_FineMining_FinalProject_Final_SubmissionVer

May 14, 2021

1 Classifying Peer-to-Peer (P2P) Loan Applicants to Reduce Default Risk

Team Fine Mining

Singer Li, Brandon Nguyen, Abhishek Yenumula, and Dan Trinh

Spring 2021 Semester

CIS 4321 - Dr. Koohikamali

Table of Contents

1.6.2 ROC Curve

Note: Table of Contents courtesy of nbextensions' toc2 addon.

[1]:
```
%matplotlib inline

import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
import seaborn as sns # Statistical data visualization https://seaborn.pydata.
↪org/
```

[2]:
```
#Functions for quick reference
'''
quickly get the value_counts for a series, check nulls, and describe it␣
↪numerical summary.
'''
def vc_null_summary(series):
    print(series.value_counts().sort_values(ascending=False))
    print(series.isnull().value_counts())
    print(series.describe())
```

1.1 General Information about Loan Dataset

About the Dataset - Available here. - From Bondora, a P2P lending platform that services several countries in the European Economic Area. - The dataset is a dataset of the platform's entire loan data, "not covered by data protection laws." Meaning that there are some records which today would be considered non-compliant with Europe's GDPR, because they have too much customer information. Like debt-to-income or number of dependents. - Because of the size of the data (164k records, 112 attributes), recommend looking at in Excel prior to beginning analysis on Python. - The documentation to find out about the variable information can be found here (https://api.bondora.com/doc/ResourceModel?modelName=PublicDatasetItem&v=1)

[3]:
```
#load the data; read CSV file.
#low_memory set to False because of DtypeWarning message
%time dataset = pd.read_csv("LoanData.csv", low_memory=False)
```

```
Wall time: 4.03 s
```

[4]:
```
dataset.shape
```

[4]:
```
(164045, 112)
```

[5]:
```
for i, col in enumerate(dataset.columns):
    print(f"{i} : {col}")
```

```
0 : ReportAsOfEOD
1 : LoanId
```

2 : LoanNumber
3 : ListedOnUTC
4 : BiddingStartedOn
5 : BidsPortfolioManager
6 : BidsApi
7 : BidsManual
8 : UserName
9 : NewCreditCustomer
10 : LoanApplicationStartedDate
11 : LoanDate
12 : ContractEndDate
13 : FirstPaymentDate
14 : MaturityDate_Original
15 : MaturityDate_Last
16 : ApplicationSignedHour
17 : ApplicationSignedWeekday
18 : VerificationType
19 : LanguageCode
20 : Age
21 : DateOfBirth
22 : Gender
23 : Country
24 : AppliedAmount
25 : Amount
26 : Interest
27 : LoanDuration
28 : MonthlyPayment
29 : County
30 : City
31 : UseOfLoan
32 : Education
33 : MaritalStatus
34 : NrOfDependants
35 : EmploymentStatus
36 : EmploymentDurationCurrentEmployer
37 : EmploymentPosition
38 : WorkExperience
39 : OccupationArea
40 : HomeOwnershipType
41 : IncomeFromPrincipalEmployer
42 : IncomeFromPension
43 : IncomeFromFamilyAllowance
44 : IncomeFromSocialWelfare
45 : IncomeFromLeavePay
46 : IncomeFromChildSupport
47 : IncomeOther
48 : IncomeTotal
49 : ExistingLiabilities

50 : LiabilitiesTotal
51 : RefinanceLiabilities
52 : DebtToIncome
53 : FreeCash
54 : MonthlyPaymentDay
55 : ActiveScheduleFirstPaymentReached
56 : PlannedPrincipalTillDate
57 : PlannedInterestTillDate
58 : LastPaymentOn
59 : CurrentDebtDaysPrimary
60 : DebtOccuredOn
61 : CurrentDebtDaysSecondary
62 : DebtOccuredOnForSecondary
63 : ExpectedLoss
64 : LossGivenDefault
65 : ExpectedReturn
66 : ProbabilityOfDefault
67 : DefaultDate
68 : PrincipalOverdueBySchedule
69 : PlannedPrincipalPostDefault
70 : PlannedInterestPostDefault
71 : EAD1
72 : EAD2
73 : PrincipalRecovery
74 : InterestRecovery
75 : RecoveryStage
76 : StageActiveSince
77 : ModelVersion
78 : Rating
79 : EL_V0
80 : Rating_V0
81 : EL_V1
82 : Rating_V1
83 : Rating_V2
84 : Status
85 : Restructured
86 : ActiveLateCategory
87 : WorseLateCategory
88 : CreditScoreEsMicroL
89 : CreditScoreEsEquifaxRisk
90 : CreditScoreFiAsiakasTietoRiskGrade
91 : CreditScoreEeMini
92 : PrincipalPaymentsMade
93 : InterestAndPenaltyPaymentsMade
94 : PrincipalWriteOffs
95 : InterestAndPenaltyWriteOffs
96 : PrincipalBalance
97 : InterestAndPenaltyBalance

```
98 : NoOfPreviousLoansBeforeLoan
99 : AmountOfPreviousLoansBeforeLoan
100 : PreviousRepaymentsBeforeLoan
101 : PreviousEarlyRepaymentsBefoleLoan
102 : PreviousEarlyRepaymentsCountBeforeLoan
103 : GracePeriodStart
104 : GracePeriodEnd
105 : NextPaymentDate
106 : NextPaymentNr
107 : NrOfScheduledPayments
108 : ReScheduledOn
109 : PrincipalDebtServicingCost
110 : InterestAndPenaltyDebtServicingCost
111 : ActiveLateLastPaymentCategory
```

[6]:
```
#Spelling mistake on Predictor 101: PreviousEarlyRepaymentsBefoleLoan
dataset.rename(columns={"PreviousEarlyRepaymentsBefoleLoan" :␣
 ↪"PreviousEarlyRepaymentsBeforeLoan"},
           inplace=True)
```

1.2 Preprocessing

1.2.1 Dimensional Reduction on the Dataset

Introduction and Reduction Strategy Because the dataset is a record of *every single loan* that has been made since the platform's inception, the dataset contains over 160,000 records and 112 predictors.

To reduce the dimensions, we considered the following: - Since we're trying to predict before the user is approved for the loan, we can only use information collected on the application. We cannot use information acquired the creation of the loan, such as operational status. Anyways, the contents of the application information available is listed here on Bondora's FAQ. * Personal information * Socio-demographic data * Employment information * Income data * Data on outstanding liabilities * Supporting documentation - After January 1, 2017, the Bondora platform made changes to their data collection methods in order to comply with the European Union's **General Data Protection Regulation** (GDPR) act. As a result, several attributes that would have been collected on the application prior to 1/1/2017 are now considered obsolete. This includes attributes like number of dependents or marital status. These will be dropped considering that this model is a proposed solution for the platform. Anyhow, even if we wanted to use these values, they are increasingly null after the date. - Guidance from various related studies, some even involving Bondora itself: * Lending Club Study * Bondora Study on Determinants of Successful Loan Apps * Chinese P2P Lending Platform Study ***

Applying the Whitelist After the first round of elimination, the variables that could be ***potentially*** kept are:

Note that not all variables will be used here. Some are only kept for exploratory analysis, such as UserName or LoanNumber. Moreover, further analysis is necessary to see which of these should be kept!

[7]:
```python
# Rather than excluding the variables from the original dataset,
# we can instead just use a whitelist approach.
whitelist = ["IncomeTotal", "LoanNumber", "UserName", "NewCreditCustomer",
             "VerificationType", "Age", "Gender", "Country", "AppliedAmount",
             "Interest", "LoanDuration", "Education",
             "EmploymentDurationCurrentEmployer", "ExistingLiabilities",
             "LiabilitiesTotal", "DefaultDate", "Status", "PrincipalBalance",
             "InterestAndPenaltyBalance", "NoOfPreviousLoansBeforeLoan",
             "AmountOfPreviousLoansBeforeLoan", "PreviousRepaymentsBeforeLoan",
             "PreviousEarlyRepaymentsBeforeLoan",
             "PreviousEarlyRepaymentsCountBeforeLoan"]

# We'll create this to
# save the original dataset
# in case we want to do exploratory analysis elsewhere.
df_original = dataset.loc[:, whitelist]
df = dataset.loc[:, whitelist]

df.shape
```

[7]: (164045, 24)

Let's actually drop some of the variables that we originally intended to keep for exploratory analysis.

[8]:
```python
exploratory_vars = ["LoanNumber", "UserName", "PrincipalBalance",␣
↪"InterestAndPenaltyBalance"]

df = df.drop(columns=exploratory_vars)
df.shape
```

[8]: (164045, 20)

Filtering 'current' loans We can remove the loans with a 'current' status. We have no idea if they'll default in the future. However, there are some loans that are 'current' but have defaulted (148 of them); chances are that they've renegotiated the debt or the loan terms. We'll keep those records, but consider them in the DEFAULT category later.

Otherwise, we'll remove all of the 'current' loans.

[9]:
```python
# get the index of those that are current but not defaulted
current_but_not_defaulted = df.loc[(df.Status =="Current") & (df.DefaultDate.
↪isnull())].index

df.loc[(df.Status =="Current") & (df.DefaultDate.isnull() != True)].index
```

[9]:
```
Int64Index([   754,   5248,   7557,   9160,   9162,  10332,  10369,  11185,
             11476,  13528,
            …
```

```
        132597, 133069, 133397, 137600, 139796, 141061, 143065, 143068,
        143270, 163964],
       dtype='int64', length=148)
```

[10]:
```python
df = df.drop(index=current_but_not_defaulted)
df.shape
```

[10]:
```
(113805, 20)
```

Computing New Attribute based on Default Date The next step is to classify any with a DEFAULT date.

Note the high default count… this is because Bondora considers any that have DEFAULTED as those who have missed payments on their loans for 60+ days. At that point, they initiate legal action to recover the principal. Nonetheless, hearing that their borrower has defaulted would probably concern investors and dissuade them from making further investments on the platform. Additionally, the recovery rate has been on the decline – we talk about this more in the milestone paper. So, it remains in our best interest to reduce this any way we can.

[11]:
```python
df.DefaultDate.isnull().value_counts()
```

[11]:
```
False    67321
True     46484
Name: DefaultDate, dtype: int64
```

[12]:
```python
# The Tilda here inverts a pandas Series containing booleans.
Default_Indicator_Series = ~df.DefaultDate.isnull()
Default_Indicator_Series.name = "Defaulted"
```

[13]:
```python
# Print a sample of the data.
df = pd.concat([df, Default_Indicator_Series], axis=1)
df.loc[:5, ["DefaultDate", "Defaulted"]]
```

[13]:

	DefaultDate	Defaulted
0	12-06-2017	True
1	2-19-2016	True
2	NaN	False
3	NaN	False
4	6-20-2017	True
5	2-26-2016	True

Now that we've classified them as a defaulted or not defaulted, we can remove the DefaultDate and Status from our dataset.

[14]:
```python
df.drop(columns=['Status', 'DefaultDate'], inplace = True)
df.shape
```

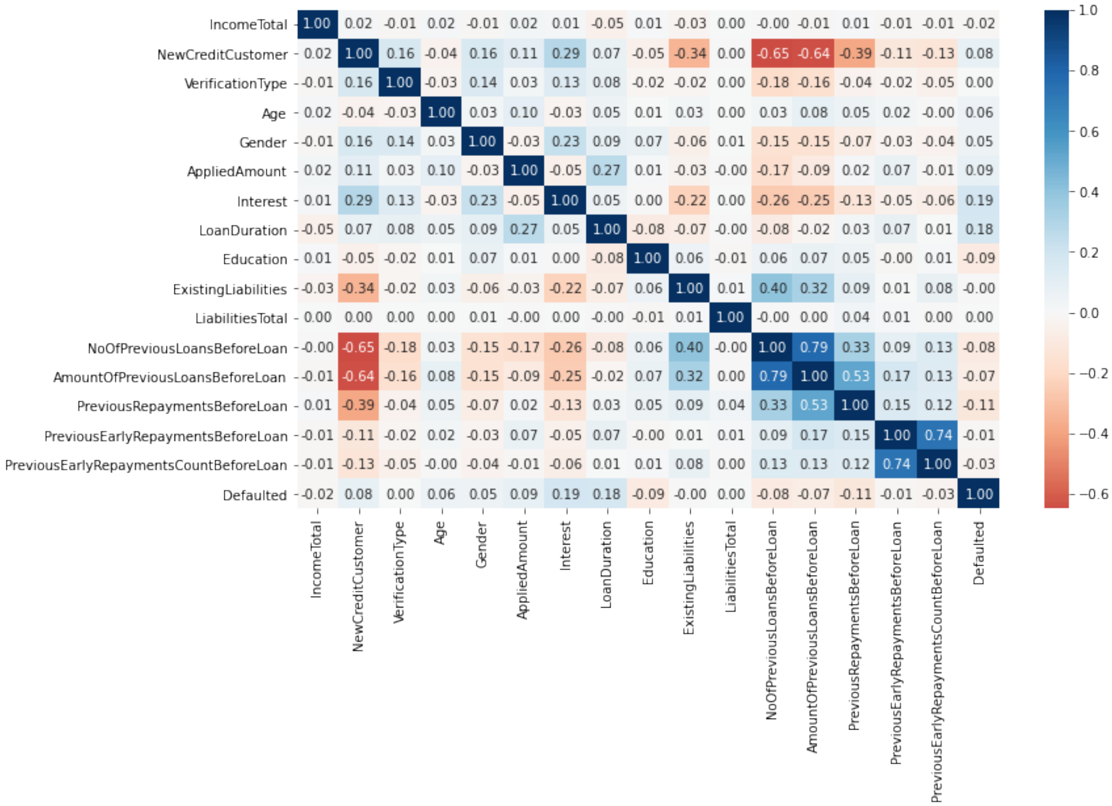
[14]:
```
(113805, 19)
```

[15]:
```python
# Look at the remaining columns.
print(df.dtypes)
```

```
IncomeTotal                               float64
NewCreditCustomer                            bool
VerificationType                          float64
Age                                         int64
Gender                                    float64
Country                                    object
AppliedAmount                             float64
Interest                                  float64
LoanDuration                                int64
Education                                 float64
EmploymentDurationCurrentEmployer          object
ExistingLiabilities                         int64
LiabilitiesTotal                          float64
NoOfPreviousLoansBeforeLoan                 int64
AmountOfPreviousLoansBeforeLoan           float64
PreviousRepaymentsBeforeLoan              float64
PreviousEarlyRepaymentsBeforeLoan         float64
PreviousEarlyRepaymentsCountBeforeLoan      int64
Defaulted                                    bool
dtype: object
```

Removing Correlated Attributes

[16]:
```python
# From Visualization Demo in week 4
import seaborn as sns # Statistical data visualization https://seaborn.pydata.
↪org/

corr = df.corr()

fig, ax = plt.subplots()
fig.set_size_inches(12, 7)
sns.heatmap(corr, annot=True, fmt=".2f", cmap="RdBu", center=0, ax=ax)

plt.show()
```



Some of the variables have informational overlap, which is conveyed by the correlation they have. We can remove them from the dataset. These are:

- NoOfPreviousLoansBeforeLoan: Number of previous loans
- AmountOfPreviousLoansBeforeLoan: Total value of said number of previous loans
- PreviousRepaymentsBeforeLoan: How much the borrower had repaid before the loan
- PreviousEarlyRepaymentsBeforeLoan: Previous early repaid loans before this loan
- PreviousEarlyRepaymentsCountBeforeLoan: Count of previous early repaid loans before this loan

Most likely, only PreviousEarlyRepaymentsCountBeforeLoan would be helpful. Intuitively, the more early repaid loans is more indicative of good behavior more than anything else.

Let's axe them all, save PreviousEarlyRepaymentsCount.

[17]:
```
df.drop(columns=["NoOfPreviousLoansBeforeLoan", 
 ↪"AmountOfPreviousLoansBeforeLoan",
                  "PreviousRepaymentsBeforeLoan", 
 ↪"PreviousEarlyRepaymentsBeforeLoan"],
        inplace=True)
df.shape
```

[17]:
```
(113805, 15)
```

Dealing with Liabilities Attributes, Computing `DebtToIncomeRatio`

[18]:
```
# Take a peak at some existingliabilities and liabilities total
df.sample(n=15).loc[:, ["ExistingLiabilities", "LiabilitiesTotal"]]
```

[18]:

	ExistingLiabilities	LiabilitiesTotal
24035	3	832.00
56223	0	0.00
28276	2	415.99
1954	0	0.00
49824	2	210.55
93926	0	0.00
8159	5	732.00
81837	4	760.00
17679	1	458.00
90514	0	0.00
120062	0	0.00
34318	2	406.53
20417	1	600.00
80138	2	200.00
111705	0	0.00

Let's drop ExistingLiabilities. The number of liabilities they have doesn't really matter when we look at that value in regard to the monthly debt expenses they have. What matters more is the monthly liability expenses that they have – and how significant their debt it is relative to their monthly income, so we'll create a new attribute for that – DebtToIncome, and then drop both `existingliabilities` and `liabilitiestotal`.

`DTI = Total Monthly Debt Payments / Total Income`

To further highlight the importance, we can see related studies on P2P lending such as those from Li et al. (2018), Gavurova et al. (2018), and Alomari & Fingerman (2017) retain some form of debt-to-income attribute in their dataset.

[19]:
```
# Round income to nearest Euro
df["IncomeTotal"] = df["IncomeTotal"].round()
# Ditto for liabilities.
df["LiabilitiesTotal"] = df["LiabilitiesTotal"].round()

# Compute new attribute.
df["DebtToIncomeRatio"] = df["LiabilitiesTotal"]/df["IncomeTotal"]

print(df.DebtToIncomeRatio.describe())
print("Uh oh, the describe() has issues. it's returning infinity and NaN!")

# Digging deeper, we see this results when income is equal to 0.
# Dividing by zero is either NaN or infinite.
bad_ratio_index = df.DebtToIncomeRatio.sort_values(ascending=False).head(25).
↪index
```

```
df.loc[bad_ratio_index, ["IncomeTotal", "LiabilitiesTotal",␣
↪"DebtToIncomeRatio", "Defaulted"]].head(10)
```

```
count    1.137700e+05
mean              inf
std               NaN
min      0.000000e+00
25%      5.145302e-02
50%      2.752960e-01
75%      5.080000e-01
max               inf
Name: DebtToIncomeRatio, dtype: float64
Uh oh, the describe() has issues. it's returning infinity and NaN!
```

[19]:

	IncomeTotal	LiabilitiesTotal	DebtToIncomeRatio	Defaulted
93982	0.0	1500.0	inf	False
2679	0.0	2303.0	inf	True
55692	0.0	49.0	inf	False
4095	0.0	150.0	inf	False
4001	0.0	166.0	inf	False
3912	0.0	197.0	inf	False
3084	0.0	2750.0	inf	False
2960	0.0	200.0	inf	False
2901	0.0	48.0	inf	False
2843	0.0	3295.0	inf	True

Anywhere where income is equal to 0, let us set the DebtToIncome ratio equal to liabilities. Though not exact, it still returns a high number that should alarm an analyst.

[20]:
```
df.loc[df[(df.IncomeTotal == 0)].index, "DebtToIncomeRatio"] = df.loc[df[(df.
↪IncomeTotal == 0)].index, "LiabilitiesTotal"]

df.loc[bad_ratio_index, ["IncomeTotal", "LiabilitiesTotal",␣
↪"DebtToIncomeRatio", "Defaulted"]].head(10)
```

[20]:

	IncomeTotal	LiabilitiesTotal	DebtToIncomeRatio	Defaulted
93982	0.0	1500.0	1500.0	False
2679	0.0	2303.0	2303.0	True
55692	0.0	49.0	49.0	False
4095	0.0	150.0	150.0	False
4001	0.0	166.0	166.0	False
3912	0.0	197.0	197.0	False
3084	0.0	2750.0	2750.0	False
2960	0.0	200.0	200.0	False
2901	0.0	48.0	48.0	False
2843	0.0	3295.0	3295.0	True

[21]:
```
# Now, we have an adequate measure of liabilities, so we can remove the two␣
↪initial liabilities columns.
df.drop(columns=["LiabilitiesTotal", "ExistingLiabilities"], inplace=True)
```

[22]:
```
print(df.shape)
print(df.dtypes)
```

```
(113805, 14)
IncomeTotal                                float64
NewCreditCustomer                             bool
VerificationType                           float64
Age                                          int64
Gender                                     float64
Country                                     object
AppliedAmount                              float64
Interest                                   float64
LoanDuration                                 int64
Education                                  float64
EmploymentDurationCurrentEmployer           object
PreviousEarlyRepaymentsCountBeforeLoan       int64
Defaulted                                     bool
DebtToIncomeRatio                          float64
dtype: object
```

Analyzing Whether to Keep `VerificationType` The belief is that customers who don't verify their income statements are less trustable, and therefore will likely default more. What does the data have to say about this?

Note that

```
"1.0" : "NotVerified"
"2.0" : "VerifiedByPhone"
"3.0" : "VerifiedByOtherDocument"
"4.0" : "VerifiedByBankStatement"
```

per Bondora's documentation

[23]:
```
# First, replace the numbers with the actual names of the verification status␣
↪to make it easier to read.
verificationDict = {
#"col" : "VerificationType",
1.0 : "NotVerified",
2.0 : "VerifiedByPhone",
3.0 : "VerifiedByOtherDocument",
4.0 : "VerifiedByBankStatement"
}

# Remove any that records where verification is marked as 0.0 -- these are␣
↪essentially null values.
```

```
df.drop(index=df[(df.VerificationType == 0.0) | (df.VerificationType.isnull())].
 ↪index, inplace=True)

# Replace numericals with the actual values
df.VerificationType.replace(to_replace=verificationDict, inplace=True)

# Convert to a category type
df.VerificationType = df.VerificationType.astype('category')

df.VerificationType.value_counts()
```

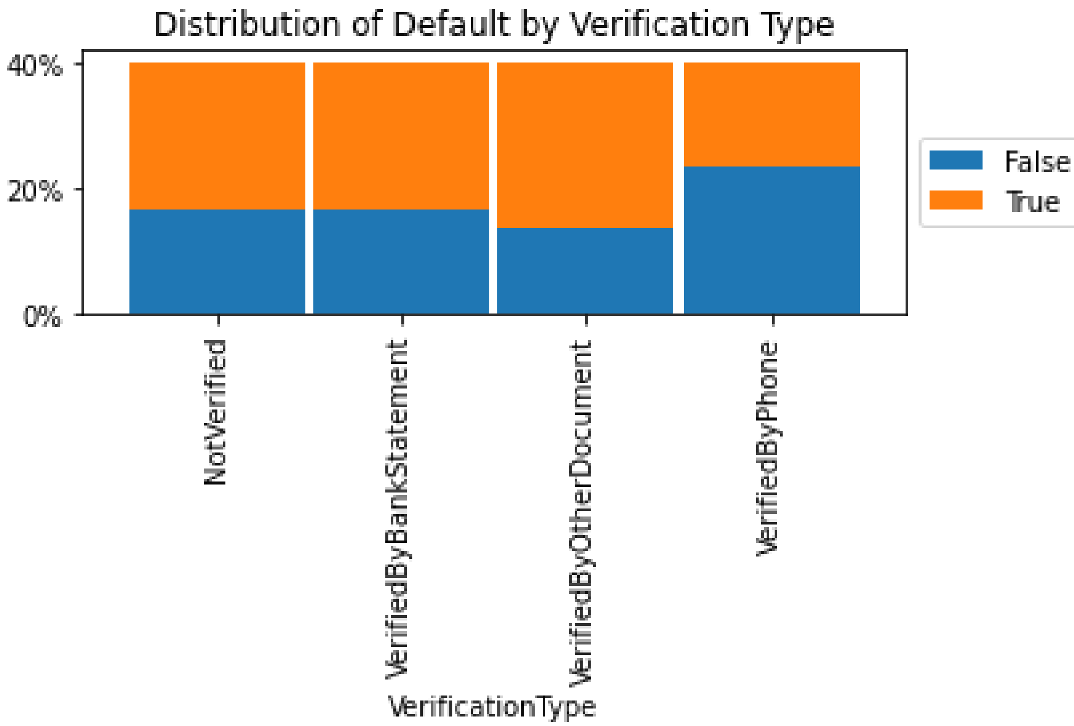
[23]:
```
VerifiedByBankStatement    67028
NotVerified                35960
VerifiedByOtherDocument     8936
VerifiedByPhone             1828
Name: VerificationType, dtype: int64
```

[24]:
```
# Create a cross table and then use a lambda function to compute the proportion␣
 ↪of the categories.
# when axis = 0 (vertical), that's the proportion of each default status over␣
 ↪each category
# when axis = 1 (horizontal), that's the proportion of each category over the␣
 ↪default status
tab = pd.crosstab(df.Defaulted, df.VerificationType)
prop_tab_vertical = tab.apply(lambda r: r/r.sum(), axis=0)
prop_tab_vertical
```

[24]:
```
VerificationType  NotVerified  VerifiedByBankStatement  \
Defaulted
False                  0.4104                 0.411127
True                   0.5896                 0.588873

VerificationType  VerifiedByOtherDocument  VerifiedByPhone
Defaulted
False                            0.344561         0.582057
True                             0.655439         0.417943
```

[25]:
```
ax = prop_tab_vertical.transpose().plot(kind='bar', stacked=True, width=0.95)
ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
plt.title('Distribution of Default by Verification Type')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

```
<ipython-input-25-9248df748e95>:2: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
```



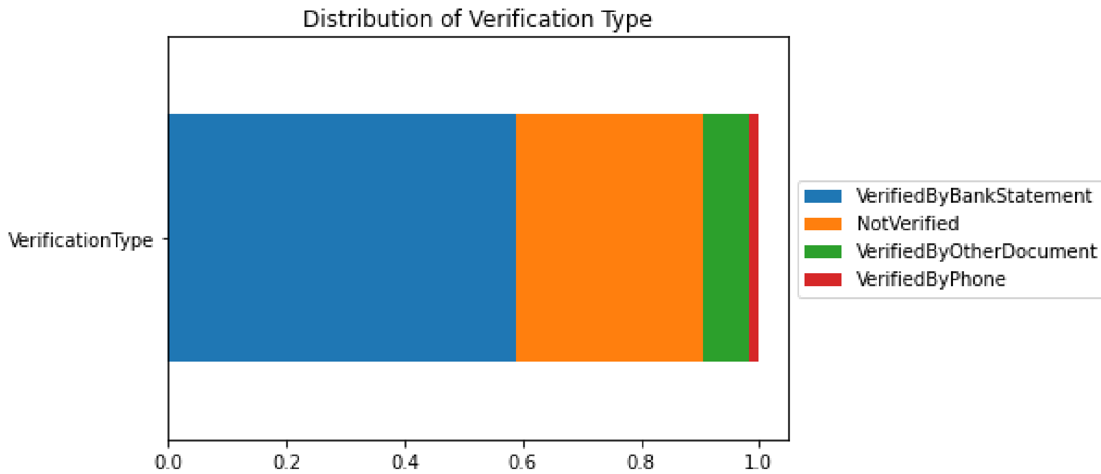
From the crosstab() and the generated graphic, the proportions appear about the same. Only `VerifiedByPhone` is a bit different. How many records in the dataframe are actually verified by phone, though?

[26]:
```python
verification_prop = df.VerificationType.value_counts(normalize=True)
verification_prop.map('{:.2%}'.format)
```

[26]:
```
VerifiedByBankStatement    58.92%
NotVerified                31.61%
VerifiedByOtherDocument     7.86%
VerifiedByPhone             1.61%
Name: VerificationType, dtype: object
```

It's a tiny minority – only 2% of the data.

[27]:
```python
# Display distribution graphically in a stacked single bar chart.
pd.DataFrame(verification_prop).T.plot.barh(stacked=True, width=0.8)
plt.title('Distribution of Verification Type')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))

plt.show()
```



Given the similar proportions in terms of default for each verification type, except for `verifiedByPhone`, which only makes up a small fraction of the dataset, it is likely safe to remove verification type from our dataset for the sake of reducing the noise that could impact our models.

[28]:
```
df.drop(columns="VerificationType", inplace=True)
```

Analyzing `Gender` This was on the cutting block already because of ethical issues and possible legal consequences if we discriminated loan applicants by gender. A quick Google search yields the existence of a gender equality law in the EU.

[29]:
```
# First, replace the numbers with the actual names of the gender status to make␣
↪it easier to read.
genderDict = {
"col" : "Gender",
"0.0" : "Male",
"1.0" : "Female",
"2.0" : "Unknown"
}

# Convert to string to be able to use the dictionary
df.Gender = df.Gender.astype('string')

# Replace numericals with the actual values
df.Gender.replace(to_replace=genderDict, inplace=True)

# Convert to a category type
df.Gender = df.Gender.astype('category')


df.Gender.value_counts(normalize=True).round(4) * 100
```

[29]:
```
Male       62.42
Female     28.23
Unknown     9.35
Name: Gender, dtype: float64
```

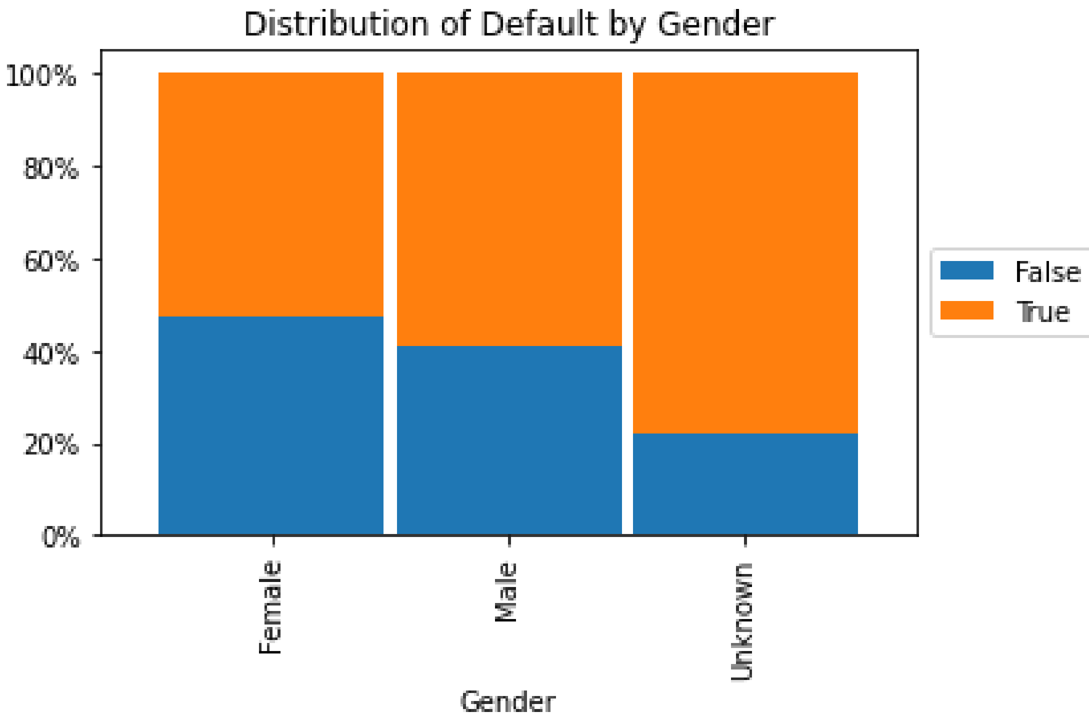
[30]:
```python
# Create a cross table and then use a lambda function to compute the proportion␣
↪of the categories.
# when axis = 0 (vertical), that's the proportion of each default status over␣
↪each category
# when axis = 1 (horizontal), that's the proportion of each category over the␣
↪default status
tab = pd.crosstab(df.Defaulted, df.Gender)
prop_tab_vertical = tab.apply(lambda r: r/r.sum(), axis=0)
prop_tab_vertical
```

[30]:

Gender	Female	Male	Unknown
Defaulted			
False	0.472614	0.407577	0.22009
True	0.527386	0.592423	0.77991

[31]:
```python
# Let's graph it out.
ax = prop_tab_vertical.transpose().plot(kind='bar', stacked=True, width=0.95)
ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
plt.title('Distribution of Default by Gender')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

```
<ipython-input-31-3e43c95871b7>:3: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
```



Interestingly, females default less than men. Unknown genders default the most though.

Anyways, we cannot keep gender because of European Union Directive 2004/113/EC mandating all member states adhere to the principle of equal treatment between men and women in the access to and supply of goods and services.

Drop gender.

[32]:
```
df.drop(columns="Gender", inplace=True)
```

Analyzing `Education` The goal is not to remove Education, but to see which categories can be combined.

[33]:
```
educationDict = {
"col": "Education",
"-1.0" : "N/A",
"0.0"  : "N/A",
"1.0"  : "Primary",
"2.0"  : "Basic",
"3.0"  : "Vocational",
"4.0"  : "Secondary",
"5.0"  : "Higher"
}

# Convert to string to be able to use the dictionary
```

```
df.Education = df.Education.astype('string')

# Replace numericals with the actual values
df.Education.replace(to_replace=educationDict, inplace=True)

# Convert to a category type
df.Education = df.Education.astype('category')

# Remove all records with N/A or null for education.
df.drop(df[(df.Education == "N/A") | (df.Education.isnull())].index,␣
↪inplace=True)

df.Education.value_counts(normalize=True).round(4) * 100
```

[33]:

```
Secondary     36.94
Higher        26.45
Vocational    22.47
Primary        8.98
Basic          5.16
N/A            0.00
Name: Education, dtype: float64
```

[34]:

```
# Create a cross table and then use a lambda function to compute the proportion␣
↪of the categories.
# when axis = 0 (vertical), that's the proportion of each default status over␣
↪each category
# when axis = 1 (horizontal), that's the proportion of each category over the␣
↪default status
tab = pd.crosstab(df.Defaulted, df.Education)
prop_tab_vertical = tab.apply(lambda r: r/r.sum(), axis=0)
prop_tab_vertical
```

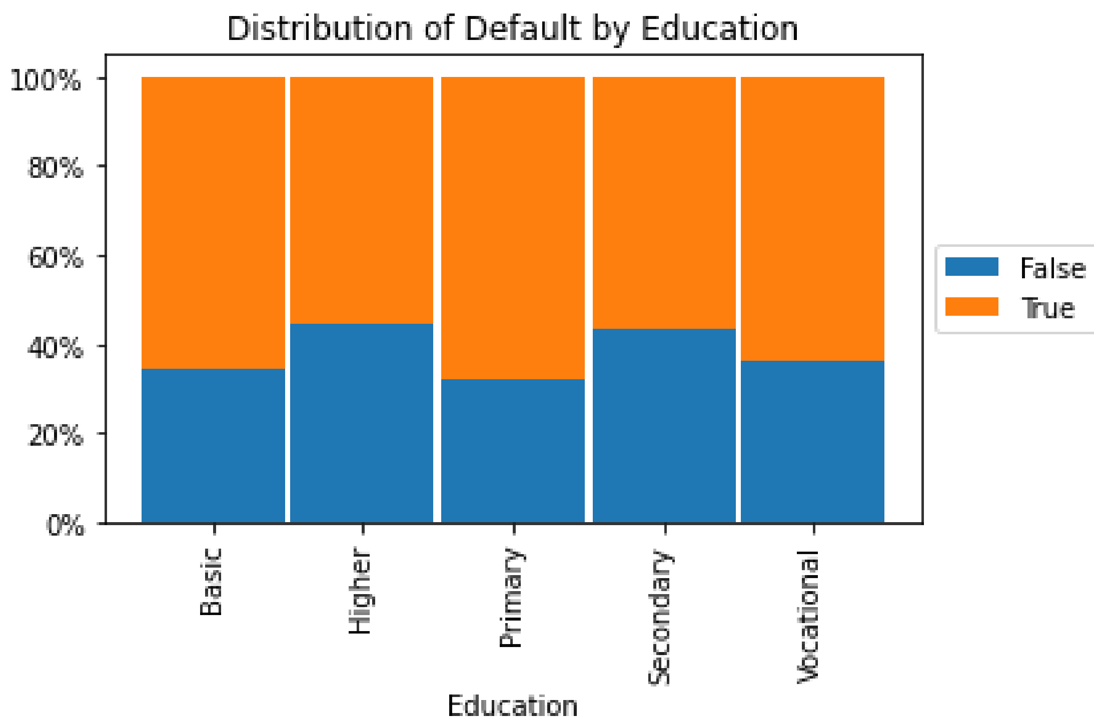
[34]:

Education	Basic	Higher	Primary	Secondary	Vocational
Defaulted					
False	0.347574	0.448079	0.322107	0.436511	0.364045
True	0.652426	0.551921	0.677893	0.563489	0.635955

[35]:

```
# Let's graph it out.
ax = prop_tab_vertical.transpose().plot(kind='bar', stacked=True, width=0.95)
ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
plt.title('Distribution of Default by Education')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

```
<ipython-input-35-daf507a410f8>:3: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
```



Nice, there's pretty low separation between the classes - `Higher` / `Secondary` - `Basic` / `Primary` / `Vocational`

So let's convert this into a binary categorical variable. `EducationSecondaryOrHigher`. Then, we'll drop `Education`.

[36]:
```
df["EducationSecondaryOrHigher"] = [1 if (edu == 'Higher' or edu == 'Secondary')
                                    else 0 for edu in df.Education]
```

[37]:
```
df.loc[:15, ["Education", "EducationSecondaryOrHigher"]]
```

[37]:

	Education	EducationSecondaryOrHigher
0	Secondary	1
1	Basic	0
2	Secondary	1
3	Basic	0
4	Higher	1
5	Secondary	1
6	Basic	0
8	Vocational	0
9	Basic	0
10	Secondary	1
11	Basic	0
12	Vocational	0

```
13   Secondary                          1
14      Higher                          1
15   Secondary                          1
```

[38]:
```
# Remove old column education.
df.drop(columns="Education", inplace=True)
```

Analyzing EmploymentDuration

[39]:
```
# drop nulls b/c bondora doesn't explain what a null employment duration is.
# It could be unemployed, self-employed, this is unknown.
print(df.EmploymentDurationCurrentEmployer.isnull().sum())
df.drop(df.loc[(df.EmploymentDurationCurrentEmployer.isnull() == True)].index,␣
 ↪inplace=True)
```

```
822
```

[40]:
```
df.EmploymentDurationCurrentEmployer.value_counts()
```

[40]:
```
MoreThan5Years    43008
UpTo5Years        23488
UpTo1Year         21068
Retiree            6223
UpTo2Years         5984
UpTo3Years         4954
Other              4218
UpTo4Years         3310
TrialPeriod         673
Name: EmploymentDurationCurrentEmployer, dtype: int64
```

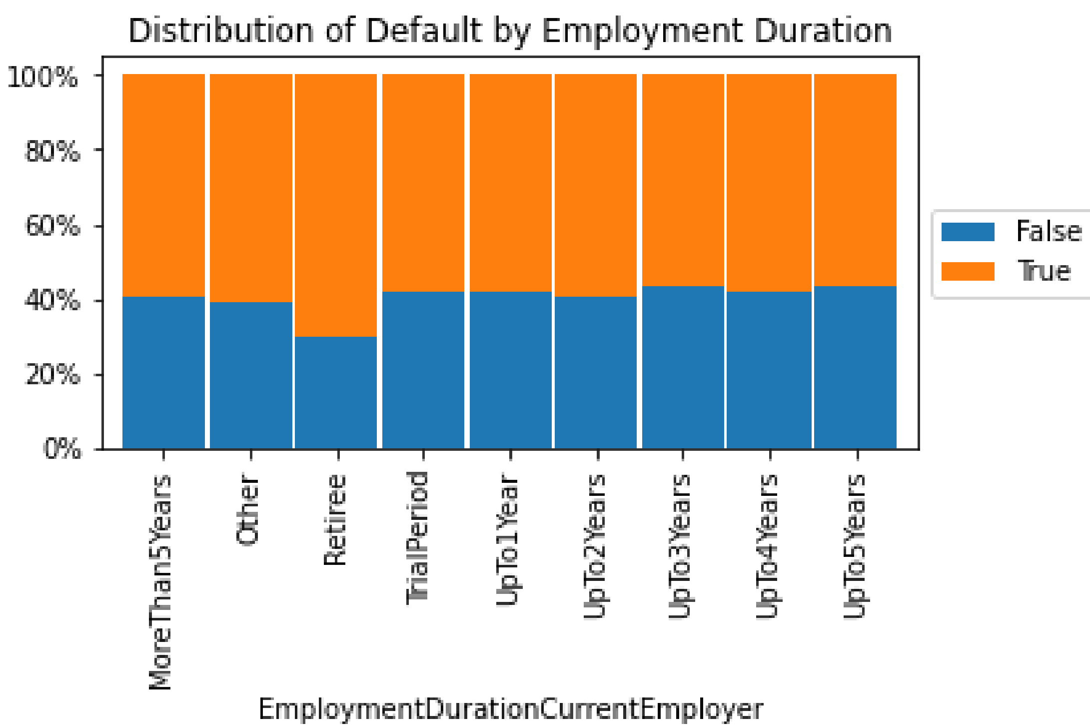
[41]:
```
# Create a cross table and then use a lambda function to compute the proportion␣
 ↪of the categories.
# when axis = 0 (vertical), that's the proportion of each default status over␣
 ↪each category
# when axis = 1 (horizontal), that's the proportion of each category over the␣
 ↪default status
tab = pd.crosstab(df.Defaulted, df.EmploymentDurationCurrentEmployer)
prop_tab_vertical = tab.apply(lambda r: r/r.sum(), axis=0)
prop_tab_vertical
```

[41]:
```
EmploymentDurationCurrentEmployer  MoreThan5Years     Other   Retiree  \
Defaulted
False                                    0.403251  0.386439  0.296641
True                                     0.596749  0.613561  0.703359

EmploymentDurationCurrentEmployer  TrialPeriod  UpTo1Year  UpTo2Years  \
Defaulted
False                                 0.417533   0.419404    0.405247
```

```
True                                      0.582467   0.580596    0.594753

EmploymentDurationCurrentEmployer  UpTo3Years  UpTo4Years  UpTo5Years
Defaulted
False                                    0.430359    0.415408    0.429113
True                                     0.569641    0.584592    0.570887
```

[42]:
```python
# Let's graph it out.
ax = prop_tab_vertical.transpose().plot(kind='bar', stacked=True, width=0.95)
ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
plt.title('Distribution of Default by Employment Duration')
plt.legend(loc='center left', bbox_to_anchor=(1, 0.5))
plt.tight_layout()
plt.show()
```

```
<ipython-input-42-6427d376c9ad>:3: UserWarning: FixedFormatter should only be
used together with FixedLocator
  ax.set_yticklabels(['{:,.0%}'.format(x) for x in ax.get_yticks()])
```



Default is about the same for the entirety of employment duration. However, retirees default much more – by 10%.

[43]:
```python
df["EmploymentStatusRetiree"] = [1 if (emp == 'Retiree')
```

```
                                    else 0 for emp in df.
↪EmploymentDurationCurrentEmployer]
```

[44]:
```
df["EmploymentStatusRetiree"].value_counts(normalize=True)
```

[44]:
```
0    0.944893
1    0.055107
Name: EmploymentStatusRetiree, dtype: float64
```

Retirees make up 5.5% of the dataset, so we should keep them around.

[45]:
```
df.drop(columns="EmploymentDurationCurrentEmployer", inplace=True)
```

Converting predictors to the right types

[46]:
```
typedict = {
"IncomeTotal" : "int64",
"NewCreditCustomer" : "category", # binary
# "VerificationType" : "string",  # Removed this.
"Age" : "int64",
#"Gender" : "string",
"Country" : "string",
"AppliedAmount" : "float64",
"Interest" : "float64",
"LoanDuration" : "int64",
"EducationSecondaryOrHigher" : "category",
#"Education" : "string",#
# "EmploymentDurationCurrentEmployer" : "category",
"EmploymentStatusRetiree" : "category",
"PreviousEarlyRepaymentsCountBeforeLoan" : "int64",
"Defaulted" : "category",         # binary
"DebtToIncomeRatio" : "float64"
}

df = df.astype(typedict)
```

[47]:
```
#Convert the categories that are numbers into their equivalent values.
# Commented this out because we removed verificationType.
# verificationDict = {
# "col" : "VerificationType",
# "1.0" : "NotVerified",
# "2.0" : "VerifiedByPhone",
# "3.0" : "VerifiedByOtherDocument",
# "4.0" : "VerifiedByBankStatement"
# }

# genderDict = {
# "col" : "Gender",
```

```
# "0.0" : "Male",
# "1.0" : "Female",
# "2.0" : "Unknown"
# }

countryDict = {
"col": "Country",
"EE" : "Estonia",
"FI" : "Finland",
"ES" : "Spain",
"SK" : "Slovakia"
}

# educationDict = {
# "col": "Education",
# "-1.0" : "N/A",
# "0.0"  : "N/A",
# "1.0"  : "Primary",
# "2.0"  : "Basic",
# "3.0"  : "Vocational",
# "4.0"  : "Secondary",
# "5.0"  : "Higher"
# }

#category_dicts = [genderDict, countryDict]#, educationDict]
category_dicts = [countryDict]

for cat in category_dicts:
    df[cat["col"]].replace(cat, inplace=True)
```

[48]:
```
# Now replace those with string with category
for col in df.select_dtypes(include=["object", "string"]):
    df[col] = df[col].astype('category')
df.dtypes
```

[48]:
```
IncomeTotal                                int64
NewCreditCustomer                       category
Age                                        int64
Country                                 category
AppliedAmount                            float64
Interest                                 float64
LoanDuration                               int64
PreviousEarlyRepaymentsCountBeforeLoan     int64
Defaulted                               category
DebtToIncomeRatio                        float64
EducationSecondaryOrHigher              category
EmploymentStatusRetiree                 category
```

```
dtype: object
```

[49]:
```
#There was an extra category in VerificationType.
# df.VerificationType.cat.remove_categories("0.0", inplace=True)
#There was an extra category in Education.
#df.Education.cat.remove_categories("N/A", inplace=True)
```

Chosen Predictors for Dataset

[50]:
```
print(f"Out of the original {dataset.shape[1]} variables,{df.shape[1]} made the␣
↪final cut.\n These are:")

df_chosen_preds = df
print(df_chosen_preds.dtypes)
print(df_chosen_preds.shape)
```

```
Out of the original 112 variables,12 made the final cut.
 These are:
IncomeTotal                                 int64
NewCreditCustomer                        category
Age                                         int64
Country                                  category
AppliedAmount                             float64
Interest                                  float64
LoanDuration                                int64
PreviousEarlyRepaymentsCountBeforeLoan      int64
Defaulted                                category
DebtToIncomeRatio                         float64
EducationSecondaryOrHigher               category
EmploymentStatusRetiree                  category
dtype: object
(112926, 12)
```

1.2.2 Addressing Nulls

Education Attribute There are some education attributes with 'null' or "N/A". We pursue omission because Bondora does not give an definition for those values.

[51]:
```
'''
# Remove all records with N/A or null for education.
df.drop(df[(df.Education == "N/A") | (df.Education.isnull())].index,␣
↪inplace=True)
'''
```

[51]:
```
'\n# Remove all records with N/A or null for
education.\ndf.drop(df[(df.Education == "N/A") | (df.Education.isnull())].index,
inplace=True)\n'
```

[52]:
```
# # Now which others are null?
# df_copy = df.copy()
# df.isnull().sum()
```

Employment Duration Attribute Since Bondora gives no information about what n/a represents in employment duration, we have no idea whether it represents `unemployed` or `self-employed`. Let us drop the category since it is less than 1% of the dataset.

[53]:
```
# # A lot in employment duration.
# # it means they have no employment.
# # Convert to string since we have some exceptions coming up if we try to␣
↪convert NaN to "Unemployed"
# df.EmploymentDurationCurrentEmployer = df.EmploymentDurationCurrentEmployer.
↪astype("string")
# df.loc[df[(df.EmploymentDurationCurrentEmployer.isnull())].index,␣
↪"EmploymentDurationCurrentEmployer"] = "Unemployed"
# df.EmploymentDurationCurrentEmployer = df.EmploymentDurationCurrentEmployer.
↪astype("category")
```

[54]:
```
# df.EmploymentDurationCurrentEmployer.value_counts(normalize=True).round(4)*100
```

1.2.3 Addressing Outliers

[55]:
```
# Outliers.
df.skew()
```

[55]:
```
IncomeTotal                                116.014856
Age                                          0.409347
AppliedAmount                                1.526566
Interest                                     3.339557
LoanDuration                                -0.753873
PreviousEarlyRepaymentsCountBeforeLoan       8.801628
DebtToIncomeRatio                          220.550673
dtype: float64
```

Skewed Income Income is way too high. Why, though? Let's take at a summary and then look at the highest values.

[56]:
```
df.IncomeTotal.describe().round()
```

[56]:
```
count    112926.0
mean       1760.0
std        6138.0
min           0.0
25%         900.0
50%        1300.0
75%        1932.0
```

```
max      1012019.0
Name: IncomeTotal, dtype: float64
```

The average borrower on the platform makes around ~1800 Euros a month. But the max is over 1 million Euros a month! How many more are like this? Let's define a function where we can analyze the skew.

[57]:
```python
def analyze_skew(num_of_records, column, ascending_value=False):
    """ Analyze the skew of a column. Generates a report indicating the skew␣
↪change from removing numerous outliers.
        Params:
        num_of_records = The number of records to analyze from the column.
        column = The name of the column from the data frame.
        ascending_value = Whether to look at the highest (True) values or the␣
↪lowest (False) values of the column.
    """
    print(f"Top {num_of_records} values (Subset is {(num_of_records/df[column].
↪shape[0]):.2%} of dataset)")
    records = df[column].sort_values(ascending=ascending_value).
↪head(num_of_records)
    outputs = []
    skew_val = None
    for i in range(num_of_records):
        skew_val_prev = 0 if (skew_val == None) else skew_val
        skew_val = df.drop(df[column].sort_values(ascending=ascending_value).
↪head(i).index).skew()[column].round(2)
        diff = skew_val_prev - skew_val if (skew_val_prev != 0) else 0
        outputs.append(f"Index: {records.index[i]:6} | Value: {records.iloc[i]:.
↪2f} | Drop Count: {i} " +
                       f"| Current Skew: {skew_val:.2f} | Change: {diff:.2f}")

    for record in outputs:
        print(record)

    return
```

[58]:
```python
analyze_skew(250, 'IncomeTotal', False)
```

```
Top 250 values (Subset is 0.22% of dataset)
Index:  83832 | Value: 1012019.00 | Drop Count: 0 | Current Skew: 116.01 |
Change: 0.00
Index:  69483 | Value: 1012019.00 | Drop Count: 1 | Current Skew: 115.49 |
Change: 0.52
Index: 153411 | Value: 900555.00 | Drop Count: 2 | Current Skew: 98.77 | Change:
16.72
Index: 160332 | Value: 500600.00 | Drop Count: 3 | Current Skew: 48.80 | Change:
49.97
Index:  86356 | Value: 280000.00 | Drop Count: 4 | Current Skew: 31.84 | Change:
```

```
16.96
Index:  56677 | Value: 235800.00 | Drop Count: 5 | Current Skew: 28.88 | Change:
2.96
Index: 102379 | Value: 235000.00 | Drop Count: 6 | Current Skew: 27.07 | Change:
1.81
Index:  19280 | Value: 228550.00 | Drop Count: 7 | Current Skew: 24.92 | Change:
2.15
Index:  97582 | Value: 220000.00 | Drop Count: 8 | Current Skew: 22.59 | Change:
2.33
Index:  22447 | Value: 190900.00 | Drop Count: 9 | Current Skew: 20.18 | Change:
2.41
Index:  73255 | Value: 160974.00 | Drop Count: 10 | Current Skew: 18.51 |
Change: 1.67
Index:   2291 | Value: 133000.00 | Drop Count: 11 | Current Skew: 17.53 |
Change: 0.98
Index:  43857 | Value: 122026.00 | Drop Count: 12 | Current Skew: 17.02 |
Change: 0.51
Index:  70162 | Value: 120045.00 | Drop Count: 13 | Current Skew: 16.65 |
Change: 0.37
Index:  87909 | Value: 120045.00 | Drop Count: 14 | Current Skew: 16.28 |
Change: 0.37
Index:  83777 | Value: 120045.00 | Drop Count: 15 | Current Skew: 15.88 |
Change: 0.40
Index:  39583 | Value: 117900.00 | Drop Count: 16 | Current Skew: 15.46 |
Change: 0.42
Index:  39116 | Value: 108476.00 | Drop Count: 17 | Current Skew: 15.03 |
Change: 0.43
Index:  36730 | Value: 108457.00 | Drop Count: 18 | Current Skew: 14.72 |
Change: 0.31
Index:  55341 | Value: 101571.00 | Drop Count: 19 | Current Skew: 14.38 |
Change: 0.34
Index: 160775 | Value: 100000.00 | Drop Count: 20 | Current Skew: 14.11 |
Change: 0.27
Index:  76960 | Value: 100000.00 | Drop Count: 21 | Current Skew: 13.85 |
Change: 0.26
Index:  87040 | Value: 90000.00 | Drop Count: 22 | Current Skew: 13.57 | Change:
0.28
Index:  83549 | Value: 90000.00 | Drop Count: 23 | Current Skew: 13.38 | Change:
0.19
Index:   1923 | Value: 90000.00 | Drop Count: 24 | Current Skew: 13.18 | Change:
0.20
Index: 134724 | Value: 90000.00 | Drop Count: 25 | Current Skew: 12.98 | Change:
0.20
Index:   2199 | Value: 87100.00 | Drop Count: 26 | Current Skew: 12.77 | Change:
0.21
Index:  54832 | Value: 85000.00 | Drop Count: 27 | Current Skew: 12.57 | Change:
0.20
Index:  84868 | Value: 85000.00 | Drop Count: 28 | Current Skew: 12.39 | Change:
```

```
0.18
Index:  62145 | Value: 84000.00 | Drop Count: 29 | Current Skew: 12.20 | Change:
0.19
Index: 139691 | Value: 82000.00 | Drop Count: 30 | Current Skew: 12.01 | Change:
0.19
Index: 128376 | Value: 80000.00 | Drop Count: 31 | Current Skew: 11.84 | Change:
0.17
Index: 112153 | Value: 80000.00 | Drop Count: 32 | Current Skew: 11.67 | Change:
0.17
Index:  77426 | Value: 80000.00 | Drop Count: 33 | Current Skew: 11.50 | Change:
0.17
Index:  70598 | Value: 75000.00 | Drop Count: 34 | Current Skew: 11.32 | Change:
0.18
Index: 116453 | Value: 69000.00 | Drop Count: 35 | Current Skew: 11.18 | Change:
0.14
Index:  30136 | Value: 68000.00 | Drop Count: 36 | Current Skew: 11.08 | Change:
0.10
Index:  69494 | Value: 65600.00 | Drop Count: 37 | Current Skew: 10.98 | Change:
0.10
Index:   1028 | Value: 65000.00 | Drop Count: 38 | Current Skew: 10.90 | Change:
0.08
Index:  72213 | Value: 64089.00 | Drop Count: 39 | Current Skew: 10.82 | Change:
0.08
Index:   1552 | Value: 62500.00 | Drop Count: 40 | Current Skew: 10.73 | Change:
0.09
Index:  83348 | Value: 60000.00 | Drop Count: 41 | Current Skew: 10.66 | Change:
0.07
Index: 103214 | Value: 60000.00 | Drop Count: 42 | Current Skew: 10.60 | Change:
0.06
Index: 109133 | Value: 58800.00 | Drop Count: 43 | Current Skew: 10.53 | Change:
0.07
Index:  96373 | Value: 56400.00 | Drop Count: 44 | Current Skew: 10.47 | Change:
0.06
Index:   1517 | Value: 55000.00 | Drop Count: 45 | Current Skew: 10.42 | Change:
0.05
Index:  82246 | Value: 55000.00 | Drop Count: 46 | Current Skew: 10.38 | Change:
0.04
Index:   1101 | Value: 55000.00 | Drop Count: 47 | Current Skew: 10.33 | Change:
0.05
Index:   1634 | Value: 55000.00 | Drop Count: 48 | Current Skew: 10.29 | Change:
0.04
Index: 126730 | Value: 53100.00 | Drop Count: 49 | Current Skew: 10.24 | Change:
0.05
Index:  52846 | Value: 53000.00 | Drop Count: 50 | Current Skew: 10.20 | Change:
0.04
Index:  49704 | Value: 53000.00 | Drop Count: 51 | Current Skew: 10.16 | Change:
0.04
Index:  75611 | Value: 51600.00 | Drop Count: 52 | Current Skew: 10.11 | Change:
```

```
0.05
Index:  63350 | Value: 51600.00 | Drop Count: 53 | Current Skew: 10.08 | Change:
0.03
Index: 113243 | Value: 51600.00 | Drop Count: 54 | Current Skew: 10.04 | Change:
0.04
Index:  65121 | Value: 50000.00 | Drop Count: 55 | Current Skew: 10.00 | Change:
0.04
Index:  35330 | Value: 50000.00 | Drop Count: 56 | Current Skew: 9.96 | Change:
0.04
Index:  78493 | Value: 50000.00 | Drop Count: 57 | Current Skew: 9.93 | Change:
0.03
Index:  73008 | Value: 50000.00 | Drop Count: 58 | Current Skew: 9.89 | Change:
0.04
Index: 112319 | Value: 50000.00 | Drop Count: 59 | Current Skew: 9.86 | Change:
0.03
Index:  80924 | Value: 50000.00 | Drop Count: 60 | Current Skew: 9.82 | Change:
0.04
Index: 100565 | Value: 50000.00 | Drop Count: 61 | Current Skew: 9.78 | Change:
0.04
Index: 135964 | Value: 48595.00 | Drop Count: 62 | Current Skew: 9.75 | Change:
0.03
Index: 111025 | Value: 48000.00 | Drop Count: 63 | Current Skew: 9.71 | Change:
0.04
Index: 119246 | Value: 48000.00 | Drop Count: 64 | Current Skew: 9.68 | Change:
0.03
Index:  78374 | Value: 48000.00 | Drop Count: 65 | Current Skew: 9.65 | Change:
0.03
Index:   1503 | Value: 48000.00 | Drop Count: 66 | Current Skew: 9.62 | Change:
0.03
Index:  86081 | Value: 48000.00 | Drop Count: 67 | Current Skew: 9.58 | Change:
0.04
Index:  76745 | Value: 48000.00 | Drop Count: 68 | Current Skew: 9.55 | Change:
0.03
Index:  49557 | Value: 47053.00 | Drop Count: 69 | Current Skew: 9.51 | Change:
0.04
Index:  54688 | Value: 47053.00 | Drop Count: 70 | Current Skew: 9.48 | Change:
0.03
Index:  58472 | Value: 47053.00 | Drop Count: 71 | Current Skew: 9.45 | Change:
0.03
Index:   2948 | Value: 45200.00 | Drop Count: 72 | Current Skew: 9.42 | Change:
0.03
Index:  89319 | Value: 45000.00 | Drop Count: 73 | Current Skew: 9.39 | Change:
0.03
Index:  95576 | Value: 45000.00 | Drop Count: 74 | Current Skew: 9.36 | Change:
0.03
Index:  69155 | Value: 45000.00 | Drop Count: 75 | Current Skew: 9.33 | Change:
0.03
Index: 130744 | Value: 44000.00 | Drop Count: 76 | Current Skew: 9.31 | Change:
```

```
0.02
Index:  80104 | Value: 44000.00 | Drop Count: 77 | Current Skew: 9.28 | Change:
0.03
Index:  48652 | Value: 43000.00 | Drop Count: 78 | Current Skew: 9.26 | Change:
0.02
Index:  45009 | Value: 43000.00 | Drop Count: 79 | Current Skew: 9.23 | Change:
0.03
Index:  49596 | Value: 43000.00 | Drop Count: 80 | Current Skew: 9.21 | Change:
0.02
Index:  80519 | Value: 43000.00 | Drop Count: 81 | Current Skew: 9.19 | Change:
0.02
Index: 104909 | Value: 42500.00 | Drop Count: 82 | Current Skew: 9.16 | Change:
0.03
Index: 114827 | Value: 42000.00 | Drop Count: 83 | Current Skew: 9.14 | Change:
0.02
Index:  54104 | Value: 42000.00 | Drop Count: 84 | Current Skew: 9.12 | Change:
0.02
Index:  53659 | Value: 42000.00 | Drop Count: 85 | Current Skew: 9.09 | Change:
0.03
Index:  94070 | Value: 41375.00 | Drop Count: 86 | Current Skew: 9.07 | Change:
0.02
Index:  72265 | Value: 41375.00 | Drop Count: 87 | Current Skew: 9.05 | Change:
0.02
Index:  73293 | Value: 41375.00 | Drop Count: 88 | Current Skew: 9.03 | Change:
0.02
Index: 101703 | Value: 40000.00 | Drop Count: 89 | Current Skew: 9.01 | Change:
0.02
Index:   3139 | Value: 40000.00 | Drop Count: 90 | Current Skew: 8.99 | Change:
0.02
Index:   2365 | Value: 40000.00 | Drop Count: 91 | Current Skew: 8.97 | Change:
0.02
Index:  98374 | Value: 40000.00 | Drop Count: 92 | Current Skew: 8.95 | Change:
0.02
Index: 112298 | Value: 40000.00 | Drop Count: 93 | Current Skew: 8.93 | Change:
0.02
Index: 135268 | Value: 40000.00 | Drop Count: 94 | Current Skew: 8.91 | Change:
0.02
Index:  65670 | Value: 40000.00 | Drop Count: 95 | Current Skew: 8.90 | Change:
0.01
Index:   3202 | Value: 40000.00 | Drop Count: 96 | Current Skew: 8.88 | Change:
0.02
Index: 126240 | Value: 40000.00 | Drop Count: 97 | Current Skew: 8.86 | Change:
0.02
Index:  90244 | Value: 40000.00 | Drop Count: 98 | Current Skew: 8.84 | Change:
0.02
Index: 137486 | Value: 40000.00 | Drop Count: 99 | Current Skew: 8.81 | Change:
0.03
Index:   2425 | Value: 40000.00 | Drop Count: 100 | Current Skew: 8.79 | Change:
```

```
0.02
Index:  88845 | Value: 40000.00 | Drop Count: 101 | Current Skew: 8.77 | Change:
0.02
Index: 111020 | Value: 40000.00 | Drop Count: 102 | Current Skew: 8.75 | Change:
0.02
Index:  75138 | Value: 39000.00 | Drop Count: 103 | Current Skew: 8.73 | Change:
0.02
Index:  62213 | Value: 39000.00 | Drop Count: 104 | Current Skew: 8.71 | Change:
0.02
Index:  59235 | Value: 39000.00 | Drop Count: 105 | Current Skew: 8.69 | Change:
0.02
Index:  68618 | Value: 39000.00 | Drop Count: 106 | Current Skew: 8.67 | Change:
0.02
Index:  88284 | Value: 39000.00 | Drop Count: 107 | Current Skew: 8.65 | Change:
0.02
Index: 139819 | Value: 38000.00 | Drop Count: 108 | Current Skew: 8.63 | Change:
0.02
Index: 112488 | Value: 38000.00 | Drop Count: 109 | Current Skew: 8.61 | Change:
0.02
Index: 130534 | Value: 38000.00 | Drop Count: 110 | Current Skew: 8.59 | Change:
0.02
Index:  76108 | Value: 38000.00 | Drop Count: 111 | Current Skew: 8.58 | Change:
0.01
Index:  82678 | Value: 38000.00 | Drop Count: 112 | Current Skew: 8.56 | Change:
0.02
Index:  69353 | Value: 38000.00 | Drop Count: 113 | Current Skew: 8.54 | Change:
0.02
Index: 100914 | Value: 36500.00 | Drop Count: 114 | Current Skew: 8.52 | Change:
0.02
Index:  93865 | Value: 36000.00 | Drop Count: 115 | Current Skew: 8.50 | Change:
0.02
Index:  71290 | Value: 36000.00 | Drop Count: 116 | Current Skew: 8.49 | Change:
0.01
Index:  88187 | Value: 36000.00 | Drop Count: 117 | Current Skew: 8.48 | Change:
0.01
Index:   1858 | Value: 36000.00 | Drop Count: 118 | Current Skew: 8.46 | Change:
0.02
Index:  86004 | Value: 36000.00 | Drop Count: 119 | Current Skew: 8.45 | Change:
0.01
Index:  68895 | Value: 36000.00 | Drop Count: 120 | Current Skew: 8.43 | Change:
0.02
Index: 133710 | Value: 36000.00 | Drop Count: 121 | Current Skew: 8.42 | Change:
0.01
Index:  50416 | Value: 35820.00 | Drop Count: 122 | Current Skew: 8.40 | Change:
0.02
Index:  39198 | Value: 35820.00 | Drop Count: 123 | Current Skew: 8.38 | Change:
0.02
Index:  58828 | Value: 35820.00 | Drop Count: 124 | Current Skew: 8.37 | Change:
```

```
0.01
Index: 116700 | Value: 35390.00 | Drop Count: 125 | Current Skew: 8.35 | Change:
0.02
Index: 136131 | Value: 35000.00 | Drop Count: 126 | Current Skew: 8.34 | Change:
0.01
Index:  34146 | Value: 35000.00 | Drop Count: 127 | Current Skew: 8.32 | Change:
0.02
Index:  72108 | Value: 35000.00 | Drop Count: 128 | Current Skew: 8.31 | Change:
0.01
Index: 137836 | Value: 35000.00 | Drop Count: 129 | Current Skew: 8.30 | Change:
0.01
Index:   1194 | Value: 35000.00 | Drop Count: 130 | Current Skew: 8.28 | Change:
0.02
Index:   1352 | Value: 35000.00 | Drop Count: 131 | Current Skew: 8.27 | Change:
0.01
Index:  72043 | Value: 35000.00 | Drop Count: 132 | Current Skew: 8.25 | Change:
0.02
Index:  63359 | Value: 35000.00 | Drop Count: 133 | Current Skew: 8.24 | Change:
0.01
Index: 120680 | Value: 35000.00 | Drop Count: 134 | Current Skew: 8.22 | Change:
0.02
Index:   1195 | Value: 35000.00 | Drop Count: 135 | Current Skew: 8.21 | Change:
0.01
Index:  37329 | Value: 35000.00 | Drop Count: 136 | Current Skew: 8.19 | Change:
0.02
Index: 125073 | Value: 35000.00 | Drop Count: 137 | Current Skew: 8.18 | Change:
0.01
Index:  84717 | Value: 35000.00 | Drop Count: 138 | Current Skew: 8.16 | Change:
0.02
Index: 127870 | Value: 35000.00 | Drop Count: 139 | Current Skew: 8.14 | Change:
0.02
Index: 138909 | Value: 35000.00 | Drop Count: 140 | Current Skew: 8.13 | Change:
0.01
Index:  68380 | Value: 35000.00 | Drop Count: 141 | Current Skew: 8.11 | Change:
0.02
Index: 116016 | Value: 35000.00 | Drop Count: 142 | Current Skew: 8.09 | Change:
0.02
Index:  85376 | Value: 35000.00 | Drop Count: 143 | Current Skew: 8.08 | Change:
0.01
Index:  83159 | Value: 34900.00 | Drop Count: 144 | Current Skew: 8.06 | Change:
0.02
Index:  82593 | Value: 34900.00 | Drop Count: 145 | Current Skew: 8.04 | Change:
0.02
Index:   1720 | Value: 34100.00 | Drop Count: 146 | Current Skew: 8.02 | Change:
0.02
Index: 127122 | Value: 34000.00 | Drop Count: 147 | Current Skew: 8.01 | Change:
0.01
Index:  66941 | Value: 34000.00 | Drop Count: 148 | Current Skew: 7.99 | Change:
```

0.02
Index: 52493 | Value: 33700.00 | Drop Count: 149 | Current Skew: 7.98 | Change: 0.01
Index: 61082 | Value: 33700.00 | Drop Count: 150 | Current Skew: 7.96 | Change: 0.02
Index: 74058 | Value: 33600.00 | Drop Count: 151 | Current Skew: 7.95 | Change: 0.01
Index: 83328 | Value: 33000.00 | Drop Count: 152 | Current Skew: 7.93 | Change: 0.02
Index: 57535 | Value: 33000.00 | Drop Count: 153 | Current Skew: 7.92 | Change: 0.01
Index: 78329 | Value: 33000.00 | Drop Count: 154 | Current Skew: 7.91 | Change: 0.01
Index: 55842 | Value: 33000.00 | Drop Count: 155 | Current Skew: 7.89 | Change: 0.02
Index: 142341 | Value: 32500.00 | Drop Count: 156 | Current Skew: 7.88 | Change: 0.01
Index: 2046 | Value: 32400.00 | Drop Count: 157 | Current Skew: 7.86 | Change: 0.02
Index: 1076 | Value: 32000.00 | Drop Count: 158 | Current Skew: 7.85 | Change: 0.01
Index: 68070 | Value: 32000.00 | Drop Count: 159 | Current Skew: 7.84 | Change: 0.01
Index: 48049 | Value: 32000.00 | Drop Count: 160 | Current Skew: 7.83 | Change: 0.01
Index: 51767 | Value: 32000.00 | Drop Count: 161 | Current Skew: 7.81 | Change: 0.02
Index: 82279 | Value: 32000.00 | Drop Count: 162 | Current Skew: 7.80 | Change: 0.01
Index: 89546 | Value: 32000.00 | Drop Count: 163 | Current Skew: 7.79 | Change: 0.01
Index: 37357 | Value: 32000.00 | Drop Count: 164 | Current Skew: 7.77 | Change: 0.02
Index: 88652 | Value: 32000.00 | Drop Count: 165 | Current Skew: 7.76 | Change: 0.01
Index: 1313 | Value: 32000.00 | Drop Count: 166 | Current Skew: 7.75 | Change: 0.01
Index: 97136 | Value: 32000.00 | Drop Count: 167 | Current Skew: 7.73 | Change: 0.02
Index: 49153 | Value: 32000.00 | Drop Count: 168 | Current Skew: 7.72 | Change: 0.01
Index: 93306 | Value: 31595.00 | Drop Count: 169 | Current Skew: 7.71 | Change: 0.01
Index: 119101 | Value: 31000.00 | Drop Count: 170 | Current Skew: 7.69 | Change: 0.02
Index: 2450 | Value: 31000.00 | Drop Count: 171 | Current Skew: 7.68 | Change: 0.01
Index: 2449 | Value: 31000.00 | Drop Count: 172 | Current Skew: 7.67 | Change:

```
0.01
Index: 100995 | Value: 31000.00 | Drop Count: 173 | Current Skew: 7.66 | Change:
0.01
Index:   3215 | Value: 31000.00 | Drop Count: 174 | Current Skew: 7.65 | Change:
0.01
Index:  75080 | Value: 31000.00 | Drop Count: 175 | Current Skew: 7.63 | Change:
0.02
Index:  79529 | Value: 31000.00 | Drop Count: 176 | Current Skew: 7.62 | Change:
0.01
Index:   2371 | Value: 31000.00 | Drop Count: 177 | Current Skew: 7.61 | Change:
0.01
Index: 128514 | Value: 31000.00 | Drop Count: 178 | Current Skew: 7.60 | Change:
0.01
Index: 107011 | Value: 31000.00 | Drop Count: 179 | Current Skew: 7.58 | Change:
0.02
Index:   3103 | Value: 31000.00 | Drop Count: 180 | Current Skew: 7.57 | Change:
0.01
Index:  79780 | Value: 30970.00 | Drop Count: 181 | Current Skew: 7.56 | Change:
0.01
Index:  77767 | Value: 30000.00 | Drop Count: 182 | Current Skew: 7.55 | Change:
0.01
Index: 104610 | Value: 30000.00 | Drop Count: 183 | Current Skew: 7.53 | Change:
0.02
Index:  47739 | Value: 30000.00 | Drop Count: 184 | Current Skew: 7.52 | Change:
0.01
Index:  81232 | Value: 30000.00 | Drop Count: 185 | Current Skew: 7.51 | Change:
0.01
Index:   1694 | Value: 30000.00 | Drop Count: 186 | Current Skew: 7.50 | Change:
0.01
Index:  94121 | Value: 30000.00 | Drop Count: 187 | Current Skew: 7.49 | Change:
0.01
Index:  60759 | Value: 30000.00 | Drop Count: 188 | Current Skew: 7.48 | Change:
0.01
Index: 108404 | Value: 30000.00 | Drop Count: 189 | Current Skew: 7.47 | Change:
0.01
Index:  81421 | Value: 30000.00 | Drop Count: 190 | Current Skew: 7.45 | Change:
0.02
Index:   1518 | Value: 30000.00 | Drop Count: 191 | Current Skew: 7.44 | Change:
0.01
Index:  41468 | Value: 30000.00 | Drop Count: 192 | Current Skew: 7.43 | Change:
0.01
Index:   3058 | Value: 30000.00 | Drop Count: 193 | Current Skew: 7.42 | Change:
0.01
Index:  99026 | Value: 30000.00 | Drop Count: 194 | Current Skew: 7.41 | Change:
0.01
Index:  95575 | Value: 30000.00 | Drop Count: 195 | Current Skew: 7.39 | Change:
0.02
Index:  82165 | Value: 30000.00 | Drop Count: 196 | Current Skew: 7.38 | Change:
```

0.01
Index: 69412 | Value: 30000.00 | Drop Count: 197 | Current Skew: 7.37 | Change: 0.01
Index: 91107 | Value: 30000.00 | Drop Count: 198 | Current Skew: 7.35 | Change: 0.02
Index: 75059 | Value: 30000.00 | Drop Count: 199 | Current Skew: 7.34 | Change: 0.01
Index: 92911 | Value: 30000.00 | Drop Count: 200 | Current Skew: 7.33 | Change: 0.01
Index: 66419 | Value: 30000.00 | Drop Count: 201 | Current Skew: 7.32 | Change: 0.01
Index: 2968 | Value: 30000.00 | Drop Count: 202 | Current Skew: 7.30 | Change: 0.02
Index: 2652 | Value: 29300.00 | Drop Count: 203 | Current Skew: 7.29 | Change: 0.01
Index: 1509 | Value: 29000.00 | Drop Count: 204 | Current Skew: 7.28 | Change: 0.01
Index: 3178 | Value: 29000.00 | Drop Count: 205 | Current Skew: 7.27 | Change: 0.01
Index: 2344 | Value: 29000.00 | Drop Count: 206 | Current Skew: 7.25 | Change: 0.02
Index: 3096 | Value: 29000.00 | Drop Count: 207 | Current Skew: 7.24 | Change: 0.01
Index: 2296 | Value: 29000.00 | Drop Count: 208 | Current Skew: 7.23 | Change: 0.01
Index: 2345 | Value: 29000.00 | Drop Count: 209 | Current Skew: 7.22 | Change: 0.01
Index: 67530 | Value: 28900.00 | Drop Count: 210 | Current Skew: 7.21 | Change: 0.01
Index: 71132 | Value: 28900.00 | Drop Count: 211 | Current Skew: 7.19 | Change: 0.02
Index: 76559 | Value: 28900.00 | Drop Count: 212 | Current Skew: 7.18 | Change: 0.01
Index: 73106 | Value: 28800.00 | Drop Count: 213 | Current Skew: 7.17 | Change: 0.01
Index: 44578 | Value: 28511.00 | Drop Count: 214 | Current Skew: 7.16 | Change: 0.01
Index: 55320 | Value: 28511.00 | Drop Count: 215 | Current Skew: 7.15 | Change: 0.01
Index: 56969 | Value: 28511.00 | Drop Count: 216 | Current Skew: 7.14 | Change: 0.01
Index: 70734 | Value: 28511.00 | Drop Count: 217 | Current Skew: 7.12 | Change: 0.02
Index: 59916 | Value: 28511.00 | Drop Count: 218 | Current Skew: 7.11 | Change: 0.01
Index: 52946 | Value: 28511.00 | Drop Count: 219 | Current Skew: 7.10 | Change: 0.01
Index: 62134 | Value: 28500.00 | Drop Count: 220 | Current Skew: 7.09 | Change:

```
0.01
Index:  76692 | Value: 28200.00 | Drop Count: 221 | Current Skew: 7.08 | Change:
0.01
Index:  68975 | Value: 28000.00 | Drop Count: 222 | Current Skew: 7.07 | Change:
0.01
Index:  35575 | Value: 28000.00 | Drop Count: 223 | Current Skew: 7.05 | Change:
0.02
Index:  85563 | Value: 28000.00 | Drop Count: 224 | Current Skew: 7.04 | Change:
0.01
Index: 120376 | Value: 28000.00 | Drop Count: 225 | Current Skew: 7.03 | Change:
0.01
Index:  69990 | Value: 28000.00 | Drop Count: 226 | Current Skew: 7.02 | Change:
0.01
Index:  97351 | Value: 28000.00 | Drop Count: 227 | Current Skew: 7.01 | Change:
0.01
Index:  82446 | Value: 28000.00 | Drop Count: 228 | Current Skew: 7.00 | Change:
0.01
Index:  86067 | Value: 28000.00 | Drop Count: 229 | Current Skew: 6.98 | Change:
0.02
Index: 103908 | Value: 28000.00 | Drop Count: 230 | Current Skew: 6.97 | Change:
0.01
Index:  77037 | Value: 28000.00 | Drop Count: 231 | Current Skew: 6.96 | Change:
0.01
Index:   2186 | Value: 27600.00 | Drop Count: 232 | Current Skew: 6.95 | Change:
0.01
Index:   1637 | Value: 27500.00 | Drop Count: 233 | Current Skew: 6.94 | Change:
0.01
Index:  68680 | Value: 27500.00 | Drop Count: 234 | Current Skew: 6.93 | Change:
0.01
Index:  74345 | Value: 27500.00 | Drop Count: 235 | Current Skew: 6.92 | Change:
0.01
Index: 126448 | Value: 27008.00 | Drop Count: 236 | Current Skew: 6.90 | Change:
0.02
Index:  73706 | Value: 27000.00 | Drop Count: 237 | Current Skew: 6.89 | Change:
0.01
Index:   1756 | Value: 27000.00 | Drop Count: 238 | Current Skew: 6.88 | Change:
0.01
Index: 140525 | Value: 27000.00 | Drop Count: 239 | Current Skew: 6.87 | Change:
0.01
Index:  94707 | Value: 27000.00 | Drop Count: 240 | Current Skew: 6.86 | Change:
0.01
Index:  70683 | Value: 27000.00 | Drop Count: 241 | Current Skew: 6.85 | Change:
0.01
Index:   3177 | Value: 26600.00 | Drop Count: 242 | Current Skew: 6.84 | Change:
0.01
Index:  94902 | Value: 26000.00 | Drop Count: 243 | Current Skew: 6.83 | Change:
0.01
Index:   2481 | Value: 26000.00 | Drop Count: 244 | Current Skew: 6.82 | Change:
```

```
0.01
Index:  69984 | Value: 26000.00 | Drop Count: 245 | Current Skew: 6.81 | Change:
0.01
Index:  87942 | Value: 26000.00 | Drop Count: 246 | Current Skew: 6.80 | Change:
0.01
Index:  62335 | Value: 26000.00 | Drop Count: 247 | Current Skew: 6.79 | Change:
0.01
Index:   1268 | Value: 26000.00 | Drop Count: 248 | Current Skew: 6.78 | Change:
0.01
Index:  65112 | Value: 26000.00 | Drop Count: 249 | Current Skew: 6.78 | Change:
0.00
```

[59]:
```
df.drop(df.IncomeTotal.sort_values(ascending=False).head(50).index,␣
 ↪inplace=True)
df.skew()
```

[59]:
```
IncomeTotal                                10.197099
Age                                         0.409368
AppliedAmount                               1.526706
Interest                                    3.339798
LoanDuration                               -0.753498
PreviousEarlyRepaymentsCountBeforeLoan      8.801725
DebtToIncomeRatio                         220.501881
dtype: float64
```

Skewed DebtToIncome There are many extreme debt-to-income ratios (likely because of how we dealt with them in section 2.1.6). Keep in mind that ratio of 1 means that you're incurring as much debt as you make every month. They're so high because some people put down that they don't earn anything in terms of total monthly income, while still having debt.

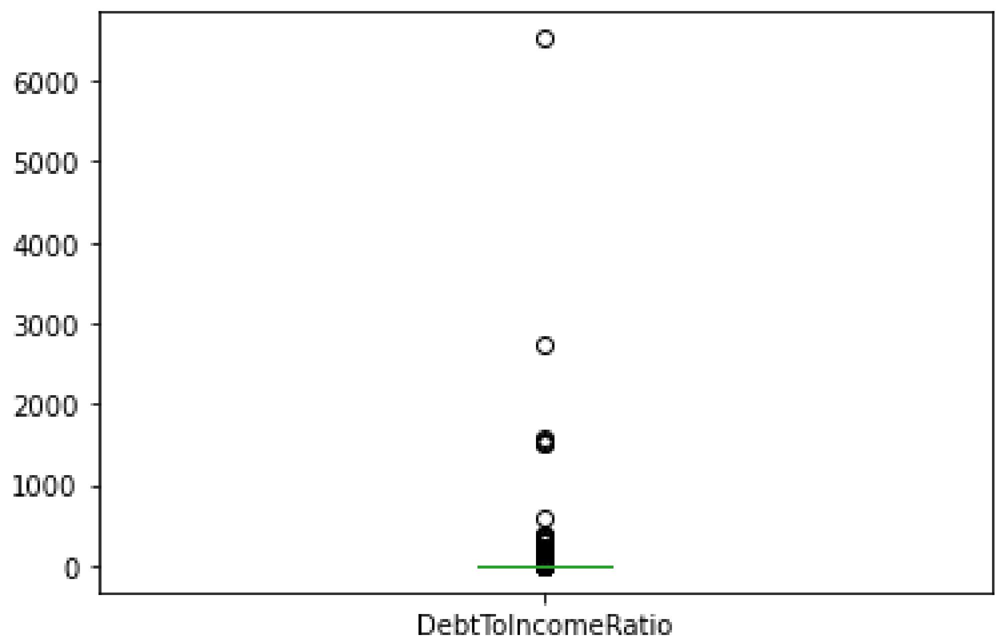
We tried to deal with this earlier in Section 2.1.6, but clearly they've created outliers.

We can also drop this too because we have an abundance of records for repaid and default loans.

[60]:
```
print(df.DebtToIncomeRatio.describe())
df.DebtToIncomeRatio.plot.box()
```

```
count    112876.000000
mean          0.521563
std          23.275133
min           0.000000
25%           0.053457
50%           0.276796
75%           0.508579
max        6526.315789
Name: DebtToIncomeRatio, dtype: float64
```

[60]:
```
<AxesSubplot:>
```



[61]:
```
analyze_skew(50, "DebtToIncomeRatio", ascending_value=False)
```

```
Top 50 values (Subset is 0.04% of dataset)
Index: 121310 | Value: 6526.32 | Drop Count: 0 | Current Skew: 220.50 | Change:
0.00
Index:   3084 | Value: 2750.00 | Drop Count: 1 | Current Skew: 150.89 | Change:
69.61
Index: 141766 | Value: 1578.00 | Drop Count: 2 | Current Skew: 139.29 | Change:
11.60
Index: 129008 | Value: 1554.00 | Drop Count: 3 | Current Skew: 151.54 | Change:
-12.25
Index:  98169 | Value: 1521.00 | Drop Count: 4 | Current Skew: 166.25 | Change:
-14.71
Index:  93982 | Value: 1500.00 | Drop Count: 5 | Current Skew: 179.53 | Change:
-13.28
Index: 135586 | Value: 600.00 | Drop Count: 6 | Current Skew: 99.65 | Change:
79.88
Index:   2183 | Value: 400.00 | Drop Count: 7 | Current Skew: 91.97 | Change:
7.68
Index: 125188 | Value: 375.00 | Drop Count: 8 | Current Skew: 93.57 | Change:
-1.60
Index:  95879 | Value: 370.00 | Drop Count: 9 | Current Skew: 95.42 | Change:
-1.85
Index:  33384 | Value: 344.00 | Drop Count: 10 | Current Skew: 96.41 | Change:
-0.99
Index:  95413 | Value: 341.50 | Drop Count: 11 | Current Skew: 97.38 | Change:
```

```
-0.97
Index: 124584 | Value: 292.00 | Drop Count: 12 | Current Skew: 95.84 | Change:
1.54
Index: 132453 | Value: 278.00 | Drop Count: 13 | Current Skew: 95.85 | Change:
-0.01
Index: 100542 | Value: 231.67 | Drop Count: 14 | Current Skew: 94.26 | Change:
1.59
Index: 132249 | Value: 225.00 | Drop Count: 15 | Current Skew: 94.97 | Change:
-0.71
Index: 137900 | Value: 200.00 | Drop Count: 16 | Current Skew: 94.15 | Change:
0.82
Index: 134308 | Value: 169.00 | Drop Count: 17 | Current Skew: 93.88 | Change:
0.27
Index: 142221 | Value: 166.67 | Drop Count: 18 | Current Skew: 95.57 | Change:
-1.69
Index: 111816 | Value: 160.00 | Drop Count: 19 | Current Skew: 96.24 | Change:
-0.67
Index: 101059 | Value: 151.50 | Drop Count: 20 | Current Skew: 95.59 | Change:
0.65
Index:  91705 | Value: 143.00 | Drop Count: 21 | Current Skew: 92.91 | Change:
2.68
Index:  63568 | Value: 111.57 | Drop Count: 22 | Current Skew: 86.37 | Change:
6.54
Index:  61245 | Value: 111.55 | Drop Count: 23 | Current Skew: 84.15 | Change:
2.22
Index:  52731 | Value: 111.54 | Drop Count: 24 | Current Skew: 76.96 | Change:
7.19
Index:  39888 | Value: 87.00 | Drop Count: 25 | Current Skew: 57.20 | Change:
19.76
Index: 117046 | Value: 58.00 | Drop Count: 26 | Current Skew: 40.50 | Change:
16.70
Index:  46675 | Value: 51.50 | Drop Count: 27 | Current Skew: 34.57 | Change:
5.93
Index: 144267 | Value: 46.67 | Drop Count: 28 | Current Skew: 29.29 | Change:
5.28
Index: 139364 | Value: 37.00 | Drop Count: 29 | Current Skew: 24.36 | Change:
4.93
Index: 131750 | Value: 32.00 | Drop Count: 30 | Current Skew: 21.74 | Change:
2.62
Index:  83339 | Value: 32.00 | Drop Count: 31 | Current Skew: 19.94 | Change:
1.80
Index:  97635 | Value: 30.00 | Drop Count: 32 | Current Skew: 17.83 | Change:
2.11
Index: 129788 | Value: 28.46 | Drop Count: 33 | Current Skew: 15.90 | Change:
1.93
Index: 120882 | Value: 28.45 | Drop Count: 34 | Current Skew: 14.07 | Change:
1.83
Index: 113459 | Value: 25.00 | Drop Count: 35 | Current Skew: 11.98 | Change:
```

