Column: 25, Selected False, Rank: 9.000
Column: 26, Selected False, Rank: 5.000
Column: 27, Selected True, Rank: 1.000
Column: 28, Selected True, Rank: 1.000
Column: 29, Selected True, Rank: 1.000
Column: 30, Selected True, Rank: 1.000
```

In [ ]: