

Project based on

CLASSIFICATION OF BAD ACCOUNTS IN CREDIT CARD INDUSTRY

**Submitted in partial fulfillment of the Requirements for the award of the Degree of Bachelor of Technology**

**In**

**Computer science and Engineering**

By

**NARASIMHA RAO GURRAM**

**TABLE OF CONTENTS**

**CHAPTERS PAGE NO**

ABSTRACT 6

CHAPTER 1: INTRODUCTION

* 1. INTRODUCTION 7
  2. PROBLEM DEFINITION 8
  3. SCOPE 9
  4. PURPOSE 10
  5. PROBLEM AND EXISTING TECHNOLOGY 11
  6. PROPOSED SYSTEM 12

CHAPTER 2: REQIUREMENTS & ANALYSIS

* 1. PLATFORM REQUIREMENTS 13
  2. MODULE DESCRIPTION 14

CHAPTER 3: DESIGN & IMPLEMENTATION

* 1. ALGORITHMS 15 - 17
  2. PSEUDO CODE 18 - 35

CHAPTER 4: SCREENSHOTS 36-44

CHAPTER 5: CONCLUSION 45

CHAPTER 6: REFERENCES 46

**ABSTRACT**

Due to the advanced technology data availability and computing power, most banks or lending institutions are renewing their business models. Credit risk predictions, monitoring, model reliability and effective loan processing are key to decision-making and transparency. In this work, we build binary classifiers based on machine and deep learning models on real data in predicting loan default probability. The top 10 important features from these models are selected and then used in the modeling process to test the stability of binary classifiers by comparing their performance on separate data. We observe that the tree-based models are more stable than the models based on multilayer artificial neural networks. This opens several questions relative to the intensive use of deep learning systems in enterprises.

**CHAPTER 1: INTRODUCTION**

* 1. **INTRODUCTION**

Risk management is critical for a credit card company to survive in such competing industry. In addition to operational expenses, provisional loss is a major driver to a company’s expense. The provisional loss arises due to the “bad” accounts booked – bank lends the money to customers who eventually do not have capability to pay back. In the risk management, there are generally two stages a company can take to manage and control credit risks. The first stage occurs when booking a customer.

An aggressive underwriting strategy could book and approve high risk population who seeks for credit card; while a conservative policy may only focus on upmarket and affluent population. As expected, the first strategy could generate both high revenues (interests charged) and high expenses due to bad accounts booked, resulting in trivial incremental net income; and the second strategy could generate not only low revenues but also low losses, resulting in incremental net income be trivial as well.

There is always trade-off for different strategies in terms revenue generation and loss control. Finding an optimal strategy is often difficult and needs to be adjusted accordingly due to internal or external factors such as macroeconomic change, for example, almost all credit companies suppressed their approval rates for high risk segments of population and incurred huge financial loss during 2008/2009 economic depression.

The second stage happens in customer management after the customer is booked. Although booked customers pass the first screen of risk control, the chance of false negative (false “good” accounts) could still be high. However, in the second stage, by leveraging their performance such as credit card utilization, payment information, risks can further be managed to control provisional loss.

In our project, we will focus on the second stage of risk management, and particularly are interested in classifying if a booked account will be a “bad” account within 12 months since booked. Since an internal classification model is already available, our second interest is to train a better classifier to outperform the benchmark model.

.

**1.2 PROBLEM DEFINITION**

A limited understanding of mortgage contracts and the risks involved may have contributed to the outbreak of the 2007–2008 financial crisis. We developed a special questionnaire relating mortgage loan decisions to financial knowledge and financial advice. Our results demonstrate that homeowners appear to be well aware of mortgage risks. Large loans relative to home value are perceived as riskier, as are loans with large mortgage payments relative to income and loans linked to investment vehicles. Homeowners with riskier mortgages indicated that they could encounter financial problems should house prices or their income decline. Homeowners with relatively low debt literacy are more likely to take out traditional mortgages with principal repayments over the maturity of the loan. Riskier mortgages are more prevalent among homeowners with a better understanding of loan contracts. Financially less sophisticated homeowners consulting mortgage brokers, too, hold riskier mortgages.

Using account-level credit card data from six major commercial banks from January 2009 to December 2013, we apply machine-learning techniques to combined consumer tradeline, credit bureau, and macroeconomic variables to predict delinquency. In addition to providing accurate measures of loss probabilities and credit risk, our models can also be used to analyze and compare risk management practices and the drivers of delinquency across banks. We find substantial heterogeneity in risk factors, sensitivities, and predictability of delinquency across banks, implying that no single model applies to all six institutions. We measure the efficacy of a bank's risk management process by the percentage of delinquent accounts that a bank manages effectively, and find that efficacy also varies widely across institutions. These results suggest the need for a more customized approached to the supervision and regulation of financial institutions, in which capital ratios, loss reserves, and other parameters are specified individually for each institution according to its credit risk model exposures and forecasts.

**1.3 SCOPE**

The original dataset contains 1000 entries with 20 categorial/symbolic attributes prepared by Prof. Hofmann. In this dataset, each entry represents a person who takes a credit by a bank. Each person is classified as good or bad credit risks according to the set of attributes. The link to the original dataset can be found below.

It is almost impossible to understand the original dataset due to its complicated system of categories and symbols. Thus, I wrote a small Python script to convert it into a readable CSV file. Several columns are simply ignored, because in my opinion either they are not important or their descriptions are obscure. The selected attributes are:

Age (numeric)

Sex (text: male, female)

Job (numeric: 0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled)

Housing (text: own, rent, or free)

Saving accounts (text - little, moderate, quite rich, rich)

Checking account (numeric, in DM - Deutsch Mark)

Credit amount (numeric, in DM)

Duration (numeric, in month)

Purpose (text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others)

**1.4 PURPOSE**

By using Machine Learning Algorithm we can classify the bad accounts It has mainly came up after bankrupty after 2009

The gradient boosting method was proposed by Friedman (1999) and was later improved to stochastic gradient boosting by using the bagging procedure. The purpose of boosting methods is to sequentially construct a sequence of weak classifiers and then ensemble them through a weighted majority vote to produce the final prediction.

The random forests method was introduced by Breiman (2001). It is an ensemble method to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much. The de-correlation is achieved via the random selection of variables at nodes when growing trees. Random forests method has several advantages such as its capability of capturing the non-linear boundary between event and non-event, naturally embedded out-of-bag validation, no special treatment to input features, suitable for unbalanced data classification and etc.

The SVM (Cortes and Vapnik, 1995) classifier generally transforms the input attributes into a high dimensional feature space by introducing a mapping (linear or non-linear) via a kernel. When training the classifier, SVM only uses the related support vector points in feature space to find the optimal separating hyperplane. One popular kernel choice is the Gaussian kernel , which represents less parameters than other kernels. For a binary classification problem, the probability output can be generated by



Before fitting the SVM model, the input data is first standardized to a zero mean and one standard deviation. Also, we enable the probability = TRUE in the svm() function of e1071 package to obtain probability type scores

**1.5** **PROBLEM AND EXISTING TECHNOLOGY**

The gradient boosting method was proposed by Friedman (1999) and was later improved to stochastic gradient boosting by using the bagging procedure. The purpose of boosting methods is to sequentially construct a sequence of weak classifiers and then ensemble them through a weighted majority vote to produce the final prediction.

The random forests method was introduced by Breiman (2001). It is an ensemble method to improve the variance reduction of bagging by reducing the correlation between the trees, without increasing the variance too much. The de-correlation is achieved via the random selection of variables at nodes when growing trees. Random forests method has several advantages such as its capability of capturing the non-linear boundary between event and non-event, naturally embedded out-of-bag validation, no special treatment to input features, suitable for unbalanced data classification and etc.

The SVM (Cortes and Vapnik, 1995) classifier generally transforms the input attributes into a high dimensional feature space by introducing a mapping (linear or non-linear) via a kernel. When training the classifier, SVM only uses the related support vector points in feature space to find the optimal separating hyperplane. One popular kernel choice is the Gaussian kernel , which represents less parameters than other kernels. For a binary classification problem, the probability output can be generated by



Before fitting the SVM model, the input data is first standardized to a zero mean and one standard deviation. Also, we enable the probability = TRUE in the svm() function of e1071 package to obtain probability type scores

**1.6 PROPOSED SYSTEM**

Decision trees are powerful models that can be viewed as par- titions of the space X , with a specific prediction of y (either 0 or 1) for each such partition. If the model partitions the space into k mutually exclusive regions , then the model returned by a decision tree can be viewed as f( x ) = ∑ k m =1 c m I[ x ∈ R m ] where c m ∈ {0, 1} and I is an indicator function (see Hastie et al., 2009 ). The partitioning is typically implemented through a series of hier- archical tests, thus the “tree” nomenclature..

A major benefit of the decision tree model as a whole is its interpretability. While the greedy algorithm described above is not guaranteed to find the best model in the space of models it searches, greedy decision tree learners have been very successful in practice because of the combination of speed and reasonably good out-of-sample classification performance that they typically achieve. However, this comes as a tradeoff. The major negative of decision trees as a machine-learning algorithm is that they do not achieve state-of-the-art performance in out-of-sample classifi- cation ( Dietterich, 20 0 0 ; Hastie et al., 20 09 ). Unfortunately, models that do achieve better performance are typically much harder to interpret, a significant negative for the domain of credit risk anal- ysis. In order to determine how much improvement may be possi- ble, we compare the decision tree models with one of these state- of-the art techniques, namely random forests ( Breiman, 2001 ; Breiman and Cutler, 2004 )

**CHAPTER 2: REQIUREMENTS & ANALYSIS**

**2.1 PLATFORM REQUIREMENTS**

* **SOFTWARE REQUIREMENTS:**

The major software requirements of the project are as follows:

Language : Python

Operating system : Windows

* **HARDWARE REQUIREMENTS:**

The hardware requirements that map towards the software are as follows:

RAM : 8 GB

Processor : Intel i5 processor

**2.2 MODULE DESCRIPTION**

**Modules:**

1. **Preprocessing/Training the dataset:**

Importing ML librarys

Setting X and y variables to the prediction

Splitting Data

1. **Random forest predictive model:**

**Random forests** or **random decision forests** are an [ensemble learning](https://en.wikipedia.org/wiki/Ensemble_learning) method for [classification](https://en.wikipedia.org/wiki/Statistical_classification), [regression](https://en.wikipedia.org/wiki/Regression_analysis) and other tasks that operate by constructing a multitude of [decision trees](https://en.wikipedia.org/wiki/Decision_tree_learning) at training time and outputting the class that is the [mode](https://en.wikipedia.org/wiki/Mode_(statistics)) of the classes (classification) or mean prediction (regression) of the individual trees.[[1]](https://en.wikipedia.org/wiki/Random_forest#cite_note-ho1995-1)[[2]](https://en.wikipedia.org/wiki/Random_forest#cite_note-ho1998-2) Random decision forests correct for decision trees' habit of [overfitting](https://en.wikipedia.org/wiki/Overfitting) to their [training set](https://en.wikipedia.org/wiki/Test_set)

1. **Spectogram Generation:**

A spectrogram is a 2D representation of a signal, having time on the x-axis and frequency on the y-axis. A colormap is used to quantify the magnitude of a given frequency within a given time window. In this study, each audio signal was converted into a MEL spectrogram (having MEL frequency bins on the y-axis).

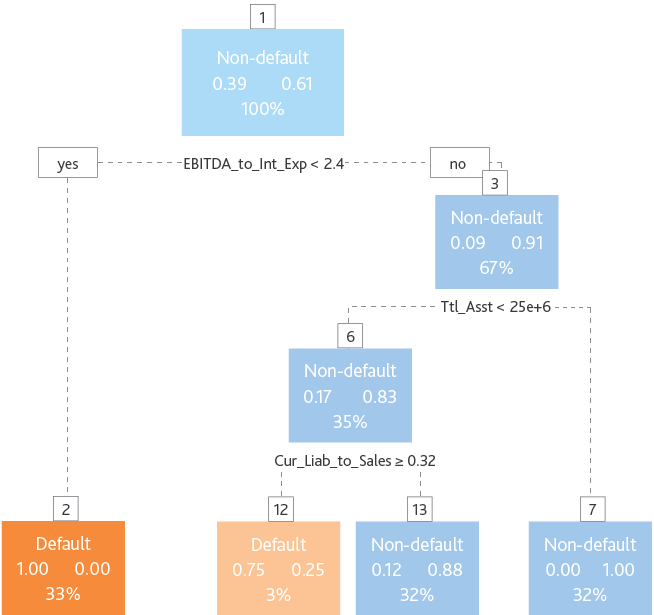
**CHAPTER 3: DESIGN & IMPLEMENTATION**

**3.1 ALGORITHM:**

**Random Forest Classifier**

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an [ensemble](https://en.wikipedia.org/wiki/Ensemble_learning). Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model’s prediction





**Convolution:**

This step involves sliding a matrix ﬁlter (say 3x3 size) over the input image which is of dimension image width x image height. The ﬁlter is ﬁrst placed on the image matrix and then we compute an element-wise multiplication between the ﬁlter and the overlapping portion of the image,followed by a summation to give a feature value. We use many such ﬁlters , the values of which are ’learned’ during the training of the neural network via backpropagation.

**Pooling:**

This is a way to reduce the dimension of the feature map obtained from the convolution step, formally know as the process of down sampling. For example, by max pooling with 2x2 window size, we only retain the element with the maximum value among the 4 elements of the feature map that are covered in this window. We keep moving this window across the feature map with a pre-deﬁned stride.

**Feature Extraction:**

We need to extract meaningful features from audio files. To classify our audio clips, we will choose 5 features, i.e. Mel-Frequency Cepstral Coefficients, Spectral Centroid, Zero Crossing Rate, Chroma Frequencies, Spectral Roll-off. All the features are then appended into a .csv file so that classification algorithms can be used.

**Classifier:**

**Random Forest (RF):** Random Forest is a ensemble learner that combines the prediction from a pre-speciﬁed number of decision trees. It works on the integration of two main principles:

1. Each decision tree is trained with only a subset of the training samples which is known as bootstrap aggregation (or bagging) (Breiman,1996).

2) Each decision tree is required to make its prediction using only a random subset of the features (Amit and Geman,1997). The ﬁnal predicted class of the RF is determined based on the majority vote from the individual classiﬁers.