```
2.09
Index: 105166 | Value: 24.00 | Drop Count: 36 | Current Skew: 10.46 | Change:
1.52
Index: 133597 | Value: 23.00 | Drop Count: 37 | Current Skew: 9.00 | Change:
1.46
Index: 107556 | Value: 19.05 | Drop Count: 38 | Current Skew: 7.59 | Change:
1.41
Index: 113171 | Value: 19.00 | Drop Count: 39 | Current Skew: 6.78 | Change:
0.81
Index: 130928 | Value: 17.00 | Drop Count: 40 | Current Skew: 5.92 | Change:
0.86
Index: 120622 | Value: 15.50 | Drop Count: 41 | Current Skew: 5.28 | Change:
0.64
Index: 132913 | Value: 14.00 | Drop Count: 42 | Current Skew: 4.79 | Change:
0.49
Index:   2322 | Value: 13.79 | Drop Count: 43 | Current Skew: 4.42 | Change:
0.37
Index: 138240 | Value: 11.50 | Drop Count: 44 | Current Skew: 4.06 | Change:
0.36
Index:  53450 | Value: 10.66 | Drop Count: 45 | Current Skew: 3.86 | Change:
0.20
Index:  47287 | Value: 10.64 | Drop Count: 46 | Current Skew: 3.69 | Change:
0.17
Index:  40811 | Value: 10.63 | Drop Count: 47 | Current Skew: 3.53 | Change:
0.16
Index:   3907 | Value: 10.04 | Drop Count: 48 | Current Skew: 3.36 | Change:
0.17
Index:  20459 | Value: 9.74 | Drop Count: 49 | Current Skew: 3.22 | Change: 0.14
```

[62]:
```python
df.drop(df.DebtToIncomeRatio.sort_values(ascending=False).head(40).index,
 ↪inplace=True)
df.skew()
```

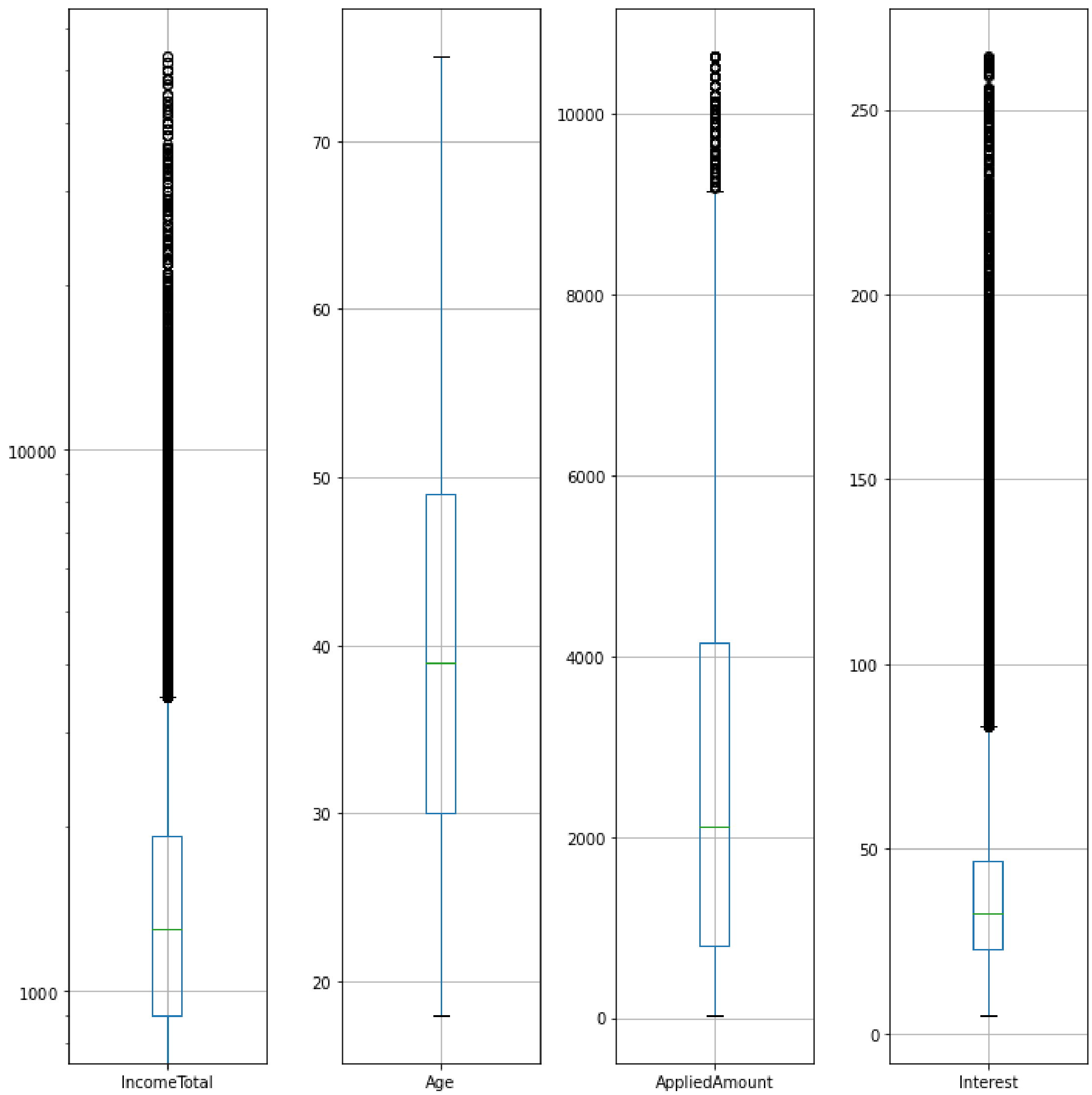
[62]:
```
IncomeTotal                              10.197471
Age                                       0.409744
AppliedAmount                             1.526441
Interest                                  3.341054
LoanDuration                             -0.753587
PreviousEarlyRepaymentsCountBeforeLoan    8.800809
DebtToIncomeRatio                         5.916551
dtype: float64
```

1.2.4 Descriptive Analysis

[63]:
```python
plt.rcParams['axes.formatter.min_exponent'] = 6

fig = plt.figure(dpi=140)
```

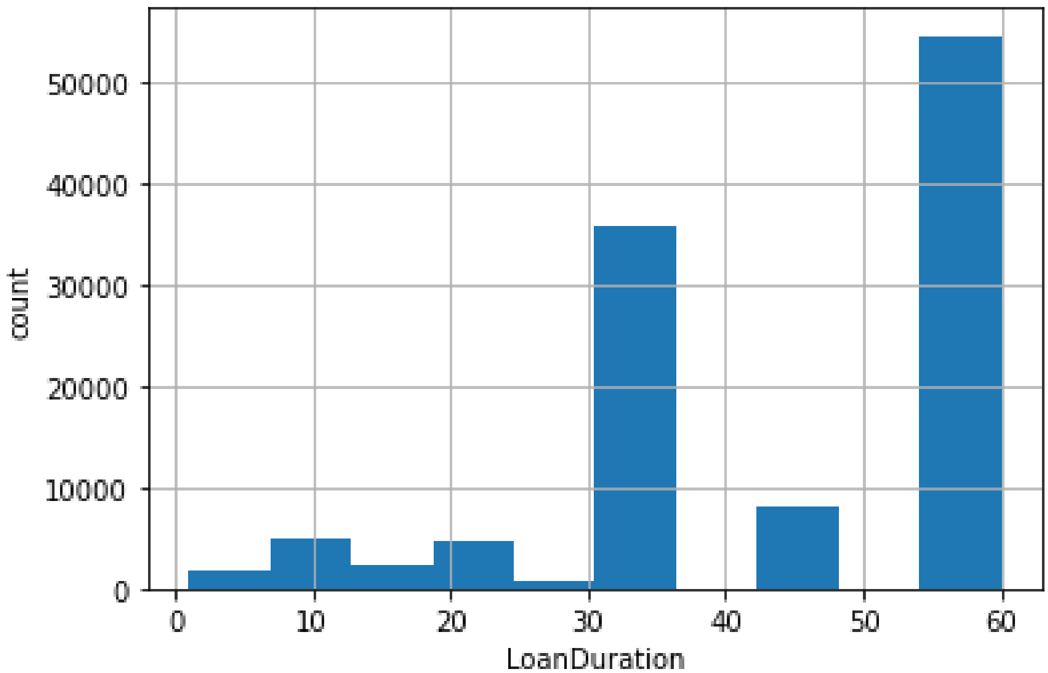
```
fig, axes = plt.subplots(nrows=1, ncols=4)

fig.set_size_inches(10, 10)
df.boxplot(column = 'IncomeTotal', ax = axes[0])
axes[0].set_yscale('log')
df.boxplot(column = 'Age', ax = axes[1])
df.boxplot(column = 'AppliedAmount', ax = axes[2])
df.boxplot(column = 'Interest', ax = axes[3])
plt.tight_layout()
plt.show()
```

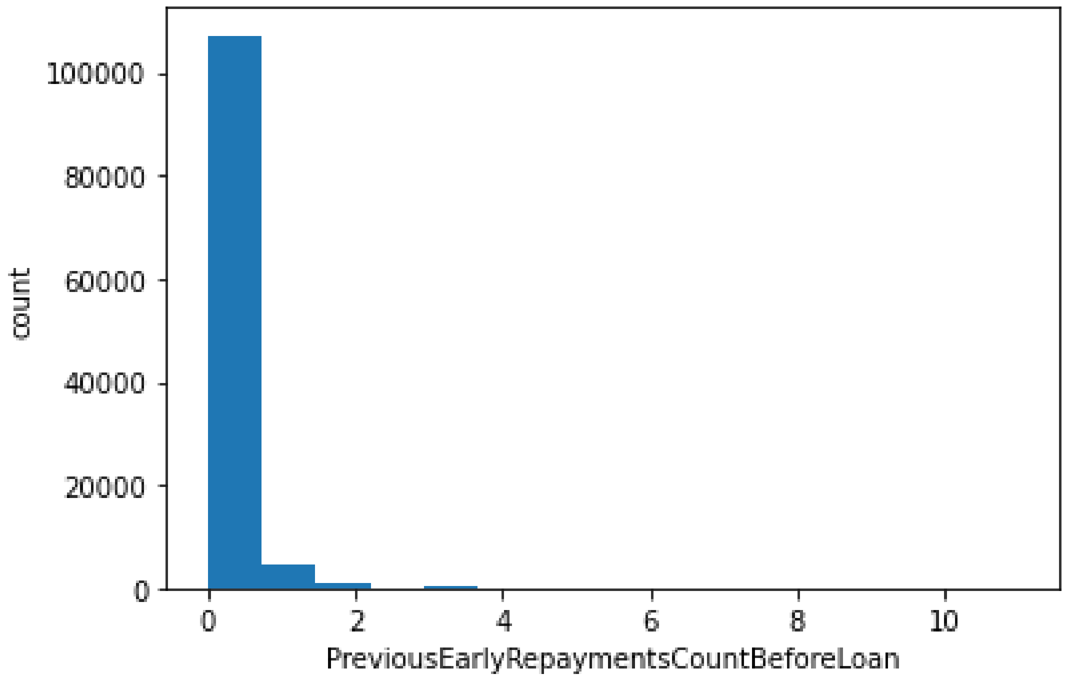
```
<Figure size 840x560 with 0 Axes>
```

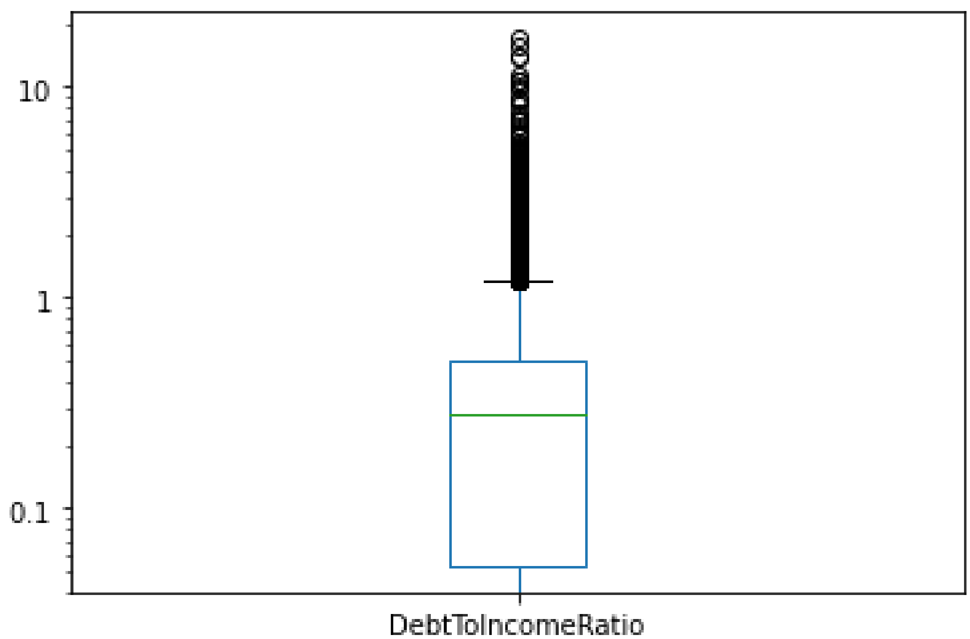


[64]:

```
ax = df.LoanDuration.hist()
ax.set_xlabel('LoanDuration')
ax.set_ylabel('count')
plt.show()

ax = df.PreviousEarlyRepaymentsCountBeforeLoan.plot.hist(bins=15)
ax.set_xlabel('PreviousEarlyRepaymentsCountBeforeLoan')
ax.set_ylabel('count')
plt.show()

ax = df.DebtToIncomeRatio.plot.box()
ax.set_yscale('log')
```

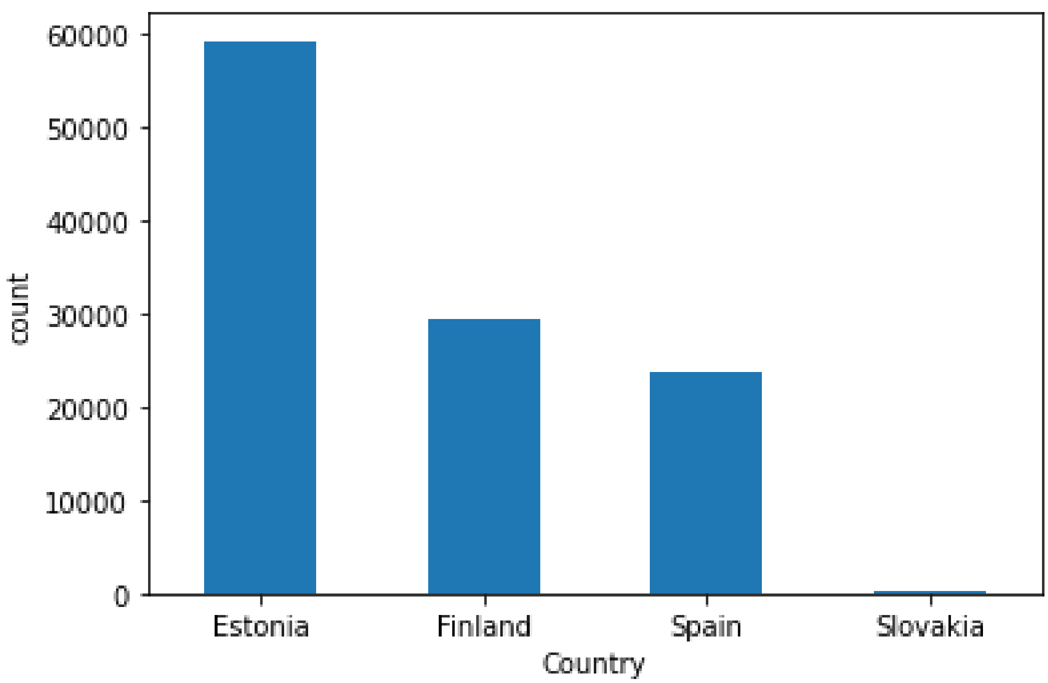


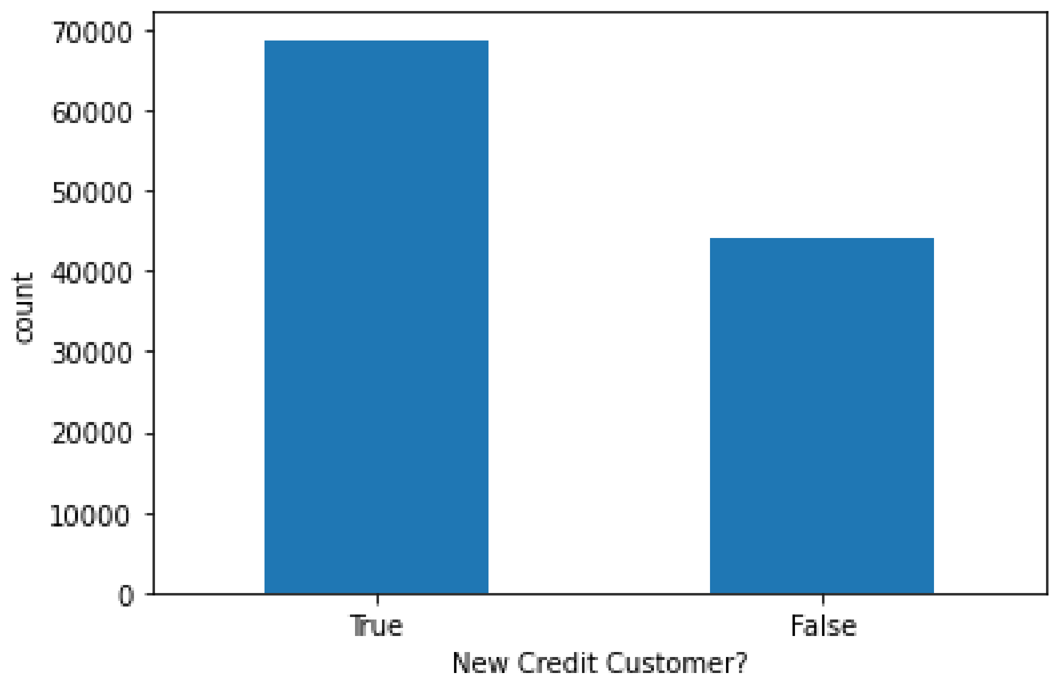


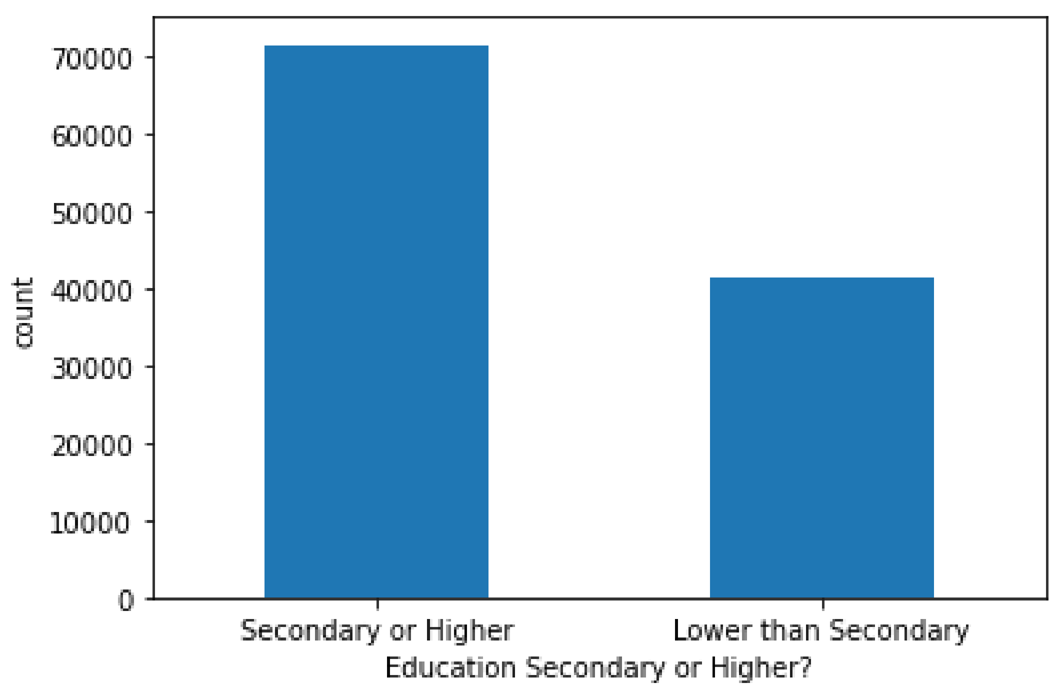


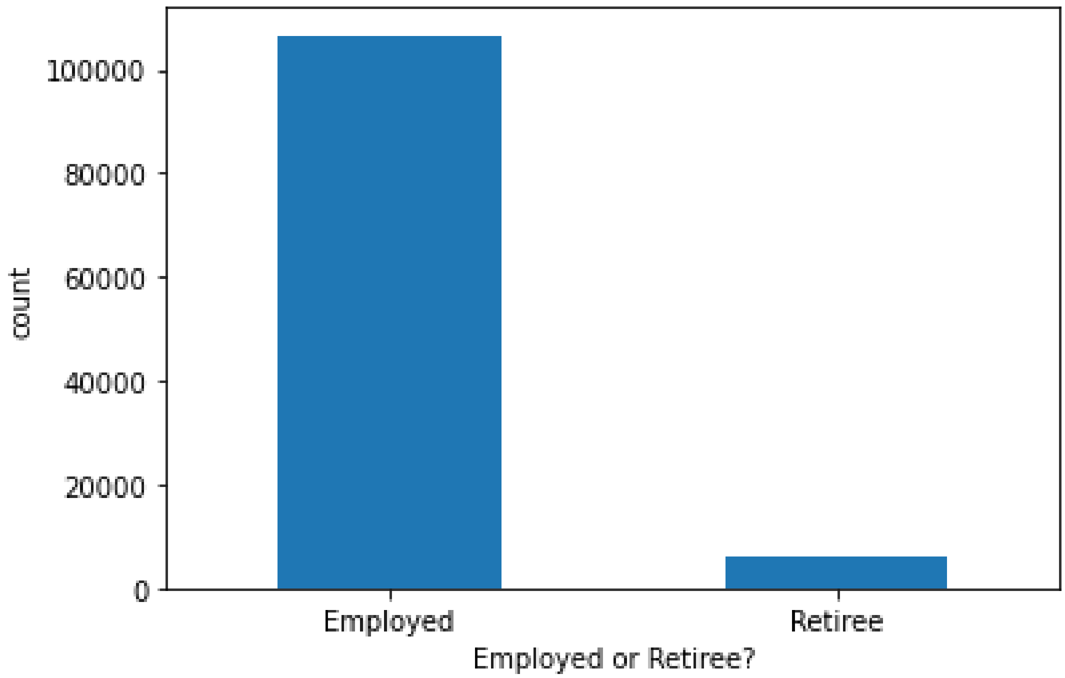
[65]:
```
ax = df.Country.value_counts().plot.bar(rot=0)
ax.set_xlabel('Country')
ax.set_ylabel('count')
```

```
plt.show()

ax = df.NewCreditCustomer.value_counts().plot.bar(rot=0)
ax.set_xlabel('New Credit Customer?')
ax.set_ylabel('count')
plt.show()

ax = df.EducationSecondaryOrHigher.value_counts().rename(index={1:'Secondary or␣
 ↪Higher',
                                                        0: 'Lower than␣
 ↪Secondary'}).plot.bar(rot=0)
ax.set_xlabel('Education Secondary or Higher?')
ax.set_ylabel('count')
plt.show()

ax = df.EmploymentStatusRetiree.value_counts().rename(index={1:'Retiree',
                                                        0: 'Employed'}).plot.
 ↪bar(rot=0)
ax.set_xlabel('Employed or Retiree?')
ax.set_ylabel('count')
plt.show()
```




70000
60000
50000
40000
30000
20000
10000
0
count
True
False
New Credit Customer?


70000
60000
50000
40000
30000
20000
10000
0
count
Secondary or Higher
Lower than Secondary
Education Secondary or Higher?



1.2.5 Normalizing Numericals

Since we plan to use KNN, we'll need a normalized rendition of the numerical variables.

[66]:
```
# Create a normalized variant of the numericals, needed for KNN
numerical_predictors = df.select_dtypes(exclude='category').dtypes

print(numerical_predictors)
print(f"\nThere are {numerical_predictors.shape[0]} numerical variables here.")

scaler = preprocessing.StandardScaler()

normalized_numericals = pd.DataFrame(scaler.
 ↪fit_transform(df[numerical_predictors.index]),
                                    index=df[numerical_predictors.index].index,
                                    columns=df[numerical_predictors.index].
 ↪columns)

df = pd.concat([df, normalized_numericals.add_prefix("norm_")], axis=1)
```

```
IncomeTotal                               int64
Age                                       int64
AppliedAmount                           float64
Interest                                float64
LoanDuration                              int64
```

```
PreviousEarlyRepaymentsCountBeforeLoan      int64
DebtToIncomeRatio                         float64
dtype: object

There are 7 numerical variables here.
```

1.2.6 Converting Categoricals to Dummies

Since we plan to use KNN, Naive Bayes, and CART, we need to create dummy variants of the categoricals (see Pages 514, 549, and 588 of the textbook).

In the cases where predictors only have 2 categories (`NewCreditCustomer` and `Defaulted`), we drop a dummy. One variable is all we need to inform us about all the possible categories for that predictor. Note that gender is excluded due to the availability of an 'other' option.

[67]:
```
cat_predictors = df.select_dtypes(include='category').dtypes
```

[68]:
```
#pd.get_dummies(df, columns=["Defaulted", "NewCreditCustomer"],␣
↪drop_first=True, prefix="is").dtypes

# equivalent of using get_dummies and dropping the first
df = df.astype({"Defaulted":"uint8",
                "NewCreditCustomer":"uint8",
                "EducationSecondaryOrHigher":"uint8",
                "EmploymentStatusRetiree":"uint8"})
```

[69]:
```
#update the remaining category predictor list
cat_predictors = df.select_dtypes(include='category').dtypes
print(cat_predictors)

df = pd.concat([df, pd.get_dummies(df[cat_predictors.index]).
↪add_prefix("gd_")], axis=1)
```

```
Country    category
dtype: object
```

[70]:
```
# Use gd_ to denote the dummies variant of the categorical variables
# with multiple categories.
df.dtypes
```

[70]:
```
IncomeTotal                                  int64
NewCreditCustomer                            uint8
Age                                          int64
Country                                   category
AppliedAmount                              float64
Interest                                   float64
LoanDuration                                 int64
PreviousEarlyRepaymentsCountBeforeLoan       int64
Defaulted                                    uint8
```

```
DebtToIncomeRatio                                   float64
EducationSecondaryOrHigher                            uint8
EmploymentStatusRetiree                               uint8
norm_IncomeTotal                                    float64
norm_Age                                            float64
norm_AppliedAmount                                  float64
norm_Interest                                       float64
norm_LoanDuration                                   float64
norm_PreviousEarlyRepaymentsCountBeforeLoan         float64
norm_DebtToIncomeRatio                              float64
gd_Country_Estonia                                    uint8
gd_Country_Finland                                    uint8
gd_Country_Slovakia                                   uint8
gd_Country_Spain                                      uint8
dtype: object
```

Converting Numericals to Categorical Bins Since we're using Naive Bayes, which requires all categorical variables, we need to bin our numerical values.

Fortunately pandas provides functions that can do the binning for us. https://pbpython.com/pandas-qcut-cut.html

Since LoanDuration and PreviousEarlyRepaymentsCountBeforeLoan are limited in range, we'll address them differently.

[71]:
```
numerical_predictors
```

[71]:
```
IncomeTotal                                   int64
Age                                           int64
AppliedAmount                               float64
Interest                                    float64
LoanDuration                                  int64
PreviousEarlyRepaymentsCountBeforeLoan        int64
DebtToIncomeRatio                           float64
dtype: object
```

IncomeTotal, Age, AppliedAmount, Interest, DebtToIncomeRatio For these numerical values we'll split them into four bins. qcut will return a bin each containing 25% of the data.

[72]:
```
for predictor in numerical_predictors.drop(["LoanDuration",␣
↪"PreviousEarlyRepaymentsCountBeforeLoan"]).index:
    print(pd.qcut(df[predictor], q=4).value_counts())
    df['bin_' + predictor] = pd.qcut(df[predictor], q=4)
```

```
(-0.001, 900.0]      30120
(900.0, 1300.0]      28977
(1930.0, 53000.0]    28187
(1300.0, 1930.0]     25552
Name: IncomeTotal, dtype: int64
```

```
(30.0, 39.0]      30206
(17.999, 30.0]    29068
(39.0, 49.0]      26845
(49.0, 75.0]      26717
Name: Age, dtype: int64
(800.0, 2125.0]      31530
(31.955, 800.0]      28301
(2125.0, 4150.0]     27959
(4150.0, 10632.0]    25046
Name: AppliedAmount, dtype: int64
(4.999, 22.78]     28230
(46.98, 264.31]    28208
(22.78, 32.6]      28202
(32.6, 46.98]      28196
Name: Interest, dtype: int64
(-0.001, 0.0533]    28215
(0.277, 0.508]      28210
(0.508, 17.0]       28207
(0.0533, 0.277]     28204
Name: DebtToIncomeRatio, dtype: int64
```

LoanDuration Loans range between 1 and 60 months. We can create bins representing 12-month intervals.

[73]:
```
loan_bin = pd.cut(df.LoanDuration, bins=np.linspace(0, 60, 6))

print(loan_bin.value_counts())

df['bin_LoanDuration'] = loan_bin
```

```
(48.0, 60.0]    54637
(24.0, 36.0]    36405
(36.0, 48.0]     8213
(12.0, 24.0]     6919
(0.0, 12.0]      6662
Name: LoanDuration, dtype: int64
```

PreviousEarlyRepaymentsCountBeforeLoan Most people have 0 early repaid loans. A small few have 1, and even less beyond that.

[74]:
```
early_repayments_bin = pd.cut(df.PreviousEarlyRepaymentsCountBeforeLoan,
                             bins=[-1, 0, 1,
                             df.PreviousEarlyRepaymentsCountBeforeLoan.max()])

df['bin_PreviousEarlyRepaymentsCountBeforeLoan'] = early_repayments_bin
```

Converting the Bins into Binary Categoricals

[75]:
```
bins = [col for col in df.columns if 'bin_' in col]
df = pd.concat([df, pd.get_dummies(df[bins]).add_prefix("gd_")], axis=1)
```

[76]:
```
print(df.dtypes.shape)

for i, d in enumerate(df.dtypes.to_dict()):
    print(f'{i}: {d}')
```

```
(58,)
0: IncomeTotal
1: NewCreditCustomer
2: Age
3: Country
4: AppliedAmount
5: Interest
6: LoanDuration
7: PreviousEarlyRepaymentsCountBeforeLoan
8: Defaulted
9: DebtToIncomeRatio
10: EducationSecondaryOrHigher
11: EmploymentStatusRetiree
12: norm_IncomeTotal
13: norm_Age
14: norm_AppliedAmount
15: norm_Interest
16: norm_LoanDuration
17: norm_PreviousEarlyRepaymentsCountBeforeLoan
18: norm_DebtToIncomeRatio
19: gd_Country_Estonia
20: gd_Country_Finland
21: gd_Country_Slovakia
22: gd_Country_Spain
23: bin_IncomeTotal
24: bin_Age
25: bin_AppliedAmount
26: bin_Interest
27: bin_DebtToIncomeRatio
28: bin_LoanDuration
29: bin_PreviousEarlyRepaymentsCountBeforeLoan
30: gd_bin_IncomeTotal_(-0.001, 900.0]
31: gd_bin_IncomeTotal_(900.0, 1300.0]
32: gd_bin_IncomeTotal_(1300.0, 1930.0]
33: gd_bin_IncomeTotal_(1930.0, 53000.0]
34: gd_bin_Age_(17.999, 30.0]
35: gd_bin_Age_(30.0, 39.0]
36: gd_bin_Age_(39.0, 49.0]
37: gd_bin_Age_(49.0, 75.0]
38: gd_bin_AppliedAmount_(31.955, 800.0]
```

```
39: gd_bin_AppliedAmount_(800.0, 2125.0]
40: gd_bin_AppliedAmount_(2125.0, 4150.0]
41: gd_bin_AppliedAmount_(4150.0, 10632.0]
42: gd_bin_Interest_(4.999, 22.78]
43: gd_bin_Interest_(22.78, 32.6]
44: gd_bin_Interest_(32.6, 46.98]
45: gd_bin_Interest_(46.98, 264.31]
46: gd_bin_DebtToIncomeRatio_(-0.001, 0.0533]
47: gd_bin_DebtToIncomeRatio_(0.0533, 0.277]
48: gd_bin_DebtToIncomeRatio_(0.277, 0.508]
49: gd_bin_DebtToIncomeRatio_(0.508, 17.0]
50: gd_bin_LoanDuration_(0.0, 12.0]
51: gd_bin_LoanDuration_(12.0, 24.0]
52: gd_bin_LoanDuration_(24.0, 36.0]
53: gd_bin_LoanDuration_(36.0, 48.0]
54: gd_bin_LoanDuration_(48.0, 60.0]
55: gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(-1, 0]
56: gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(0, 1]
57: gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(1, 11]
```

1.2.7 Get a Subset of Records

113,000 records is too much. We can probably cut it down to 15% of the dataset to build a sufficient model.

[77]:
```python
# subset is the percentage of records we'd like to keep.
subset = 15
n_sample_size = df.shape[0] * (subset / 100)
n_sample_size = round(n_sample_size)

df = df.sample(n_sample_size, random_state=1) #keep random_state same for␣
↪testing purposes

df.shape
```

[77]:
```
(16925, 58)
```

1.2.8 Prepare Datasets for Classification Methodologies

`df` as it is right is a massive amalgamation of the various variables we will need. Let's split it up for easier use.

These are three variables that didn't make it into the set because they're binary categoricals.

[78]:
```python
binary_categoricals = ["Defaulted", "NewCreditCustomer",␣
↪"EducationSecondaryOrHigher", "EmploymentStatusRetiree"]
```

KNN Requires: - Normalized Variables - Dummy Categoricals - all binary categorical variables, which have no prefix (as you'll see below)

[79]:
```python
knn_df_cols = [col for col in df.columns if 'norm_' in col or ('gd_' in col and␣
↪'bin_' not in col)]
knn_df_cols.extend(binary_categoricals)


for i, k in enumerate(knn_df_cols):
    print(i, k)

knn_df = df[knn_df_cols]
knn_df.shape
```

```
0 norm_IncomeTotal
1 norm_Age
2 norm_AppliedAmount
3 norm_Interest
4 norm_LoanDuration
5 norm_PreviousEarlyRepaymentsCountBeforeLoan
6 norm_DebtToIncomeRatio
7 gd_Country_Estonia
8 gd_Country_Finland
9 gd_Country_Slovakia
10 gd_Country_Spain
11 Defaulted
12 NewCreditCustomer
13 EducationSecondaryOrHigher
14 EmploymentStatusRetiree
```

[79]:
```
(16925, 15)
```

Naive Bayes Requires: - All Dummy Categoricals (Binned numericals) - all binary categorical variables, which have no prefix (as you'll see below)

[80]:
```python
bayes_df_cols = [col for col in df.columns if ('gd_' in col)]
bayes_df_cols.extend(binary_categoricals)

for i, k in enumerate(bayes_df_cols):
     print(i, k)

bayes_df = df[bayes_df_cols]
bayes_df.shape
```

```
0 gd_Country_Estonia
1 gd_Country_Finland
2 gd_Country_Slovakia
3 gd_Country_Spain
4 gd_bin_IncomeTotal_(-0.001, 900.0]
5 gd_bin_IncomeTotal_(900.0, 1300.0]
6 gd_bin_IncomeTotal_(1300.0, 1930.0]
```

```
7 gd_bin_IncomeTotal_(1930.0, 53000.0]
8 gd_bin_Age_(17.999, 30.0]
9 gd_bin_Age_(30.0, 39.0]
10 gd_bin_Age_(39.0, 49.0]
11 gd_bin_Age_(49.0, 75.0]
12 gd_bin_AppliedAmount_(31.955, 800.0]
13 gd_bin_AppliedAmount_(800.0, 2125.0]
14 gd_bin_AppliedAmount_(2125.0, 4150.0]
15 gd_bin_AppliedAmount_(4150.0, 10632.0]
16 gd_bin_Interest_(4.999, 22.78]
17 gd_bin_Interest_(22.78, 32.6]
18 gd_bin_Interest_(32.6, 46.98]
19 gd_bin_Interest_(46.98, 264.31]
20 gd_bin_DebtToIncomeRatio_(-0.001, 0.0533]
21 gd_bin_DebtToIncomeRatio_(0.0533, 0.277]
22 gd_bin_DebtToIncomeRatio_(0.277, 0.508]
23 gd_bin_DebtToIncomeRatio_(0.508, 17.0]
24 gd_bin_LoanDuration_(0.0, 12.0]
25 gd_bin_LoanDuration_(12.0, 24.0]
26 gd_bin_LoanDuration_(24.0, 36.0]
27 gd_bin_LoanDuration_(36.0, 48.0]
28 gd_bin_LoanDuration_(48.0, 60.0]
29 gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(-1, 0]
30 gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(0, 1]
31 gd_bin_PreviousEarlyRepaymentsCountBeforeLoan_(1, 11]
32 Defaulted
33 NewCreditCustomer
34 EducationSecondaryOrHigher
35 EmploymentStatusRetiree
```

[80]:
```
(16925, 36)
```

CART Requires: - Dummy categoricals - Can use non-normalized numericals

[81]:
```python
df_no_cat_columns = df.select_dtypes(exclude='category').dtypes.index

cart_df_cols = [col for col in df_no_cat_columns if ('norm_' not in col) and␣
↪'bin' not in col]

for i, k in enumerate(cart_df_cols):
    print(i, k)

cart_df = df[cart_df_cols]
cart_df.shape
```

```
0 IncomeTotal
1 NewCreditCustomer
2 Age
```

```
3 AppliedAmount
4 Interest
5 LoanDuration
6 PreviousEarlyRepaymentsCountBeforeLoan
7 Defaulted
8 DebtToIncomeRatio
9 EducationSecondaryOrHigher
10 EmploymentStatusRetiree
11 gd_Country_Estonia
12 gd_Country_Finland
13 gd_Country_Slovakia
14 gd_Country_Spain
```

[81]: (16925, 15)

1.3 Further Considerations for Preprocessing

Because of dummies expanding the dataset to a large dimension, we might want to tune the dataset as we test the various classification algorithms. Some things we can do are: - Test model on variations of `Education` or `EmploymentDuration` and consider combining some categories - Test model without 'Gender'; although some related studies used this, basing a model on gender may be unethical

1.4 [Milestone 2] Describing the Dataset

One of the milestone 2 requirements is to describe the dataset. First, let's list the original variables that we were using – this is the same list from section 2.1.9 of this report. Let's review it:

[82]:
```
df_chosen_preds.shape
for i, k in enumerate(df_chosen_preds):
    print(i+1, k)
```

```
1 IncomeTotal
2 NewCreditCustomer
3 Age
4 Country
5 AppliedAmount
6 Interest
7 LoanDuration
8 PreviousEarlyRepaymentsCountBeforeLoan
9 Defaulted
10 DebtToIncomeRatio
11 EducationSecondaryOrHigher
12 EmploymentStatusRetiree
```

We present a variation of the dataframe which has non-normalized numerical variables and all categoricals converted into dummies.

Datasets with binned variants / normalized variants of the numerical variables can be viewed in section 2.7 "Prepare Datasets for Classification Methodologies."

[83]:
```python
#A list of the variables
print(cart_df.dtypes)
print(cart_df.dtypes.shape)
```

```
IncomeTotal                               int64
NewCreditCustomer                         uint8
Age                                       int64
AppliedAmount                           float64
Interest                                float64
LoanDuration                              int64
PreviousEarlyRepaymentsCountBeforeLoan    int64
Defaulted                                 uint8
DebtToIncomeRatio                       float64
EducationSecondaryOrHigher                uint8
EmploymentStatusRetiree                   uint8
gd_Country_Estonia                        uint8
gd_Country_Finland                        uint8
gd_Country_Slovakia                       uint8
gd_Country_Spain                          uint8
dtype: object
(15,)
```

[84]:
```python
# describe() -- summary statistics across the dataset.
#pd.set_option('display.max_colwidth', None)
cart_df.describe()
```

[84]:
```
        IncomeTotal  NewCreditCustomer           Age  AppliedAmount  \
count  16925.000000       16925.000000  16925.000000   16925.000000
mean    1681.728449           0.608390     40.093885    2731.822947
std     2105.797059           0.488125     12.190920    2362.484660
min        0.000000           0.000000     18.000000      31.955800
25%      900.000000           0.000000     30.000000     850.000000
50%     1300.000000           1.000000     39.000000    2125.000000
75%     1940.000000           1.000000     49.000000    4146.000000
max    50000.000000           1.000000     70.000000   10632.000000

           Interest  LoanDuration  PreviousEarlyRepaymentsCountBeforeLoan  \
count  16925.000000  16925.000000                            16925.000000
mean      38.957504     45.849099                                0.070960
std       27.621186     15.672460                                0.382631
min        6.000000      1.000000                                0.000000
25%       22.930000     36.000000                                0.000000
50%       32.810000     48.000000                                0.000000
75%       47.220000     60.000000                                0.000000
max      263.590000     60.000000                                9.000000

          Defaulted  DebtToIncomeRatio  EducationSecondaryOrHigher  \
count  16925.000000       16925.000000                16925.000000
```

```
mean        0.592851            0.337393                      0.635923
std         0.491318            0.372367                      0.481185
min         0.000000            0.000000                      0.000000
25%         0.000000            0.056566                      0.000000
50%         1.000000            0.279541                      1.000000
75%         1.000000            0.509167                      1.000000
max         1.000000           17.000000                      1.000000

       EmploymentStatusRetiree  gd_Country_Estonia  gd_Country_Finland  \
count             16925.000000        16925.000000        16925.000000
mean                  0.058434            0.519645            0.262806
std                   0.234570            0.499629            0.440171
min                   0.000000            0.000000            0.000000
25%                   0.000000            0.000000            0.000000
50%                   0.000000            1.000000            0.000000
75%                   0.000000            1.000000            1.000000
max                   1.000000            1.000000            1.000000

       gd_Country_Slovakia  gd_Country_Spain
count         16925.000000      16925.000000
mean              0.002363          0.215185
std               0.048558          0.410963
min               0.000000          0.000000
25%               0.000000          0.000000
50%               0.000000          0.000000
75%               0.000000          0.000000
max               1.000000          1.000000
```

1.5 Applying the Models

We have a series of models to test on our dataset. 1. K-Nearest Neighbors 2. Classification Trees 3. Naive Bayes' Classifier

We must remember to consider the following as we test these models: - Use K-Folds Cross Validation - Visualize an ROC Curve - Use `dbma`'s confusion matrix - Use `scikit-learn`'s metrics class in order to acquire sensitivity, precision, accuracy, and recall scores.

1.5.1 Naive Rule

The *naive rule* refers to a benchmark in which each record is classified as the majority class. We're trying to classify `Defaulted` here, so let's take a look at the frequency distribution for the attribute. Remember that:

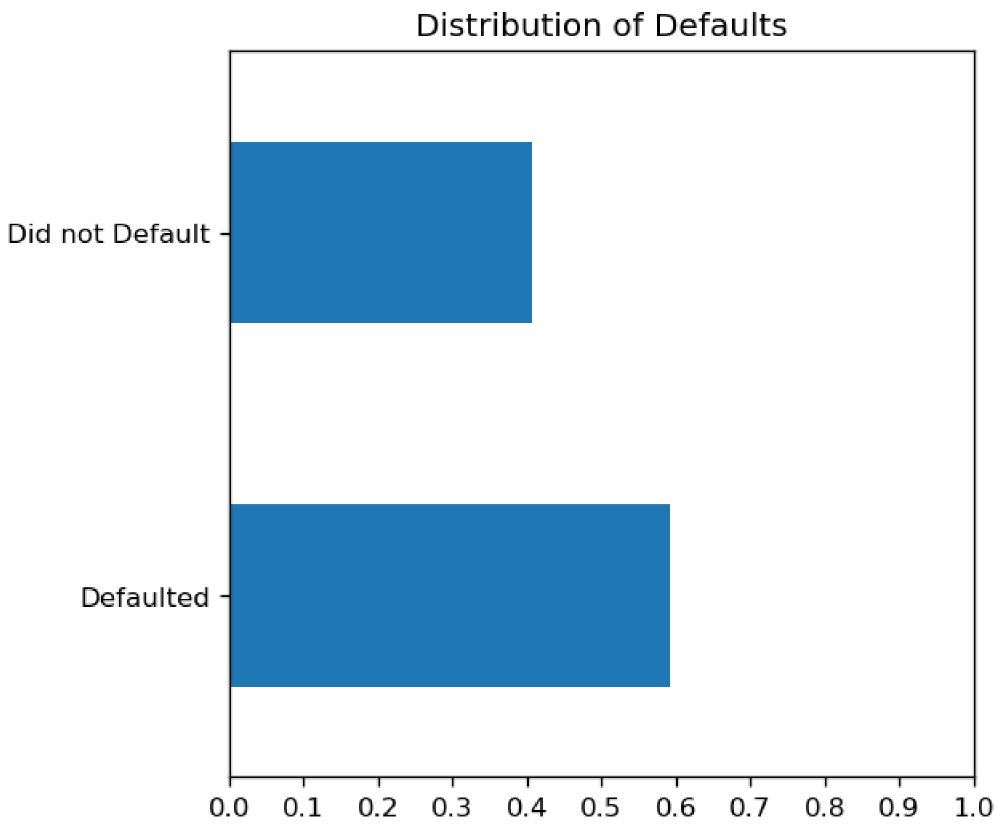
```
1 = Defaulted
0 = Did not Default
```

[85]:
```
df.Defaulted.value_counts(normalize=True).round(3) * 100
```

[85]:
```
1    59.3
0    40.7
Name: Defaulted, dtype: float64
```

[126]:
```
fig = plt.figure(figsize=(5,5), dpi=120)
df.Defaulted.value_counts(normalize=True).rename(index={1:'Defaulted',
                                                        0: 'Did not Default'}).
 ↪plot(kind='barh',
                                                                                 ␣
 ↪    xlim=[0,1.0],
                                                                                 ␣
 ↪    xticks=np.arange(0, 1.1, 0.1))
plt.title("Distribution of Defaults")
```

[126]:
```
Text(0.5, 1.0, 'Distribution of Defaults')
```



Given this, our **naive rule is that each record belongs to the defaulted class.** About 60% of the entire dataset.

However, because of asymmetric costs, we need to focus on **specificity**, which is the ratio of true negatives accurately classified.

So the goal to beat for our classification methods are 60% accuracy, oriented towards **maximizing specificity** and **precision**.

(i.e. We want the highest number of true negatives and the least amount of false positives.)

[87]:
```
# Imports.
from sklearn.tree import DecisionTreeClassifier # The model
from sklearn.metrics import accuracy_score # Metrics
from sklearn.metrics import confusion_matrix # Metrics
from sklearn.metrics import classification_report

from sklearn.model_selection import (train_test_split, cross_val_score,
cross_val_predict, GridSearchCV, RandomizedSearchCV) # for splitting and␣
↪avoiding overfitting on model
from dmba import plotDecisionTree, classificationSummary # Metrics from DBMA
from sklearn.ensemble import RandomForestClassifier