* 1. **PSEUDO CODE**

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

df\_credit = pd.read\_csv("../input/german-credit-data-with-risk/german\_credit\_data.csv",index\_col=0)

print(df\_credit.info())

print(df\_credit.nunique())

print(df\_credit.head())

import plotly.offline as py

py.init\_notebook\_mode(connected=True)

import plotly.graph\_objs as go

import plotly.tools as tls

import warnings

from collections import Counter

trace0 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'good']["Risk"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'good']["Risk"].value\_counts().values,

name='Good credit'

)

trace1 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'bad']["Risk"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'bad']["Risk"].value\_counts().values,

name='Bad credit'

)

data = [trace0, trace1]

layout = go.Layout(

)

layout = go.Layout(

yaxis=dict(

title='Count'

),

xaxis=dict(

title='Risk Variable'

),

title='Target variable distribution'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='grouped-bar')

df\_good = df\_credit.loc[df\_credit["Risk"] == 'good']['Age'].values.tolist()

df\_bad = df\_credit.loc[df\_credit["Risk"] == 'bad']['Age'].values.tolist()

df\_age = df\_credit['Age'].values.tolist()

#First plot

trace0 = go.Histogram(

x=df\_good,

histnorm='probability',

name="Good Credit"

)

#Second plot

trace1 = go.Histogram(

x=df\_bad,

histnorm='probability',

name="Bad Credit"

)

#Third plot

trace2 = go.Histogram(

x=df\_age,

histnorm='probability',

name="Overall Age"

)

#Creating the grid

fig = tls.make\_subplots(rows=2, cols=2, specs=[[{}, {}], [{'colspan': 2}, None]],

subplot\_titles=('Good','Bad', 'General Distribuition'))

#setting the figs

fig.append\_trace(trace0, 1, 1)

fig.append\_trace(trace1, 1, 2)

fig.append\_trace(trace2, 2, 1)

fig['layout'].update(showlegend=True, title='Age Distribuition', bargap=0.05)

py.iplot(fig, filename='custom-sized-subplot-with-subplot-titles')

df\_good = df\_credit[df\_credit["Risk"] == 'good']

df\_bad = df\_credit[df\_credit["Risk"] == 'bad']

fig, ax = plt.subplots(nrows=2, figsize=(12,8))

plt.subplots\_adjust(hspace = 0.4, top = 0.8)

g1 = sns.distplot(df\_good["Age"], ax=ax[0],

color="g")

g1 = sns.distplot(df\_bad["Age"], ax=ax[0],

color='r')

g1.set\_title("Age Distribuition", fontsize=15)

g1.set\_xlabel("Age")

g1.set\_xlabel("Frequency")

g2 = sns.countplot(x="Age",data=df\_credit,

palette="hls", ax=ax[1],

hue = "Risk")

g2.set\_title("Age Counting by Risk", fontsize=15)

g2.set\_xlabel("Age")

g2.set\_xlabel("Count")

plt.show()

interval = (18, 25, 35, 60, 120)

cats = ['Student', 'Young', 'Adult', 'Senior']

df\_credit["Age\_cat"] = pd.cut(df\_credit.Age, interval, labels=cats)

df\_good = df\_credit[df\_credit["Risk"] == 'good']

df\_bad = df\_credit[df\_credit["Risk"] == 'bad']

trace0 = go.Box(

y=df\_good["Credit amount"],

x=df\_good["Age\_cat"],

name='Good credit',

marker=dict(

color='#3D9970'

)

)

trace1 = go.Box(

y=df\_bad['Credit amount'],

x=df\_bad['Age\_cat'],

name='Bad credit',

marker=dict(

color='#FF4136'

)

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title='Credit Amount (US Dollar)',

zeroline=False

),

xaxis=dict(

title='Age Categorical'

),

boxmode='group'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='box-age-cat')

#First plot

trace0 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'good']["Housing"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'good']["Housing"].value\_counts().values,

name='Good credit'

)

#Second plot

trace1 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'bad']["Housing"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'bad']["Housing"].value\_counts().values,

name="Bad Credit"

)

data = [trace0, trace1]

layout = go.Layout(

title='Housing Distribuition'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='Housing-Grouped')

fig = {

"data": [

{

"type": 'violin',

"x": df\_good['Housing'],

"y": df\_good['Credit amount'],

"legendgroup": 'Good Credit',

"scalegroup": 'No',

"name": 'Good Credit',

"side": 'negative',

"box": {

"visible": True

},

"meanline": {

"visible": True

},

"line": {

"color": 'blue'

}

},

{

"type": 'violin',

"x": df\_bad['Housing'],

"y": df\_bad['Credit amount'],

"legendgroup": 'Bad Credit',

"scalegroup": 'No',

"name": 'Bad Credit',

"side": 'positive',

"box": {

"visible": True

},

"meanline": {

"visible": True

},

"line": {

"color": 'green'

}

}

],

"layout" : {

"yaxis": {

"zeroline": False,

},

"violingap": 0,

"violinmode": "overlay"

}

}

py.iplot(fig, filename = 'violin/split', validate = False)

#First plot

trace0 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'good']["Sex"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'good']["Sex"].value\_counts().values,

name='Good credit'

)

#First plot 2

trace1 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'bad']["Sex"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'bad']["Sex"].value\_counts().values,

name="Bad Credit"

)

#Second plot

trace2 = go.Box(

x = df\_credit[df\_credit["Risk"]== 'good']["Sex"],

y = df\_credit[df\_credit["Risk"]== 'good']["Credit amount"],

name=trace0.name

)

#Second plot 2

trace3 = go.Box(

x = df\_credit[df\_credit["Risk"]== 'bad']["Sex"],

y = df\_credit[df\_credit["Risk"]== 'bad']["Credit amount"],

name=trace1.name

)

data = [trace0, trace1, trace2,trace3]

fig = tls.make\_subplots(rows=1, cols=2,

subplot\_titles=('Sex Count', 'Credit Amount by Sex'))

fig.append\_trace(trace0, 1, 1)

fig.append\_trace(trace1, 1, 1)

fig.append\_trace(trace2, 1, 2)

fig.append\_trace(trace3, 1, 2)

fig['layout'].update(height=400, width=800, title='Sex Distribuition', boxmode='group')

py.iplot(fig, filename='sex-subplot')

#First plot

trace0 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'good']["Job"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'good']["Job"].value\_counts().values,

name='Good credit Distribuition'

)

#Second plot

trace1 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'bad']["Job"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'bad']["Job"].value\_counts().values,

name="Bad Credit Distribuition"

)

data = [trace0, trace1]

layout = go.Layout(

title='Job Distribuition'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='grouped-bar')

trace0 = go.Box(

x=df\_good["Job"],

y=df\_good["Credit amount"],

name='Good credit'

)

trace1 = go.Box(

x=df\_bad['Job'],

y=df\_bad['Credit amount'],

name='Bad credit'

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title='Credit Amount distribuition by Job'

),

boxmode='group'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='box-age-cat')

fig = {

"data": [

{

"type": 'violin',

"x": df\_good['Job'],

"y": df\_good['Age'],

"legendgroup": 'Good Credit',

"scalegroup": 'No',

"name": 'Good Credit',

"side": 'negative',

"box": {

"visible": True

},

"meanline": {

"visible": True

},

"line": {

"color": 'blue'