# Import module for KNN
from sklearn.neighbors import KNeighborsClassifier
```

1.5.2 KNN Approach

[88]:
```
knn_X = knn_df.drop(columns='Defaulted')
knn_y = knn_df['Defaulted']
```

[89]:
```
%pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
#SMOTE
sm = SMOTE(random_state=777)
knn_X_smote, knn_y_smote = sm.fit_resample(knn_X,knn_y)
```

```
Requirement already satisfied: imbalanced-learn in
c:\programdata\anaconda3\lib\site-packages (0.7.0)
Requirement already satisfied: scikit-learn>=0.23 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (0.23.2)
Requirement already satisfied: numpy>=1.13.3 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.19.2)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (0.17.0)
Requirement already satisfied: scipy>=0.19.1 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.23->imbalanced-
learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[90]:
```
train_knn_X, valid_knn_X, train_knn_y, valid_knn_y =␣
↪train_test_split(knn_X_smote,knn_y_smote,test_size=0.3,random_state=101)
```

[91]:
```python
# n_neighbors -> argument identifies the amount of neighbors used to ID␣
↪classification
knn = KNeighborsClassifier(n_neighbors=5)

# param_grid = {
#     'n_neighbors': np.arange(1, 21, 1)
# }
# gridSearch = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,
#                           n_jobs=-1)
# gridSearch.fit(train_knn_X, train_knn_y)
# print('Improved score: ', gridSearch.best_score_)
# print('Improved parameters: ', gridSearch.best_params_)


# knn_search_params = gridSearch.best_estimator_

#knn_search_params = knn.fit(train_knn_X, train_knn_y)

knn.fit(train_knn_X, train_knn_y)
```

[91]:
```
KNeighborsClassifier()
```

[92]:
```python
# Training run
#y_pred = knn_search_params.predict(train_knn_X)
y_pred = knn.predict(train_knn_X)

print(classificationSummary(train_knn_y,y_pred))
print(classification_report(train_knn_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.7716)

       Prediction
Actual    0    1
     0 5512 1472
     1 1736 5327
None
              precision    recall  f1-score   support

           0       0.76      0.79      0.77      6984
           1       0.78      0.75      0.77      7063

    accuracy                           0.77     14047
   macro avg       0.77      0.77      0.77     14047
weighted avg       0.77      0.77      0.77     14047
```

[93]:
```
#y_pred = knn_search_params.predict(valid_knn_X)
y_pred = knn.predict(valid_knn_X)

print(classificationSummary(valid_knn_y,y_pred))
print(classification_report(valid_knn_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.6517)

       Prediction
Actual    0    1
     0 2069  981
     1 1116 1855
None
              precision    recall  f1-score   support

           0       0.65      0.68      0.66      3050
           1       0.65      0.62      0.64      2971

    accuracy                           0.65      6021
   macro avg       0.65      0.65      0.65      6021
weighted avg       0.65      0.65      0.65      6021
```

1.5.3 Naive Bayes Approach

Additionally, we use SMOTE in order to balance the dataset before running Bayes. This improves its performance significantly.

[94]:
```
bayes_X = bayes_df.drop(columns="Defaulted")
bayes_y = bayes_df["Defaulted"]
```

[95]:
```
# If this pip doesn't work, you will need to install imbalanced-learn via conda.
# Place the following command into conda in admin mode:
# conda install -c conda-forge imbalanced-learn

%pip install imbalanced-learn
from imblearn.over_sampling import SMOTE
#SMOTE
sm = SMOTE(random_state=777)
bayes_X_smote, bayes_y_smote = sm.fit_resample(bayes_X ,bayes_y)
```

```
Requirement already satisfied: imbalanced-learn in
c:\programdata\anaconda3\lib\site-packages (0.7.0)
Requirement already satisfied: numpy>=1.13.3 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.19.2)
Requirement already satisfied: joblib>=0.11 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (0.17.0)
Requirement already satisfied: scikit-learn>=0.23 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (0.23.2)
```

```
Requirement already satisfied: scipy>=0.19.1 in
c:\programdata\anaconda3\lib\site-packages (from imbalanced-learn) (1.5.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from scikit-learn>=0.23->imbalanced-
learn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

[96]:
```
train_bayes_X, valid_bayes_X, train_bayes_y, valid_bayes_y =␣
↪train_test_split(bayes_X_smote,bayes_y_smote,test_size=0.3,random_state=1)
```

[97]:
```
#Import Gaussian Naive Bayes model
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
gnb = GaussianNB()

#Train the model using the training sets
gnb.fit(train_bayes_X, train_bayes_y)
```

[97]:
```
GaussianNB()
```

[98]:
```
# report for training set
y_pred = gnb.predict(train_bayes_X)

print(classificationSummary(train_bayes_y,y_pred))
print(classification_report(train_bayes_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.6064)

       Prediction
Actual    0    1
     0 1521 5529
     1    0 6997
None
              precision    recall  f1-score   support

           0       1.00      0.22      0.35      7050
           1       0.56      1.00      0.72      6997

    accuracy                           0.61     14047
   macro avg       0.78      0.61      0.54     14047
weighted avg       0.78      0.61      0.54     14047
```

[99]:
```
#Predict the response for test dataset
y_pred = gnb.predict(valid_bayes_X)

print(classificationSummary(valid_bayes_y,y_pred))
```

```
print(classification_report(valid_bayes_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.6168)

       Prediction
Actual    0    1
     0  677 2307
     1    0 3037
None
              precision    recall  f1-score   support

           0       1.00      0.23      0.37      2984
           1       0.57      1.00      0.72      3037

    accuracy                           0.62      6021
   macro avg       0.78      0.61      0.55      6021
weighted avg       0.78      0.62      0.55      6021
```

1.5.4 Classification Trees Approach

We follow steps from Chapter 6 of *Data Mining for Business Analytics*.

[100]:
```
cart_X = cart_df.drop(["Defaulted"], axis=1) #alternatively, axis=0 if you want␣
 ↪to drop records, not cols.
cart_y = cart_df["Defaulted"]
```

Classification Tree without Modifiers

[101]:
```
tree = DecisionTreeClassifier()
# cross_val_predict returns the predictions of the cross-validation fitted␣
 ↪model for each sample
# we then compare that with the actual y values and see if they're any good.
y_pred = cross_val_predict(tree, cart_X, cart_y, cv=10)

cm = confusion_matrix(cart_y, y_pred)

print(classificationSummary(cart_y,y_pred))

print(classification_report(cart_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.5891)

       Prediction
Actual    0    1
     0 3474 3417
     1 3537 6497
None
              precision    recall  f1-score   support
```

```
              0       0.50      0.50      0.50      6891
              1       0.66      0.65      0.65     10034

       accuracy                           0.59     16925
      macro avg       0.58      0.58      0.58     16925
   weighted avg       0.59      0.59      0.59     16925
```

Before heading back to the drawing board to cull variables, let's see if we can execute a grid search.

On page 615 of the book, it recommends to use cross validation on a training set to find the best tree, and then using that tree with the validation data to evaluate likely actual performance. For that reason we employ `train_test_split`

Classification Tree with Grid Search

[102]:
```
#SMOTE
sm = SMOTE(random_state=777)
cart_X_smote, cart_y_smote = sm.fit_resample(cart_X ,cart_y)
```

[103]:
```
train_cart_X, valid_cart_X, train_cart_y, valid_cart_y =␣
↪train_test_split(cart_X_smote, cart_y_smote, test_size=0.3, random_state=1)
```

[104]:
```
# First Grid Search. Caution! Processor-intensive so it takes a bit of time to␣
↪run.

param_grid = {
    'max_depth': np.arange(10, 60, 5),
    'min_samples_split': np.arange(10, 60, 5),
    'min_impurity_decrease': [0, 0.0005, 0.001, 0.005, 0.01],
}

gridSearch = GridSearchCV(DecisionTreeClassifier(), param_grid, cv=5, n_jobs=-1)

gridSearch.fit(cart_X, cart_y)
print('Initial Score: ', gridSearch.best_score_)
print('Initial Parameters: ', gridSearch.best_params_)
```

```
Initial Score:  0.6526440177252585
Initial Parameters:  {'max_depth': 10, 'min_impurity_decrease': 0.0005,
'min_samples_split': 10}
```

[105]:
```
# Improved Grid Search based on results of the first.
# Caution! Processor-intensive so it takes a bit of time to run.

param_grid = {
    'max_depth': np.arange(3, 14, 1),
    'min_samples_split': np.arange(5, 18, 1),
    'min_impurity_decrease': np.arange(0.0003, 0.0006, 0.00005), # 6 values
```

```
}
gridSearch = GridSearchCV(DecisionTreeClassifier(random_state=1), param_grid,␣
↪cv=5,
                          n_jobs=-1)
gridSearch.fit(train_cart_X, train_cart_y)
print('Improved score: ', gridSearch.best_score_)
print('Improved parameters: ', gridSearch.best_params_)
```

```
Improved score:  0.6639132985105071
Improved parameters:  {'max_depth': 9, 'min_impurity_decrease': 0.0004,
'min_samples_split': 5}
```

[106]:
```
# get the best estimator from the improved grid search.
tree_cv_search_params = gridSearch.best_estimator_
```

[107]:
```
# report for training set
y_pred = tree_cv_search_params.predict(train_cart_X)

print(classificationSummary(train_cart_y,y_pred))
print(classification_report(train_cart_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.6807)

       Prediction
Actual    0    1
     0 4638 2412
     1 2073 4924
None
              precision    recall  f1-score   support

           0       0.69      0.66      0.67      7050
           1       0.67      0.70      0.69      6997

    accuracy                           0.68     14047
   macro avg       0.68      0.68      0.68     14047
weighted avg       0.68      0.68      0.68     14047
```

[108]:
```
# Now use that tree on the validation data to evaluate likely performance.
y_pred = tree_cv_search_params.predict(valid_cart_X)

print(classificationSummary(valid_cart_y,y_pred))
print(classification_report(valid_cart_y,y_pred))
```

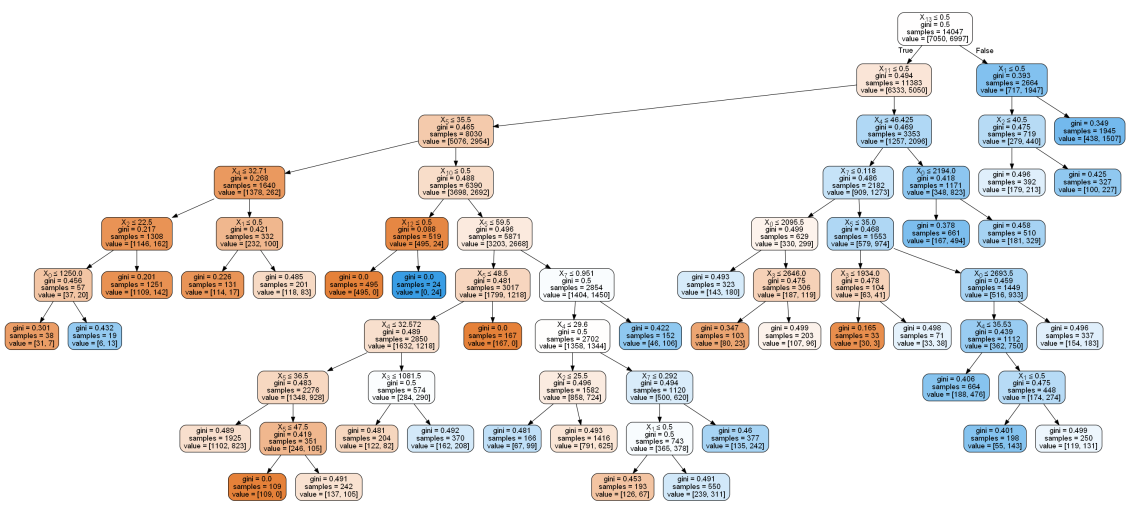
```
Confusion Matrix (Accuracy 0.6718)

       Prediction
Actual    0    1
     0 1959 1025
```

```
     1  951 2086
None
              precision    recall  f1-score   support

           0       0.67      0.66      0.66      2984
           1       0.67      0.69      0.68      3037

    accuracy                           0.67      6021
   macro avg       0.67      0.67      0.67      6021
weighted avg       0.67      0.67      0.67      6021
```

[109]:

```python
#Drawing the tree
#You need to install graphviz from https://graphviz.gitlab.io/download/
#if windows, select Stable windows installer - then, the .msi file, then install
#After installation, add the path for Graphviz bin folder to CLASSPATH␣
↪environement variable
#in Control panel/advanced setting/system properties/environmental variables
#!pip install --user pydotplus
from six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(tree_cv_search_params, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True)
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
Image(graph.create_png())
```

[109]:



Classification Trees via Random Forest Ensemble

[110]:
```python
# Let's run it through a randomized search

# Number of trees in random forest
n_estimators = [int(x) for x in np.linspace(start = 250, stop = 500, num = 10)]
# Number of features to consider at every split
max_features = ['auto', 'sqrt']
# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(0, 20, num = 2)]
max_depth.append(None)
# Minimum number of samples required to split a node
min_samples_split = [10, 11, 12]
# Minimum number of samples required at each leaf node
min_samples_leaf = [1, 2, 4]
# Method of selecting samples for training each tree
bootstrap = [True, False]
# Create the random grid
random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
```

[111]:
```python
#
# NOTE: This code has been commented out because it takes ~5 minutes to run.
#

#We preserve the output from before, however for later usage..

# rf_randomSearchCV = RandomizedSearchCV(RandomForestClassifier(), random_grid,
#                                        n_iter=100, cv=3, verbose=2, n_jobs=-1)
# rf_randomSearchCV.fit(train_cart_X, train_cart_y)
# print('Score: ', rf_randomSearchCV.best_score_)
# print('Parameters: ', rf_randomSearchCV.best_params_)


# Output
'''
Score:  0.685136323658751
Parameters:  {'n_estimators': 200,
              'min_samples_split': 12, 'min_samples_leaf': 1,
              'max_features': 'auto', 'max_depth': None, 'bootstrap': True}
'''
```

[111]:
```
"\nScore:  0.685136323658751\nParameters:  {'n_estimators': 200,\n
'min_samples_split': 12, 'min_samples_leaf': 1,\n              'max_features':
'auto', 'max_depth': None, 'bootstrap': True}\n"
```

[112]:
```
# print('Score: ', rf_randomSearchCV.best_score_)
# print('Parameters: ', rf_randomSearchCV.best_params_)
```

[113]:
```
rf_tree = RandomForestClassifier(n_estimators=388,
                          min_samples_split=2,
                          min_samples_leaf=1,
                          max_features='sqrt',
                          max_depth=9, #replace max_depth = None with 9 to prevent␣
 ↪overfitting
                          bootstrap=True)
rf_tree = rf_tree.fit(train_cart_X, train_cart_y)
```

[114]:
```
# report for training set
y_pred = rf_tree.predict(train_cart_X)
print(classificationSummary(train_cart_y,y_pred))
print(classification_report(train_cart_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.7286)

       Prediction
Actual    0    1
     0 5234 1816
     1 1996 5001
None
              precision    recall  f1-score   support

           0       0.72      0.74      0.73      7050
           1       0.73      0.71      0.72      6997

    accuracy                           0.73     14047
   macro avg       0.73      0.73      0.73     14047
weighted avg       0.73      0.73      0.73     14047
```

[115]:
```
y_pred = rf_tree.predict(valid_cart_X)
#y_pred = cross_val_predict(rf_tree, train_cart_X, train_cart_y, cv=10)

print(classificationSummary(valid_cart_y,y_pred))
print(classification_report(valid_cart_y,y_pred))
```

```
Confusion Matrix (Accuracy 0.6884)

       Prediction
Actual    0    1
     0 2095  889
     1  987 2050
None
              precision    recall  f1-score   support
```

```
           0       0.68      0.70      0.69      2984
           1       0.70      0.68      0.69      3037

    accuracy                           0.69      6021
   macro avg       0.69      0.69      0.69      6021
weighted avg       0.69      0.69      0.69      6021
```

1.6 Performance Summary

1.6.1 Accuracy, Precision, Sensitivity, and Specificity Comparisons

Negative (0): Not Default

Positive (1): Default

Note that this also is how sklearn and the textbook authors consider negative and positive, if you use `dbma.classificationSummary()` or `sklearn.classification_report()`.

Accuracy: The overall accuracy of the model. Accuracy = (TP + TN) / (TP + TN + FP + FN)

Precision: The model's ability to not label as positive a sample that is negative. Precision = (TP) / (TP + FP)

Sensitivity/Recall: The classifier's ability to detect the positive class, **default**, accurately. Sensitivity = (TP) / (TP + FN). This was recall for class 1 on `classification_report()`

Specificity: The classifier's ability to detect the negative class, **no default**, accurately. Specificity = (TN) / (TN + FP). This was recall for class 0 on `classification_report()`

[116]:
```
from sklearn.metrics import confusion_matrix

def get_APSS_metrics(model_name, model, valid_X, valid_y):
    y_pred = model.predict(valid_X)
    tn, fp, fn, tp = confusion_matrix(valid_y, y_pred).ravel()
    accuracy    = (tp + tn) / (tp + tn + fp + fn)
    precision   = (tp) / (tp + fp)
    specificity = (tn) / (tn + fp)
    sensitivity = (tp) / (tp + fn)
    print(f'''
    Classification Metrics (Higher is better)
    MODEL: {model_name}
    Accuracy:{accuracy:8.2f}
    Precision:{precision:7.2f}
    Sensitivity:{sensitivity:5.2f}
    Specificity:{specificity:5.2f}
    ''')

    classification_metric_dict = {
```

```
    "accuracy" : accuracy,
    "precision" : precision,
    "sensitivity" : sensitivity,
    "specificity" : specificity
    }

    return (model_name, classification_metric_dict)
```

Training Metrics Summary

[117]:
```
train_knn_metrics = get_APSS_metrics("KNN (Training)", knn,
                                     train_knn_X, train_knn_y)

train_bayes_metrics = get_APSS_metrics("Naive Bayes (Training)", gnb,
                 train_bayes_X, train_bayes_y)

train_tree_metrics = get_APSS_metrics("Classification Tree (Training)",␣
 ↪tree_cv_search_params,
                 train_cart_X, train_cart_y)

train_rf_metrics = get_APSS_metrics("Random Forest (Training)", rf_tree,
                 train_cart_X, train_cart_y)
```

```
Classification Metrics (Higher is better)
MODEL: KNN (Training)
Accuracy:    0.77
Precision:   0.78
Sensitivity: 0.75
Specificity: 0.79


Classification Metrics (Higher is better)
MODEL: Naive Bayes (Training)
Accuracy:    0.61
Precision:   0.56
Sensitivity: 1.00
Specificity: 0.22


Classification Metrics (Higher is better)
MODEL: Classification Tree (Training)
Accuracy:    0.68
Precision:   0.67
Sensitivity: 0.70
Specificity: 0.66
```

```
Classification Metrics (Higher is better)
MODEL: Random Forest (Training)
Accuracy:    0.73
Precision:   0.73
Sensitivity: 0.71
Specificity: 0.74
```

Validation Metrics Summary

[118]:
```
knn_metrics = get_APSS_metrics("KNN", knn,
                 valid_knn_X, valid_knn_y)

bayes_metrics = get_APSS_metrics("Naive Bayes", gnb,
                 valid_bayes_X, valid_bayes_y)

tree_metrics = get_APSS_metrics("Classification Tree", tree_cv_search_params,
                 valid_cart_X, valid_cart_y)

rf_metrics = get_APSS_metrics("Random Forest", rf_tree,
                 valid_cart_X, valid_cart_y)