}

},

{

"type": 'violin',

"x": df\_bad['Job'],

"y": df\_bad['Age'],

"legendgroup": 'Bad Credit',

"scalegroup": 'No',

"name": 'Bad Credit',

"side": 'positive',

"box": {

"visible": True

},

"meanline": {

"visible": True

},

"line": {

"color": 'green'

}

}

],

"layout" : {

"yaxis": {

"zeroline": False,

},

"violingap": 0,

"violinmode": "overlay"

}

}

py.iplot(fig, filename = 'Age-Housing', validate = False)

fig, ax = plt.subplots(figsize=(12,12), nrows=2)

g1 = sns.boxplot(x="Job", y="Credit amount", data=df\_credit,

palette="hls", ax=ax[0], hue="Risk")

g1.set\_title("Credit Amount by Job", fontsize=15)

g1.set\_xlabel("Job Reference", fontsize=12)

g1.set\_ylabel("Credit Amount", fontsize=12)

g2 = sns.violinplot(x="Job", y="Age", data=df\_credit, ax=ax[1],

hue="Risk", split=True, palette="hls")

g2.set\_title("Job Type reference x Age", fontsize=15)

g2.set\_xlabel("Job Reference", fontsize=12)

g2.set\_ylabel("Age", fontsize=12)

plt.subplots\_adjust(hspace = 0.4,top = 0.9)

plt.show()

import plotly.figure\_factory as ff

import numpy as np

# Add histogram data

x1 = np.log(df\_good['Credit amount'])

x2 = np.log(df\_bad["Credit amount"])

# Group data together

hist\_data = [x1, x2]

group\_labels = ['Good Credit', 'Bad Credit']

# Create distplot with custom bin\_size

fig = ff.create\_distplot(hist\_data, group\_labels, bin\_size=.2)

# Plot!

py.iplot(fig, filename='Distplot with Multiple Datasets')

#Ploting the good and bad dataframes in distplot

plt.figure(figsize = (8,5))

g= sns.distplot(df\_good['Credit amount'], color='r')

g = sns.distplot(df\_bad["Credit amount"], color='g')

g.set\_title("Credit Amount Frequency distribuition", fontsize=15)

plt.show()

from plotly import tools

import numpy as np

import plotly.graph\_objs as go

count\_good = go.Bar(

x = df\_good["Saving accounts"].value\_counts().index.values,

y = df\_good["Saving accounts"].value\_counts().values,

name='Good credit'

)

count\_bad = go.Bar(

x = df\_bad["Saving accounts"].value\_counts().index.values,

y = df\_bad["Saving accounts"].value\_counts().values,

name='Bad credit'

)

box\_1 = go.Box(

x=df\_good["Saving accounts"],

y=df\_good["Credit amount"],

name='Good credit'

)

box\_2 = go.Box(

x=df\_bad["Saving accounts"],

y=df\_bad["Credit amount"],

name='Bad credit'

)

scat\_1 = go.Box(

x=df\_good["Saving accounts"],

y=df\_good["Age"],

name='Good credit'

)

scat\_2 = go.Box(

x=df\_bad["Saving accounts"],

y=df\_bad["Age"],

name='Bad credit'

)

data = [scat\_1, scat\_2, box\_1, box\_2, count\_good, count\_bad]

fig = tools.make\_subplots(rows=2, cols=2, specs=[[{}, {}], [{'colspan': 2}, None]],

subplot\_titles=('Count Saving Accounts','Credit Amount by Savings Acc',

'Age by Saving accounts'))

fig.append\_trace(count\_good, 1, 1)

fig.append\_trace(count\_bad, 1, 1)

fig.append\_trace(box\_2, 1, 2)

fig.append\_trace(box\_1, 1, 2)

fig.append\_trace(scat\_1, 2, 1)

fig.append\_trace(scat\_2, 2, 1)

fig['layout'].update(height=700, width=800, title='Saving Accounts Exploration', boxmode='group')

py.iplot(fig, filename='combined-savings')

print("Description of Distribuition Saving accounts by Risk: ")

print(pd.crosstab(df\_credit["Saving accounts"],df\_credit.Risk))

fig, ax = plt.subplots(3,1, figsize=(12,12))

g = sns.countplot(x="Saving accounts", data=df\_credit, palette="hls",

ax=ax[0],hue="Risk")

g.set\_title("Saving Accounts Count", fontsize=15)

g.set\_xlabel("Saving Accounts type", fontsize=12)

g.set\_ylabel("Count", fontsize=12)

g1 = sns.violinplot(x="Saving accounts", y="Job", data=df\_credit, palette="hls",

hue = "Risk", ax=ax[1],split=True)

g1.set\_title("Saving Accounts by Job", fontsize=15)

g1.set\_xlabel("Savings Accounts type", fontsize=12)

g1.set\_ylabel("Job", fontsize=12)

g = sns.boxplot(x="Saving accounts", y="Credit amount", data=df\_credit, ax=ax[2],

hue = "Risk",palette="hls")

g2.set\_title("Saving Accounts by Credit Amount", fontsize=15)

g2.set\_xlabel("Savings Accounts type", fontsize=12)

g2.set\_ylabel("Credit Amount(US)", fontsize=12)

plt.subplots\_adjust(hspace = 0.4,top = 0.9)

plt.show()

print("Values describe: ")

print(pd.crosstab(df\_credit.Purpose, df\_credit.Risk))

plt.figure(figsize = (14,12))

plt.subplot(221)

g = sns.countplot(x="Purpose", data=df\_credit,

palette="hls", hue = "Risk")

g.set\_xticklabels(g.get\_xticklabels(),rotation=45)

g.set\_xlabel("", fontsize=12)

g.set\_ylabel("Count", fontsize=12)

g.set\_title("Purposes Count", fontsize=20)

plt.subplot(222)

g1 = sns.violinplot(x="Purpose", y="Age", data=df\_credit,

palette="hls", hue = "Risk",split=True)

g1.set\_xticklabels(g1.get\_xticklabels(),rotation=45)

g1.set\_xlabel("", fontsize=12)

g1.set\_ylabel("Count", fontsize=12)

g1.set\_title("Purposes by Age", fontsize=20)

plt.subplot(212)

g2 = sns.boxplot(x="Purpose", y="Credit amount", data=df\_credit,

palette="hls", hue = "Risk")

g2.set\_xlabel("Purposes", fontsize=12)

g2.set\_ylabel("Credit Amount", fontsize=12)

g2.set\_title("Credit Amount distribuition by Purposes", fontsize=20)

plt.subplots\_adjust(hspace = 0.6, top = 0.8)

plt.show()

plt.figure(figsize = (12,14))

g= plt.subplot(311)

g = sns.countplot(x="Duration", data=df\_credit,

palette="hls", hue = "Risk")

g.set\_xlabel("Duration Distribuition", fontsize=12)

g.set\_ylabel("Count", fontsize=12)

g.set\_title("Duration Count", fontsize=20)

g1 = plt.subplot(312)

g1 = sns.pointplot(x="Duration", y ="Credit amount",data=df\_credit,

hue="Risk", palette="hls")

g1.set\_xlabel("Duration", fontsize=12)

g1.set\_ylabel("Credit Amount(US)", fontsize=12)

g1.set\_title("Credit Amount distribuition by Duration", fontsize=20)

g2 = plt.subplot(313)

g2 = sns.distplot(df\_good["Duration"], color='g')

g2 = sns.distplot(df\_bad["Duration"], color='r')

g2.set\_xlabel("Duration", fontsize=12)

g2.set\_ylabel("Frequency", fontsize=12)

g2.set\_title("Duration Frequency x good and bad Credit", fontsize=20)

plt.subplots\_adjust(wspace = 0.4, hspace = 0.4,top = 0.9)

plt.show()

#First plot

trace0 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'good']["Checking account"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'good']["Checking account"].value\_counts().values,

name='Good credit Distribuition'

)

#Second plot

trace1 = go.Bar(

x = df\_credit[df\_credit["Risk"]== 'bad']["Checking account"].value\_counts().index.values,

y = df\_credit[df\_credit["Risk"]== 'bad']["Checking account"].value\_counts().values,

name="Bad Credit Distribuition"

)

data = [trace0, trace1]

layout = go.Layout(

title='Checking accounts Distribuition',

xaxis=dict(title='Checking accounts name'),

yaxis=dict(title='Count'),

barmode='group'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename = 'Age-ba', validate = False)

df\_good = df\_credit[df\_credit["Risk"] == 'good']

df\_bad = df\_credit[df\_credit["Risk"] == 'bad']

trace0 = go.Box(

y=df\_good["Credit amount"],

x=df\_good["Checking account"],

name='Good credit',

marker=dict(

color='#3D9970'

)

)

trace1 = go.Box(

y=df\_bad['Credit amount'],

x=df\_bad['Checking account'],

name='Bad credit',

marker=dict(

color='#FF4136'

)

)

data = [trace0, trace1]

layout = go.Layout(

yaxis=dict(

title='Cheking distribuition'

),

boxmode='group'

)

fig = go.Figure(data=data, layout=layout)

py.iplot(fig, filename='box-age-cat')

print("Total values of the most missing variable: ")

print(df\_credit.groupby("Checking account")["Checking account"].count())

plt.figure(figsize = (12,10))

g = plt.subplot(221)

g = sns.countplot(x="Checking account", data=df\_credit,

palette="hls", hue="Risk")

g.set\_xlabel("Checking Account", fontsize=12)

g.set\_ylabel("Count", fontsize=12)

g.set\_title("Checking Account Counting by Risk", fontsize=20)

g1 = plt.subplot(222)

g1 = sns.violinplot(x="Checking account", y="Age", data=df\_credit, palette="hls", hue = "Risk",split=True)

g1.set\_xlabel("Checking Account", fontsize=12)

g1.set\_ylabel("Age", fontsize=12)

g1.set\_title("Age by Checking Account", fontsize=20)

g2 = plt.subplot(212)

g2 = sns.boxplot(x="Checking account",y="Credit amount", data=df\_credit,hue='Risk',palette="hls")

g2.set\_xlabel("Checking Account", fontsize=12)

g2.set\_ylabel("Credit Amount(US)", fontsize=12)

g2.set\_title("Credit Amount by Cheking Account", fontsize=20)

plt.subplots\_adjust(wspace = 0.2, hspace = 0.3, top = 0.9)

plt.show()

plt.show()

print(pd.crosstab(df\_credit.Sex, df\_credit.Job))

plt.figure(figsize = (10,6))

g = sns.violinplot(x="Housing",y="Job",data=df\_credit,

hue="Risk", palette="hls",split=True)

g.set\_xlabel("Housing", fontsize=12)

g.set\_ylabel("Job", fontsize=12)

g.set\_title("Housing x Job - Dist", fontsize=20)

plt.show()

print(pd.crosstab(df\_credit["Checking account"],df\_credit.Sex))

date\_int = ["Purpose", 'Sex']

cm = sns.light\_palette("green", as\_cmap=True)

pd.crosstab(df\_credit[date\_int[0]], df\_credit[date\_int[1]]).style.background\_gradient(cmap = cm)

date\_int = ["Purpose", 'Sex']

cm = sns.light\_palette("green", as\_cmap=True)

pd.crosstab(df\_credit[date\_int[0]], df\_credit[date\_int[1]]).style.background\_gradient(cmap = cm)

print("Purpose : ",df\_credit.Purpose.unique())

print("Sex : ",df\_credit.Sex.unique())

print("Housing : ",df\_credit.Housing.unique())

print("Saving accounts : ",df\_credit['Saving accounts'].unique())

print("Risk : ",df\_credit['Risk'].unique())

print("Checking account : ",df\_credit['Checking account'].unique())

print("Aget\_cat : ",df\_credit['Age\_cat'].unique())

def one\_hot\_encoder(df, nan\_as\_category = False):

original\_columns = list(df.columns)

categorical\_columns = [col for col in df.columns if df[col].dtype == 'object']

df = pd.get\_dummies(df, columns= categorical\_columns, dummy\_na= nan\_as\_category, drop\_first=True)

new\_columns = [c for c in df.columns if c not in original\_columns]

return df, new\_columns

df\_credit['Saving accounts'] = df\_credit['Saving accounts'].fillna('no\_inf')

df\_credit['Checking account'] = df\_credit['Checking account'].fillna('no\_inf')

#Purpose to Dummies Variable

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit.Purpose, drop\_first=True, prefix='Purpose'), left\_index=True, right\_index=True)

#Sex feature in dummies

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit.Sex, drop\_first=True, prefix='Sex'), left\_index=True, right\_index=True)

# Housing get dummies

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit.Housing, drop\_first=True, prefix='Housing'), left\_index=True, right\_index=True)

# Housing get Saving Accounts

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit["Saving accounts"], drop\_first=True, prefix='Savings'), left\_index=True, right\_index=True)

# Housing get Risk

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit.Risk, prefix='Risk'), left\_index=True, right\_index=True)

# Housing get Checking Account

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit["Checking account"], drop\_first=True, prefix='Check'), left\_index=True, right\_index=True)

# Housing get Age categorical

df\_credit = df\_credit.merge(pd.get\_dummies(df\_credit["Age\_cat"], drop\_first=True, prefix='Age\_cat'), left\_index=True, right\_index=True)

#Excluding the missing columns

del df\_credit["Saving accounts"]

del df\_credit["Checking account"]

del df\_credit["Purpose"]

del df\_credit["Sex"]

del df\_credit["Housing"]

del df\_credit["Age\_cat"]

del df\_credit["Risk"]

del df\_credit['Risk\_good']

plt.figure(figsize=(14,12))

sns.heatmap(df\_credit.astype(float).corr(),linewidths=0.1,vmax=1.0,

square=True, linecolor='white', annot=True)

plt.show()

from sklearn.model\_selection import train\_test\_split, KFold, cross\_val\_score # to split the data

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, fbeta\_score #To evaluate our model

from sklearn.model\_selection import GridSearchCV

# Algorithmns models to be compared

from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.tree import DecisionTreeClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis

from sklearn.naive\_bayes import GaussianNB

from sklearn.svm import SVC

from xgboost import XGBClassifier

df\_credit['Credit amount'] = np.log(df\_credit['Credit amount'])

# to feed the random state

seed = 7

# prepare models

models = []

models.append(('LR', LogisticRegression()))

models.append(('LDA', LinearDiscriminantAnalysis()))

models.append(('KNN', KNeighborsClassifier()))

models.append(('CART', DecisionTreeClassifier()))

models.append(('NB', GaussianNB()))

models.append(('RF', RandomForestClassifier()))

models.append(('SVM', SVC(gamma='auto')))

models.append(('XGB', XGBClassifier()))

# evaluate each model in turn

results = []

names = []

scoring = 'recall'

for name, model in models:

kfold = KFold(n\_splits=10, random\_state=seed)

cv\_results = cross\_val\_score(model, X\_train, y\_train, cv=kfold, scoring=scoring)

results.append(cv\_results)

names.append(name)

msg = "%s: %f (%f)" % (name, cv\_results.mean(), cv\_results.std())

print(msg)

# boxplot algorithm comparison

fig = plt.figure(figsize=(11,6))

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(results)

ax.set\_xticklabels(names)

plt.show()

#Seting the Hyper Parameters

param\_grid = {"max\_depth": [3,5, 7, 10,None],

"n\_estimators":[3,5,10,25,50,150],

"max\_features": [4,7,15,20]}

#Creating the classifier

model = RandomForestClassifier(random\_state=2)

grid\_search = GridSearchCV(model, param\_grid=param\_grid, cv=5, scoring='recall', verbose=4)

grid\_search.fit(X\_train, y\_train)

print(grid\_search.best\_score\_)

print(grid\_search.best\_params\_)

rf = RandomForestClassifier(max\_depth=None, max\_features=10, n\_estimators=15, random\_state=2)

#trainning with the best params

rf.fit(X\_train, y\_train)

#Testing the model

#Predicting using our model

y\_pred = rf.predict(X\_test)

# Verificaar os resultados obtidos

print(accuracy\_score(y\_test,y\_pred)\*100)

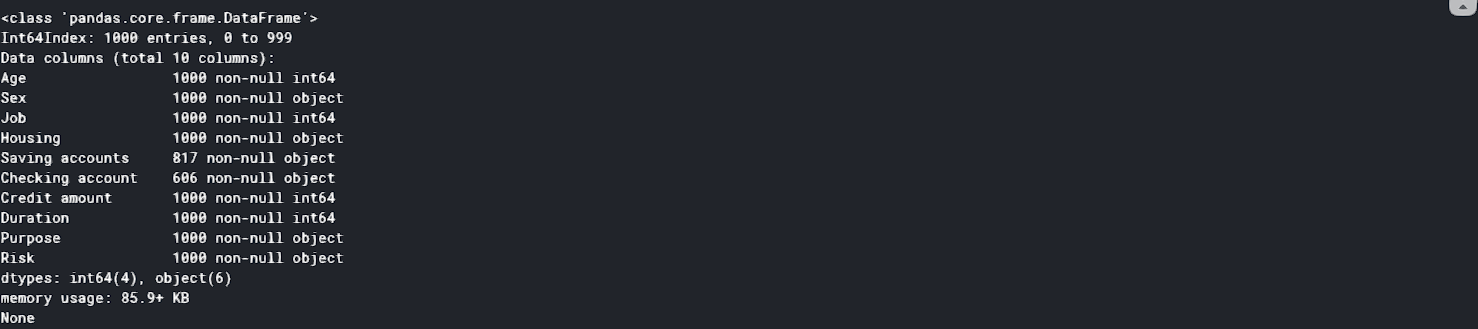
print("\n")

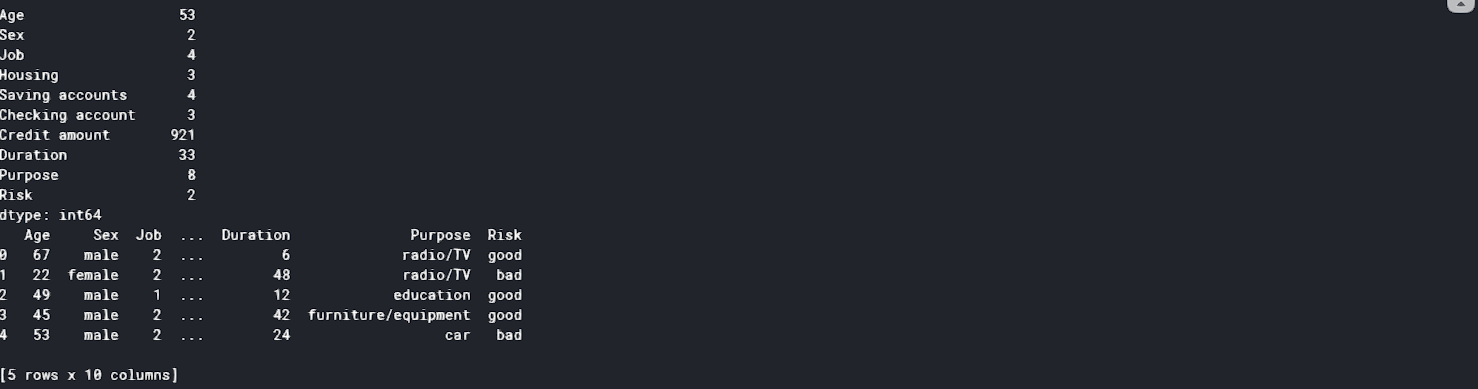
print(confusion\_matrix(y\_test, y\_pred))

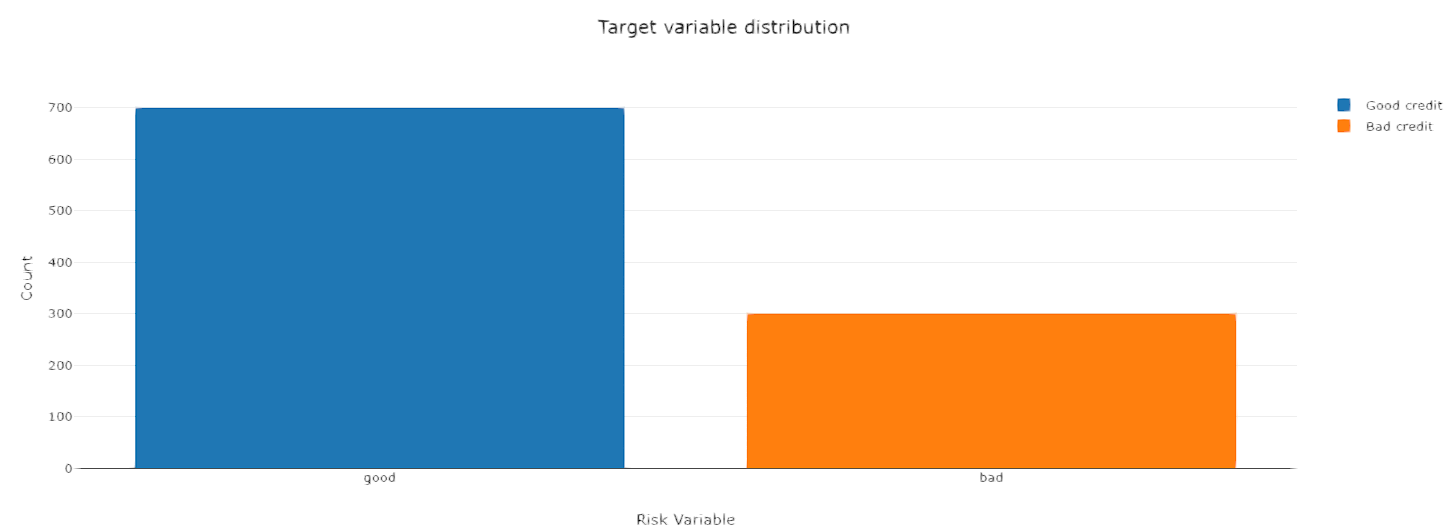
print("\n")

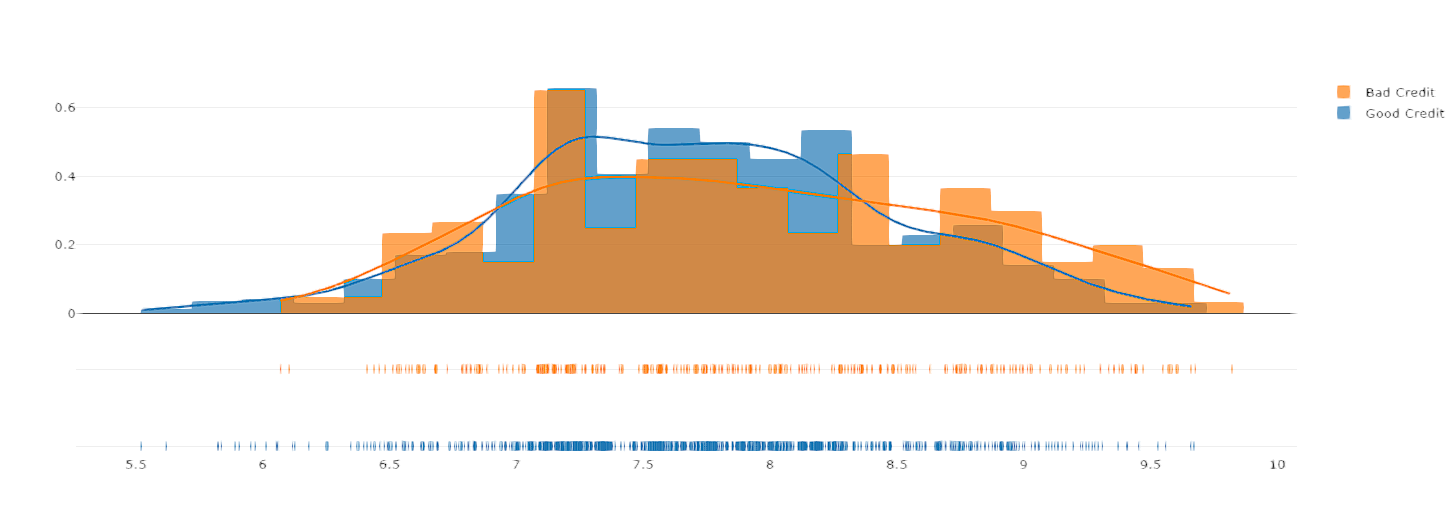
print(fbeta\_score(y\_test, y\_pred, beta=2))

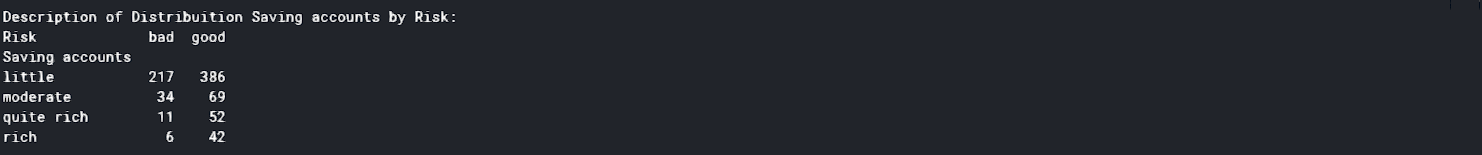
**CHAPTER 4: SCREENSHOTS**

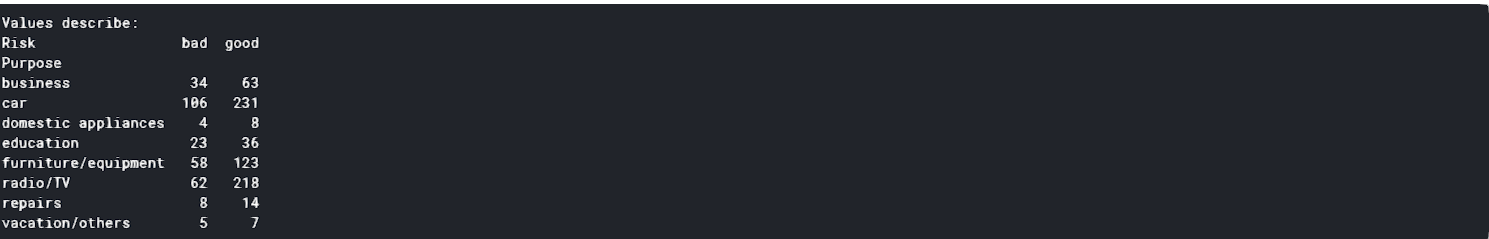


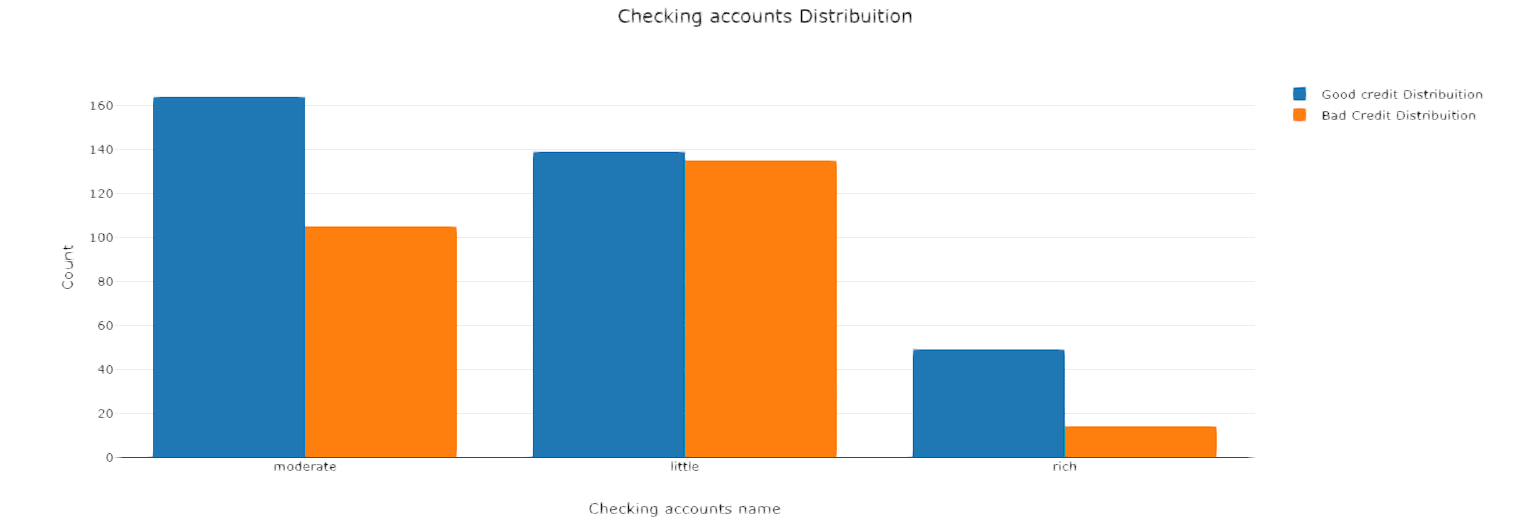


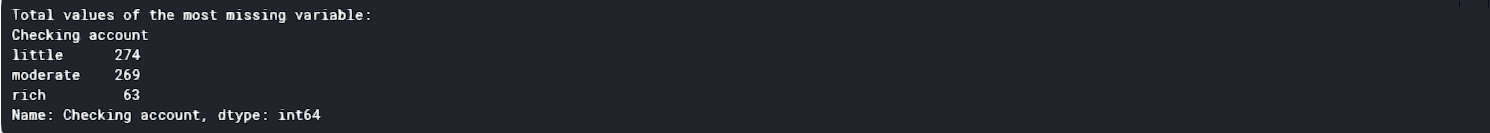




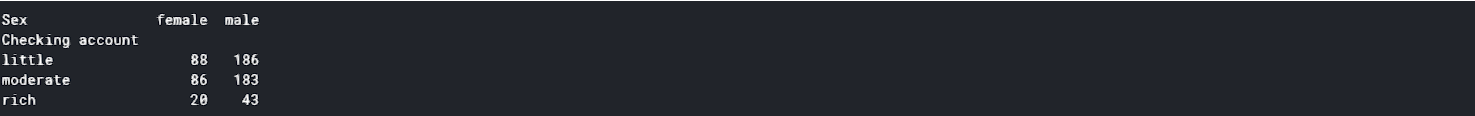


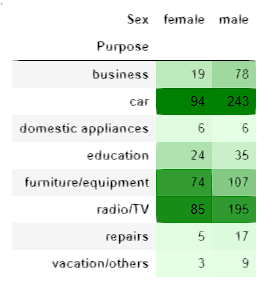


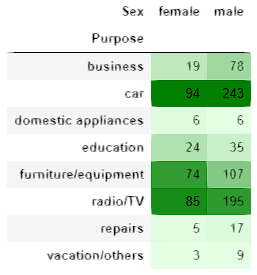




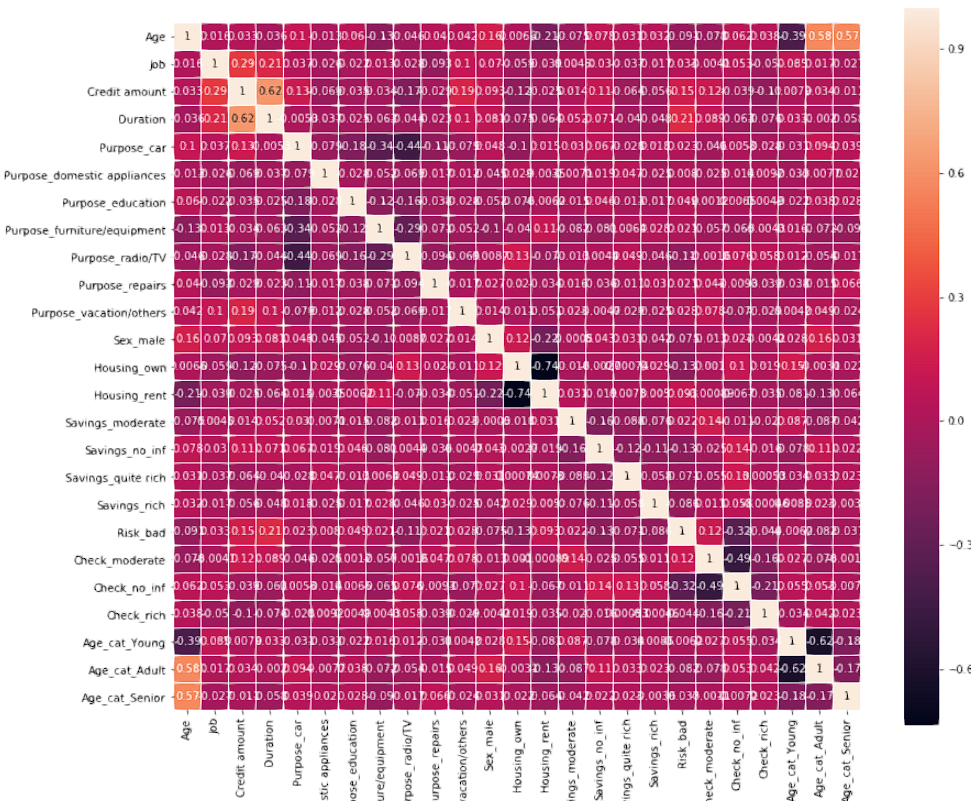


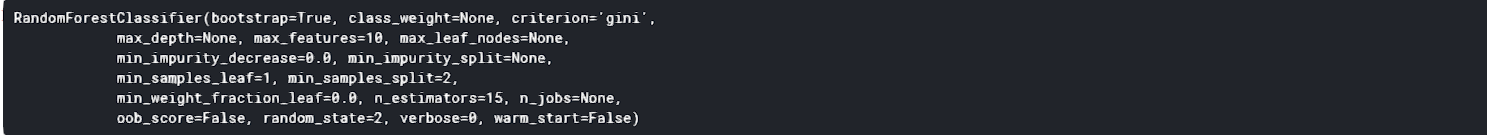


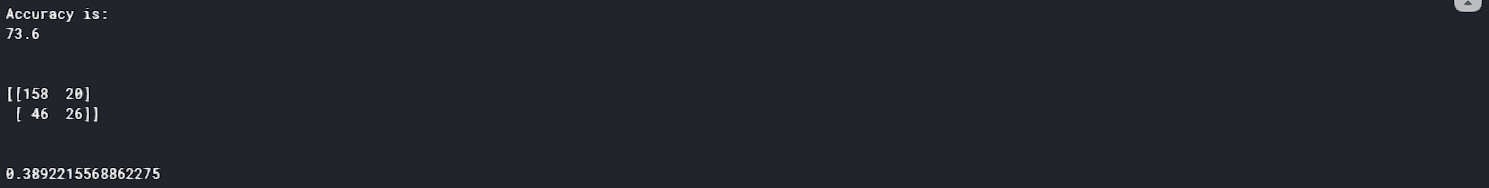






****

****

****

**CHAPTER 5: CONCLUSION**

machine learning and deep learning models, does have a significant role in credit risk modeling. In this exercise, we have showed that it is important to make checks on data quality (in the preparation and cleaning process to omit redundant variables), and it is important to deal with an imbalanced training dataset to avoid bias to a majority class

We have also indicated that the choice of the features to respond to a business objective (In our case, should a loan be awarded to an enterprise? Can we use few variables to save time in this decision making?) and the choice of the algorithm used to make the decision (whether the enterprise makes defaults) are two important keys in the decision management processing when issuing a loan (here, the bank). These decisions can have an important impact on the real economy or the social world; thus, regular and frequent training of employees in this new field is crucial to be able to adapt to and properly use these new techniques. As such, it is important that regulators and policy-makers make quick decisions to regulate the use of data science techniques to boost performance, avoid discrimination in terms of wrong decisions proposed by algorithms and to understand the limits of some of these algorithms.

.

**CHAPTER 6: REFERENCES**

* <https://blog.exxeta.com/en/2019/12/10/credit-risk-analysis-using-machine-learning/>
* <https://www.kaggle.com/vigneshj6/german-credit-data-analysis-python>
* <https://blog.exxeta.com/en/2019/12/10/credit-risk-analysis-using-machine-learning/>