# prep for metric_df
# adapted from https://stackoverflow.com/questions/62027382/
 ↪how-to-use-different-error-bars-for-grouped-data-of-a-pandas-dataframe

# vstack stacks arrays in sequence vertically.
# we just need to convert the dictionary's values to a list first.
```

```
Classification Metrics (Higher is better)
MODEL: KNN
Accuracy:    0.65
Precision:   0.65
Sensitivity: 0.62
Specificity: 0.68


Classification Metrics (Higher is better)
MODEL: Naive Bayes
Accuracy:    0.62
Precision:   0.57
Sensitivity: 1.00
Specificity: 0.23


Classification Metrics (Higher is better)
```

```
MODEL: Classification Tree
Accuracy:    0.67
Precision:   0.67
Sensitivity: 0.69
Specificity: 0.66


Classification Metrics (Higher is better)
MODEL: Random Forest
Accuracy:    0.69
Precision:   0.70
Sensitivity: 0.68
Specificity: 0.70
```

Visualization of Classification Metrics

[119]:
```
''' scratch for building the function
model_metrics_vals = np.vstack((list(tree_metrics[1].values()),
                                list(rf_metrics[1].values())))

# the indices will be the model names
model_metrics_index = [tree_metrics[0], rf_metrics[0]]

# the columns will be the metric names
model_metrics_cols  = list(tree_metrics[1].keys())

metric_df = pd.DataFrame(model_metrics_vals, index=model_metrics_index,
                                             columns=model_metrics_cols)

metric_df
'''

'''
Build a metric dataframe based on the return tuple from get_APSS_results.
args: metric_results - the tuple returned from get_APSS_results
'''
def build_metric_df(*metric_results):
    # get a tuple of the metrics to put into the vstack.
    metric_index_list = []
    metric_val_list = []
    metric_col_list = list(metric_results[0][1].keys())
    for metric in metric_results:
        metric_index_list.append(metric[0])
        #print(metric[1]) #metric[1] are the metrics for the model.
        metric_val_list.append(list(metric[1].values()))

    metric_df = pd.DataFrame(metric_val_list, index=metric_index_list,
```

```
                                                    columns=metric_col_list)

    return metric_df
```

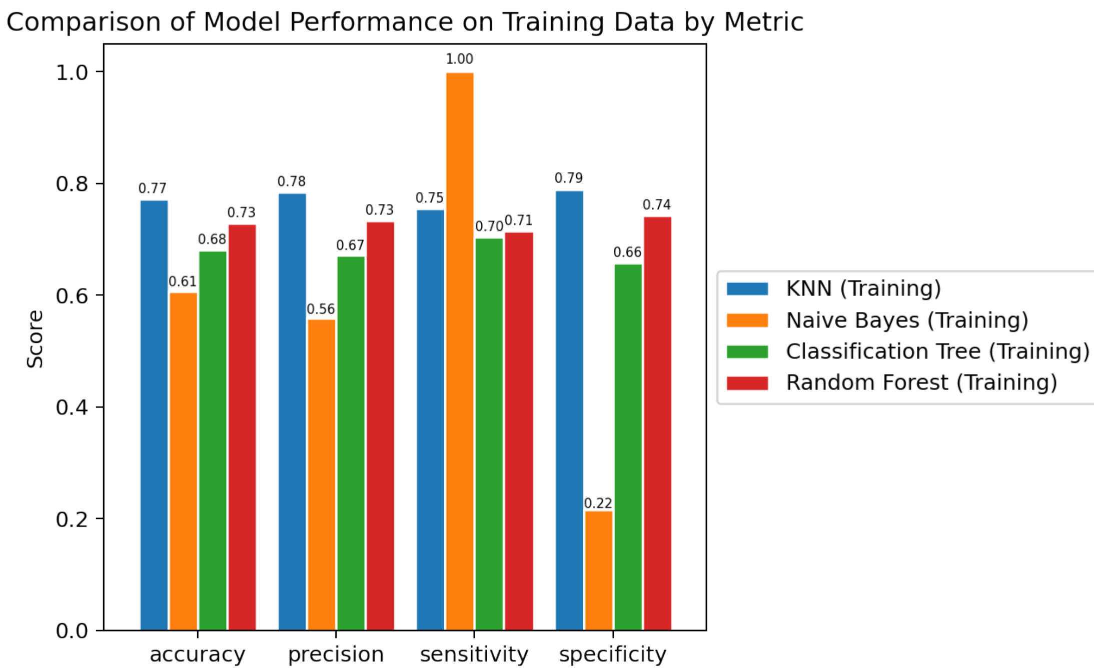
[120]:
```
train_metric_df = build_metric_df(train_knn_metrics, train_bayes_metrics,␣
 ↪train_tree_metrics, train_rf_metrics)
metric_df = build_metric_df(knn_metrics, bayes_metrics, tree_metrics,␣
 ↪rf_metrics)
metric_df
```

[120]:

	accuracy	precision	sensitivity	specificity
KNN	0.651719	0.654090	0.624369	0.678361
Naive Bayes	0.616841	0.568301	1.000000	0.226877
Classification Tree	0.671815	0.670524	0.686862	0.656501
Random Forest	0.688424	0.697516	0.675008	0.702078

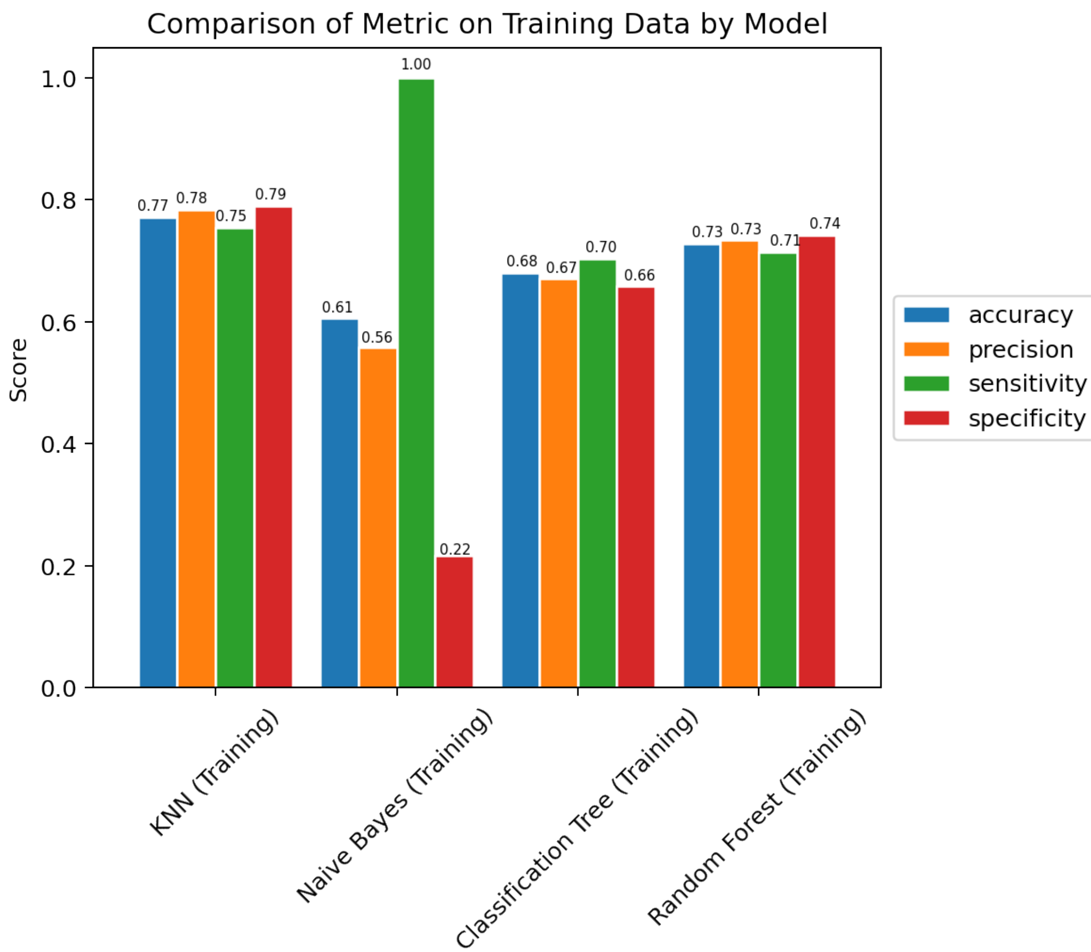
[121]:
```
fig = plt.figure(figsize=(5,5), dpi=180)

ax = train_metric_df.transpose().plot.bar(ylim=(0,1.05), ax = plt.gca(), rot=0,␣
 ↪width=0.85, edgecolor='white')

for p in ax.patches:
    ax.annotate(str(f"{p.get_height():.2f}"), (p.get_x() * 1.001, p.
 ↪get_height() * 1.015), fontsize=6)

plt.title("Comparison of Model Performance on Training Data by Metric")
plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
plt.ylabel("Score")
```

[121]: Text(0, 0.5, 'Score')



[122]:
```
fig = plt.figure(figsize=(6,5), dpi=180)

ax = train_metric_df.plot.bar(ylim=(0,1.05), ax = plt.gca(), rot=45, width=0.
↪85, edgecolor='white')

for p in ax.patches:
    ax.annotate(str(f"{p.get_height():.2f}"), (p.get_x() * 1.02, p.get_height()␣
↪* 1.015), fontsize=6)

plt.title("Comparison of Metric on Training Data by Model")
plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
plt.ylabel("Score")
```

[122]:
```
Text(0, 0.5, 'Score')
```

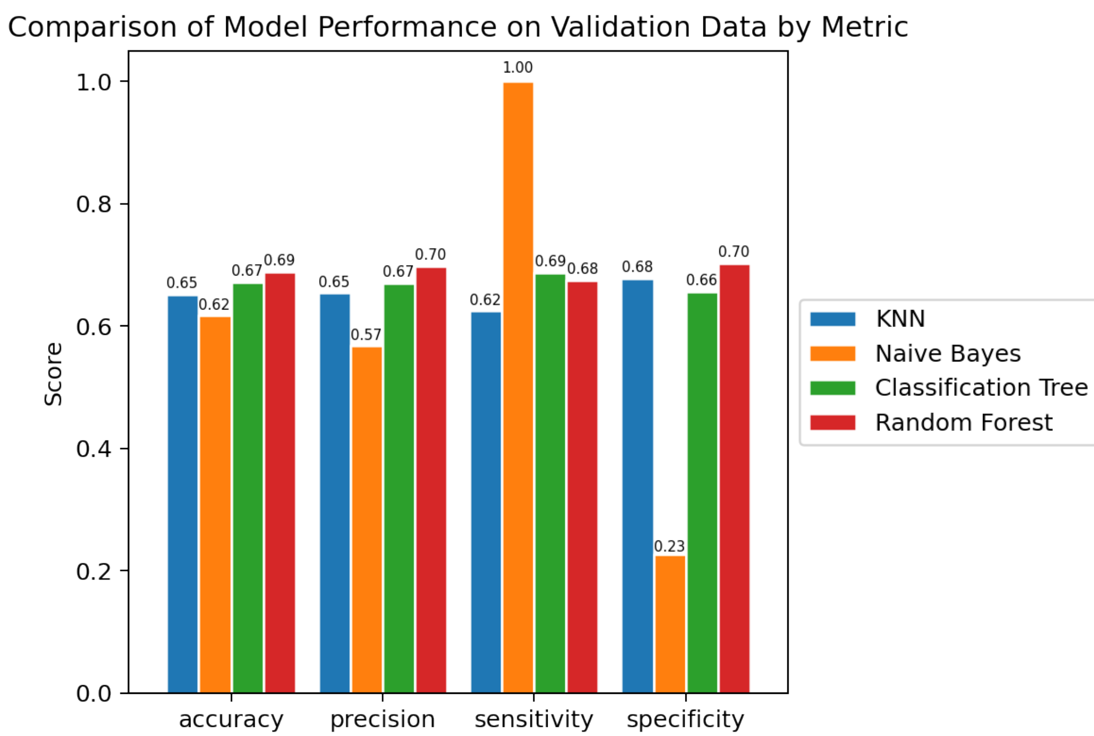


[123]:

```
# x = np.arange(len(metric_df)) # label locations
# width = 0.35

# rects = []

# fig, ax = plt.subplots()

# for row in metric_df.iterrows():
#     rect = ax.bar(x - width/len(metric_df), row[1], width, label=row[0])
#     rects.append(rect)

# # Add some text for labels, title and custom x-axis tick labels, etc.
# ax.set_ylabel('Scores')
# ax.set_title('Comparison of Metrics on Validation Data')
# ax.set_xticks(x)
# ax.set_xticklabels(metric_df.columns)
# ax.legend()
```

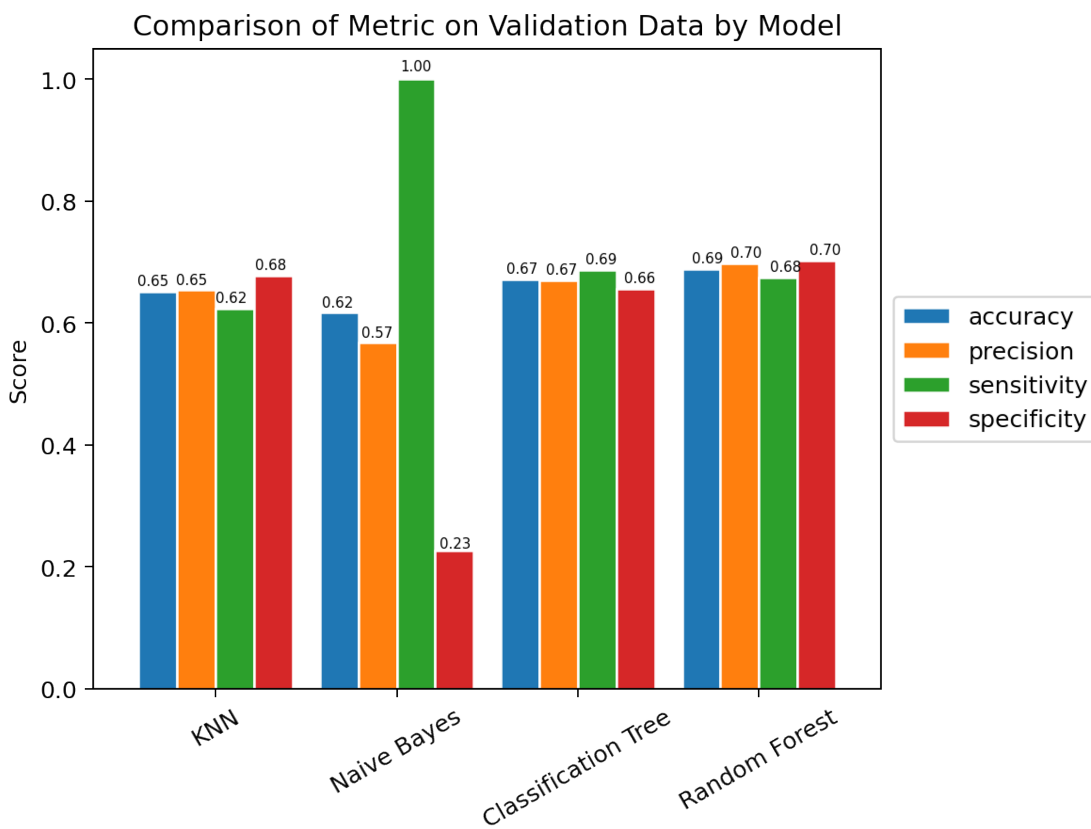
```
# for rect in rects:
#     ax.bar_label(rect, padding=3)

# fig = plt.figure(figsize=(5,5), dpi=150)


fig = plt.figure(figsize=(5,5), dpi=180)

ax = metric_df.transpose().plot.bar(ylim=(0,1.05), ax = plt.gca(), rot=0,␣
↪width=0.85, edgecolor='white')

for p in ax.patches:
    ax.annotate(str(f"{p.get_height():.2f}"), (p.get_x() * 1.001, p.
↪get_height() * 1.015), fontsize=6)

plt.title("Comparison of Model Performance on Validation Data by Metric")
plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
plt.ylabel("Score")
```

[123]: Text(0, 0.5, 'Score')



[124]:
```python
fig = plt.figure(figsize=(6,5), dpi=180)

ax = metric_df.plot.bar(ylim=(0,1.05), ax = plt.gca(), rot=30, width=0.85,␣
 ↪edgecolor='white')

for p in ax.patches:
    ax.annotate(str(f"{p.get_height():.2f}"), (p.get_x() * 1.02, p.get_height()␣
 ↪* 1.015), fontsize=6)

plt.title("Comparison of Metric on Validation Data by Model")
plt.legend(loc="center left", bbox_to_anchor=(1.0, 0.5))
plt.ylabel("Score")
```

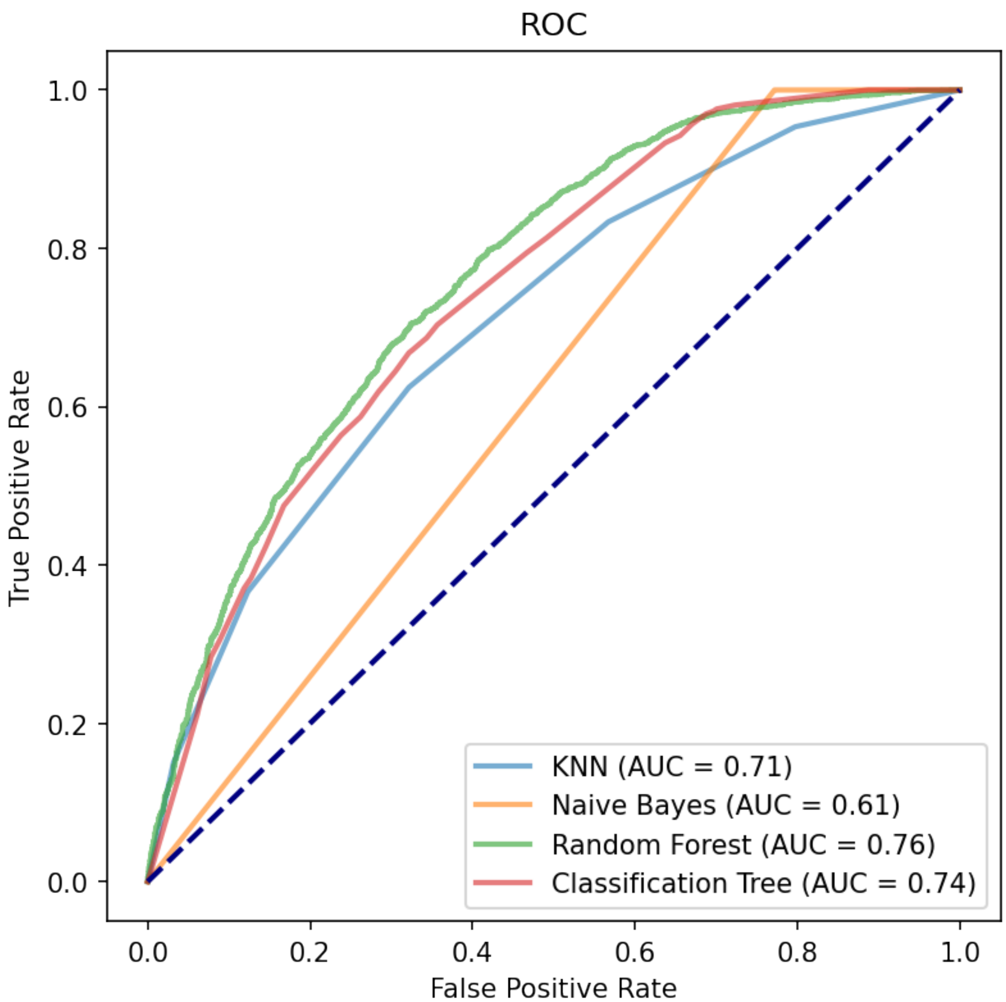
[124]: Text(0, 0.5, 'Score')



1.6.2 ROC Curve

[125]:
```
from sklearn.metrics import accuracy_score, roc_curve, auc
# Create a dictionary of the models and their dataframes.
# The key will be the model. The value will be a list containing their␣
↪validation dataset.

# tree_cv_search_params = Classification Tree
# rf_tree = Random Forest
# TODO: Add in KNN and Bayes

classification_methods = {knn : ["KNN", valid_knn_X, valid_knn_y],
                          gnb : ["Naive Bayes", valid_bayes_X, valid_bayes_y],
                          rf_tree : ["Random Forest", valid_cart_X,␣
↪valid_cart_y],
                          tree_cv_search_params : ["Classification Tree",␣
↪valid_cart_X, valid_cart_y],



                         }



# Try multiple this time.
tprs = [] # true positive rates
aucs = [] # area under curve values
mean_fpr = np.linspace(0,1,100) #linspace returns a list of 100 values evenly␣
↪spaced from 0 - 1.

plt.figure(figsize=(6, 6),dpi=150)

for model, model_df in classification_methods.items():
    prediction = model.predict_proba(model_df[1]) # [1] = validation set␣
↪predictors
    # roc_curve -> 2 args: the actual y values, and the prediction the model␣
↪gave.
    # the prediction is a two column array with probability of being class 0␣
↪and class 1.
    # [:, 1] therefore means all probabilities for being class 1.
    fpr, tpr, t = roc_curve(model_df[2], prediction[:, 1])
    tprs.append(np.interp(mean_fpr, fpr, tpr))
    roc_auc = auc(fpr, tpr)
    aucs.append(roc_auc)
    plt.plot(fpr, tpr, lw=2, alpha=0.6, label='%s (AUC = %0.2f)' %␣
↪(model_df[0], roc_auc))

plt.plot([0,1],[0,1],linestyle = '--',lw = 2,color = 'navy')
plt.xlabel('False Positive Rate')
```

```
plt.ylabel('True Positive Rate')
plt.title('ROC')
plt.legend(loc="lower right")
plt.show()
```



[ ]: