Python 语言

Gangmin Li

  Cambridge, UK 2022.02

**金融风险分析**



操作手册 3

Credit Risk Prediction信贷风险预估简介

这是三部分的实践操作手册的最后一部分。这个部分主要练习数据分析的主要步骤：

1. 问题理解
2. 原始数据获取和理解
3. 数据预处理
4. 模型选择和训练
5. 模型评估和精调
6. 测试和报告

为了演示这些基本步骤我们采用金融分析分析的主要应用之一—信贷风险评估。 通过获得历史数据使用大数据分析方法来实现我们风险评估的目的。本手册不在按照任务和工具分成多个独立的文件，而是按照实际报告形式汇集在一个大的完整报告之中。

你可以一边阅读此手册，一边阅读和运行Jupyter notebook 的内容。

**How to get the most out of this course?**

To get the most out of this tutorial you should do the following.

Download the Jupyter Notebook used (link below).

[GangminLi/Financial-Risk-Management-with-Bank-Loan: The third part of Financial Risk course (github.com)](https://github.com/GangminLi/Financial-Risk-Management-with-Bank-Loan)

Start your Jupyter Notebook. Clear all the results of the code (click cells -> clear all)

Try it yourself in Jupyter Notebook (add your interpretation or understanding of each cell of code. Some of them have been done for you).

If you don’t have or cannot install Jupyter Notebook. Alternatively, you can use some online Python running environment such as:

* **Colab** (<https://colab.research.google.com)> <https://colab.research.google.com/drive/1S7V0qHnmMGEglGEHHT_nl-vt9xnJaYQ-#scrollTo=85f91c09>
* **Mybinder** [**https://mybinder.org/v2/gh/GangminLi/Basics-Financial-Risk-Management/HEAD**](https://mybinder.org/v2/gh/GangminLi/Basics-Financial-Risk-Management/HEAD)
* **Kaggle code** [**https://www.kaggle.com/garygli/notebook62a25ca0ae/edit**](https://www.kaggle.com/garygli/notebook62a25ca0ae/edit)

# Tables of Content

Table of Contents

[1. Understand the Problem 4](#_Toc127394220)

[What is the Purpose of Credit Risk Analysis? 4](#_Toc127394221)

[What is the result of Credit risk analysis? 4](#_Toc127394222)

[What tools can be used for Credit risk analysis? 4](#_Toc127394223)

[The data set 5](#_Toc127394224)

[German-Credit-Dataset 5](#_Toc127394225)

[Content 5](#_Toc127394226)

[2. Understanding of Data 6](#_Toc127394227)

[2.1 Understanding of data with DDA 6](#_Toc127394228)

[Import raw data 6](#_Toc127394229)

[General describe of data 7](#_Toc127394230)

[Check attributes value uniqueness 8](#_Toc127394231)

[Five-number summary 9](#_Toc127394232)

[Data quality assessment 9](#_Toc127394233)

[Finding missing values and the percentage 9](#_Toc127394234)

[3. Data Attribute Analysis with EDA 11](#_Toc127394235)

[3.1 Univariate analysis 11](#_Toc127394236)

[Attribute with Object type and their distribuition analyses 11](#_Toc127394237)

[Attribute with numerical type and their distribution analyses 13](#_Toc127394238)

[3.2. Bivariate analysis 14](#_Toc127394239)

[Pairwise attributes plot with Seaborn 15](#_Toc127394240)

[Check numerical variables with Risk by histogram 16](#_Toc127394241)

[Check categorical variables against Risk 17](#_Toc127394242)

**1.**[**Understanding the problem**](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#Problem)

* The problem
* Information about datasets

**2.**[**Understand the data**](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#Data)

**4.**[**Exploring some Variables**](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#Explorations)

* 4.1 Ploting some graphical and descriptive informations

**5.**[**Prediciton Model Construction and Evaluation**](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#modelling)  
[Correlation of data](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#Correlation)\*\*

**6.**[**Preprocess**](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#Preprocessing)

* 6.1 Importing Librarys
* 6.2 Setting X and Y
* 6.3 Spliting the X and Y in train and test

**7. [Model 1 Random Forest](#Modelling 1)**

* 7.1 Random Forest
* 7.2 Score values
* 7.3 Cross Validation

**8. [Model 2 Logistic Regression](#Modelling 2)**

* 8.1 Logistic Regression
* 8.2 Score values
* 8.3 Cross Validation
* 8.4 ROC Curve

# ****Understand the**** Problem

## What is the Purpose of Credit Risk Analysis?

The purpose of credit risk analysis is to assess the likelihood of a borrower defaulting on their financial obligations and to evaluate the potential risks associated with lending money to them. Credit risk analysis is important for lenders, as it allows them to make informed decisions about extending credit to individuals or businesses, while mitigating the risk of financial losses due to default or non-payment.

Credit risk analysis involves a comprehensive evaluation of a borrower's credit history, financial status, and ability to repay their debts. This analysis may include an assessment of the borrower's credit score, income, assets, liabilities, and other relevant factors that could impact their ability to repay a loan. The credit risk analysis process also considers economic factors, such as market conditions and macroeconomic indicators, that could impact a borrower's ability to repay a loan.

Credit risk analysis helps lenders make informed decisions about the terms of the loan, such as the interest rate, collateral requirements, and repayment period. Lenders may also use credit risk analysis to determine whether or not to approve a loan application and, if approved, the maximum amount of credit that can be extended.

Overall, credit risk analysis is important for lenders to manage their exposure to risk and to make informed lending decisions. By assessing credit risk, lenders can determine the appropriate terms for lending money and reduce the likelihood of financial losses due to default or non-payment.

## What is the result of Credit risk analysis?

If the borrower presents an acceptable level of default risk, the analyst can recommend the approval of the credit application at the agreed terms. The outcome of the credit risk analysis determines the risk rating that the borrower will be assigned and their ability to access credit.

When calculating the credit risk of a particular borrower, lenders consider various factors commonly referred to as the **“5 Cs of Credit.”** The factors include the borrower’s **capacity** to repay credit, **character**, **capital**, **conditions**, and **collateral**. The lender uses the factors to evaluate the characteristics of the borrower and conditions of the loan to estimate the probability of default and the subsequent risk of financial loss.

## What tools can be used for Credit risk analysis?

Credit analysts may use various financial analysis techniques, such as ratio analysis and trend analysis to obtain measurable numbers that quantify the credit loss. The techniques measure the risk of credit loss due to changes in the creditworthiness of borrowers.

When measuring the credit loss, we consider both losses from counterparty default, as well as deteriorating credit risk rating.



## The data set

This tutorial we will use [**German Credit Dataset**](https://www.kaggle.com/datasets/uciml/german-credit) to illustrate the credit analysis process and tools. (the dataset is available with this tutorial.

### German-Credit-Dataset

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.

## Content

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**(text - little, moderate, quite rich, rich)  
**Credit amount** (numeric, in DM)  
**Duration** (numeric, in month)  
**Purpose**(text: car, furniture/equipment, radio/TV, domestic appliances, repairs, education, business, vacation/others  
**Risk**(Value target - No or Yes)

# ****Understanding of Data****

Data are a collection of facts, statistics, or information that can be processed or analysed to derive insights or conclusions. Understanding data involves comprehending the data quantity and quality, meaning, significance, and patterns within the data.

There are several methods that can be used to understand data, including:

* Descriptive statistics or **Descriptive data analysis (DDA)** : This involves summarizing the data using measures such as mean, median, mode, and standard deviation.
* **Exploratory data analysis (EDA)**: This involves visually analysing the data to identify patterns, outliers, and relationships between variables. This involves representing the data using charts, graphs, and other visual aids to help identify patterns and relationships.
* Inferential statistics or **predictive data analysis (PDA)**: This involves making predictions and drawing conclusions about a larger population based on a sample of the data.

## 2.1 Understanding of data with DDA

**Descriptive data analysis (DDA)** is a foundational tool for data analysis, and it is often used in conjunction with other advanced tools for data analysis such as predictive modelling and machine learning. Predictive modelling involves using statistical algorithms to analyse data and make predictions about future outcomes. Machine learning involves using algorithms to analyse data and automatically improve performance based on new data.

DDA is an important first step in the data analysis process because it provides a basic understanding of the data and identifies patterns and trends that can inform the development of more complex predictive models. Predictive modelling builds on the insights gained from DDA by identifying key variables that are most predictive of the outcome of interest, and then using those variables to develop a predictive model. This model can be used to make predictions about future outcomes based on new data.

First of all, we need to import necessary libraries. In any data analysis project with Python, we need: **Pandas, NumPy**, **Seaborn** and **Matplotlib** at the least.

#Load the necessary librarys

import pandas as pd #To work with dataset

import numpy as np #Math library

import seaborn as sns #Graph library that use matplot in background

import matplotlib.pyplot as plt #to plot some parameters in seaborn

import warnings

warnings.filterwarnings('ignore')

cmap=sns.color\_palette('Blues\_r')

### Import raw data

Now let us import the source data. This is an important step in any data analysis project. It is normally called data aggregation. There are several methods support this step:

* Importing from flat files (txt, CSV, Excel, HDF5, SAS) using pandas
* Import data from RDB
* Import data from a Web API
* Web Scraping

Yulei’s blog (Sandbox) has many pythons basic usage for Data Science. It is worth to have a look.

[Importing Data in Python - Yulei's Sandbox (yuleii.github.io)](https://yuleii.github.io/2020/06/22/importing-data-in-python.html)

#Importing the data

Df\_credit = pd.read\_csv("german\_credit\_data.csv",index\_col=0)

We import the data from a file called “german\_credit\_data.csv” in the current directory.

Let us have the first glance of it.

# Check DataFrame df\_credit

Df\_dredit.head()

Table

Description automatically generated

It confirms the 10 attributes introduced in the data file contents. We can see the “**Risk**” attribute has two value “good” and “bad”. Not “Yes” or “No”. We can also observe that attribute “**Saving account**” has value “NaN”, it generally means “not available”.

### General describe of data

Let us to Descriptive analysis. That normally including assess data Quantity (dataset size or volume) and Quality (any missing values or obvious problem like NaN).

# check data size and Missings values,type of data

print(df.info())

<class 'pandas.core.frame.DataFrame'>

Int64Index: 1000 entries, 0 to 999

Data columns (total 10 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Age 1000 non-null int64

1 Sex 1000 non-null object

2 Job 1000 non-null int64

3 Housing 1000 non-null object

4 Saving accounts 817 non-null object

5 Checking account 606 non-null object

6 Credit amount 1000 non-null int64

7 Duration 1000 non-null int64

8 Purpose 1000 non-null object

9 Risk 1000 non-null object

dtypes: int64(4), object(6)

memory usage: 85.9+ KB

None

The results above tell us what? Can you see the size of the dataset? What is the data type **“**object**”** mean?

(Try to add self-annotation in place you may have issues. At the end you can have a your own personal notebook.)

Here are my note: Findings: With this basic data fummary (only applies to Int attributes), we can see that

* The data size is: 1000 (smaples) X 10 (attibutes).
* Among the 10 attributes, Risk is the target variable (it is also called dependent variable in machine learning and predictive analysis. it is the variable that we need to predict for new records)
* There are 4 avrables with int64 type, and 6 with object(string) type.
* We can see the range of each variables the most value is 921 from Credit amount, and the least is a binary from Sex, and Risk.
* We can also see that Saving accounts and Checking account have missing values since their "non-null" is less than the number of samples (1000).
* Job type is NOT correct. It should has a categorical type not Int. This is arguable at least. We will see the data transform - change types and values later.

### Check attributes value uniqueness

Another thing is important to know about our dataset is check individual attributes **domain** (value range). We do this by run the **.unique()** function from Pandas.

#Looking unique values

print(df.nunique())

Age 53

Sex 2

Job 4

Housing 3

Saving accounts 4

Checking account 3

Credit amount 921

Duration 33

Purpose 8

Risk 2

dtype: int64

The results tell us that the number of the different values in each attribute. For example, **Age** has 53 different values and **Sex** has two different values. Those value reflect the Uniqueness of the attributes’ value. But they do not tell anything about the actual value themselves. That is what we want to find out next.

### Five-number summary

We can use **.describe()** form Pandas to give us “**the five-number summary**”.

#look into the 5-number summary of individual attribute

print(df.describe())

Text

Description automatically generated

Notice that the “5-number summary” only applies to integer attributes (dtypes: int64).

**Findings:** With this basic data summary (only applies to Int attributes), we can see that

* The records count: 1000 (smaples)
* The 4 integer attributes' mean and std
* The statistical 5-number summary: Min, 1st Qtr, 2nd Qtr and 3nd Qtr and Max

## 2.2 Data quality assessment with DDA

Data quality assessment is an important step a data analysis project. It involves evaluating the quality of the data to ensure that it is suitable for analysis. The goal of data quality assessment is to identify any data quality issues, such as missing or inconsistent values, outliers, errors, or duplicates.

Once data quality issues are identified, data preprocess is need to resolve these issues. therefore, data quality assessment is, in some way, setting target and goals for data preparation. This involves defining the data cleaning and transformation steps necessary to address the identified issues and improve the quality of the data.

### Finding missing values and the percentage

#Get the numbers of missing values in each column

missing = df.isnull().sum()

#Get the percentage of missing values in each column

missing\_pct = round(df.isnull().sum()/len(df) \* 100, 1)

data\_missing = pd.concat([missing, missing\_pct], axis=1)

data\_missing.columns = ["Num", "Pct"]

print(data\_missing)

#Plot missing values

data\_missing['Pct'].plot(kind= "bar")

plt.show()

Num Pct

Age 0 0.0

Sex 0 0.0

Job 0 0.0

Housing 0 0.0

Saving accounts 183 18.3

Checking account 394 39.4

Credit amount 0 0.0

Duration 0 0.0

Purpose 0 0.0

Risk 0 0.0

A picture containing icon

Description automatically generated

The result shows there are 183 missing values in “**Saving accounts**” and 194 missing values in “**Checking account**”. The percentage of missing values are 18.3 and 39.4 respectively. They are **Alarmingly high** and needs investigate and understand why. In data science there are many methods can be sued to deal with the missing values. We will deal with the missing values later.

# Data Attribute Analysis with EDA

**Exploratory Data Analysis (EDA)** is another important and useful data analysis method that can be used to understand the structure and characteristics of the data. It certainly used in understanding fo the whole dataset such as the distribution and overall dataset shape etc.

One important aspect of EDA is the analysis of data attributes. Data attributes are the characteristics or features of a dataset, such as numerical values, categorical variables, or dates. Data attribute analysis is the process of exploring and analysing the various attributes of a dataset to gain insights into the data and inform subsequent analysis. This analysis includes examining the distributions, frequencies, and relationships between different data attributes, as well as identifying any outliers or missing values.

The goal of data attribute analysis is to understand the underlying patterns and relationships within the data, which can then inform the selection of appropriate statistical methods, models, and algorithms for further analysis. For example, data attribute analysis may reveal the presence of significant correlations between different attributes, which can be exploited to develop predictive models or identify important factors that affect a particular outcome.

## 3.1 Univariate analysis

Univariate analysis is a statistical analysis technique that involves examining the distribution and characteristics of a single variable in a dataset. It is a method used to understand the characteristics and behaviour of a single variable and is often used as a preliminary analysis to explore and summarize data.

In univariate analysis, a single variable (attribute) is examined in isolation, and various statistical measures such as mean, median, mode, range, variance, standard deviation, and percentiles are used to describe the distribution of the data. Visualization techniques such as histograms, box plots, and density plots are also commonly used to represent the distribution of the data and identify any outliers, gaps, or skewness.

Univariate analysis is commonly used to answer questions such as:

* What is the range of values that the variable takes?
* What is the most common value of the variable?
* What is the average value of the variable?
* What is the variability or spread of the data?

Univariate analysis is often used as a starting point for more complex analyses such as multivariate analysis, where the relationships between multiple variables are examined. By first understanding the distribution and characteristics of individual variables through univariate analysis, it becomes easier to identify potential relationships and patterns in the data that may be relevant for more advanced analysis.

### Attribute with Object type and their distribution analyses

We know from .info() function call earlier that the whole dateset has 6 attributes with type **object**. and they are:

'Sex', 'Housing', 'Saving accounts', 'Checking account', 'Purpose', 'Risk'

To illustrate a general EDA approach in python, we first define a function and then invoke the function to plot the graph.

# Distribution graphs (histogram/bar graph) of object column

def plotObjColumnDistribution(df, nGraphShown, nGraphPerRow):

print(f'=== Distribution of features with object values ===')

df = df[[col for col in df.select\_dtypes(["object", "category"])]] # pick columns with object type

nRow, nCol = df.shape

columnNames = list(df)

print(columnNames)

nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow

plt.figure(num = None, figsize = (6 \* nGraphPerRow, 5 \* nGraphRow), dpi = 80, facecolor = 'w', edgecolor = 'k')

for i, col in enumerate(columnNames):

# Plot distribution

plt.subplot(nGraphRow, nGraphPerRow, i + 1)

columnDf = df.iloc[:, i]

valueCounts = columnDf.value\_counts()

valueCounts.plot.bar()

plt.ylabel('counts')

plt.xticks(rotation = 90)

plt.title(f'{columnNames[i]} (column {i})')

# Show the plot

plt.tight\_layout()

plt.show()

Function invocation.

plotObjColumnDistribution(df, 10, 3)

Chart, bar chart

Description automatically generated

**Findings**:

1. Among the 6 categorical attributes. The value is distributed **Unevenly** most are **Skewed** to the left. It may affect the prediction later.
2. The targeted dependent variable **Risk** is **imbalanced** the ratio is 700:300 between "good" and "bad".

### Attribute with numerical type and their distribution analyses

We will do the same for **numerical** attributes. There are 4 of them: 'Age', 'Job', 'Credit amount', 'Duration' although we don't think job should be a numerical type.

# Distribution graphs (histogram/bar graph) of column data with (num)

def plotNumColumnDistribution(df,nGraphPerRow):

print(f'=== Distribution of features with number values ===')

df = df[[col for col in df.select\_dtypes(np.number)]] # pick columns with bumber type

nRow, nCol = df.shape

columnNames = list(df)

print(columnNames)

nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow

plt.figure(num = None, figsize = (6 \* nGraphPerRow, 3 \* nGraphRow))

for i, col in enumerate(columnNames):

# Plot distribution

plt.subplot(nGraphRow, nGraphPerRow, i + 1)

sns.distplot(df[col], color='blue')

plt.title(f'Distribution of {col}')

# Show the plot

plt.tight\_layout()

plt.show()

plotNumColumnDistribution(df, 2)

Histogram

Description automatically generated

**Findings**:

Among the 3 numerical attributes. The two values **Age** and **Credit amount** are distributed **Unevenly** and are **Skewed** to the left. It may affect the prediction later.

## 3.2. Bivariate analysis

In contrast to univariate analysis, **Bivariate analysis** is another statistical analysis technique that involves examining the relationship between two variables in a dataset. Particularly, it is commonly used to analyse the interdependence and association between dependent variable and other variables.

Bivariate analysis is often used to answer questions such as:

* Is there a correlation or association between two variables?
* Does the relationship between two variables follow a particular pattern or trend?
* How do changes in one variable affect the other variable?

Bivariate analysis typically involves the use of various statistical methods, such as correlation analysis, regression analysis, and chi-square tests, to measure the strength and direction of the relationship between two variables. In addition, various visualization techniques such as scatter plots, heatmaps, and contour plots are commonly used to help identify patterns and trends in the data.

### Pairwise attributes plot with Seaborn

Pairwise attribute plot is a type of data visualization technique that allows us to visualize the relationships between pairs of variables in a dataset. Seaborn, a popular data visualization library in Python, provides an easy way to create pairwise attribute plots using the pairplot() function. This plot can be useful for identifying patterns and correlations between different attributes in the dataset.

# Seaborn pairplot() can produce a grid of Axes such that each numeric variable in data will by shared across the y-axes

# Across a single row and the x-axes across a single column.

sns.pairplot(df, hue="Risk")

A picture containing text, electronics, display

Description automatically generated

Notice that it only shows the correlation between numerical attributes and the dependent variable **“Risk”.**

It is hard to interpret the relationships in the graph. We can show more explicit relations between **“Risk”** and other numerical attributes.

### Check numerical variables with Risk by histogram

# Distribution graphs (histogram/bar graph) of Numerical columns related with dependent variable risk

def plotRiskNumColumnDistribution(df, nGraphPerRow):

df2 = df[[col for col in df.select\_dtypes(np.number)]] # pick columns with number type

nRow, nCol = df2.shape

columnNames = list(df2)

nGraphRow = (nCol + nGraphPerRow - 1) / nGraphPerRow

fig, axes = plt.subplots(int(nGraphRow), nGraphPerRow, figsize=(15, 8))

axes = axes.ravel()

for ax in axes:

ax.set\_axis\_off()

for i in range(len(columnNames)):

#print (df[df["Risk"]=="No"][columnNames[i]].head())

axes[i].hist(df[df["Risk"]=="good"][columnNames[i]], alpha=0.5, color='blue', label='No')

axes[i].hist(df[df["Risk"]=="bad"][columnNames[i]], alpha=0.5, color='red', label='Yes')

axes[i].legend(prop={'size': 10})

axes[i].set\_title(str(columnNames[i]))

axes[i].set\_axis\_on()

fig.tight\_layout()

plt.show()

plotRiskNumColumnDistribution(df, 2)

Graphical user interface, chart

Description automatically generated

**Findings**:

Among the 3 numerical attributes. The two values **Age** and **Credit amount** are distributed **Unevenly** and are **Skewed** to the left. The Risk is proportionally with the distribution. It may affect the prediction later.

### Check categorical variables against Risk

# Distribution graphs (histogram/bar graph) of object column

def plotRiskObjColumnDistribution(df, nGraphPerRow):

print(f'=== Distribution of features with object values ===')

df2 = df[[col for col in df.select\_dtypes(["object", "category"])]] # pick columns with object type

nRow, nCol = df2.shape

#print(df2)

columnNames = list(df2)

#print(columnNames)

columnNames = [item for item in columnNames if item != "Risk"]

print(columnNames)

nGraphRow = int((len(columnNames) + nGraphPerRow - 1)/nGraphPerRow)

#print(nGraphRow)

#the folloing code was suggested by ChartGPT.

#it creat a plot first then plot subplot into the plot so the size can be controled.

#creat a plot

fig, axes = plt.subplots(nrows=nGraphRow, ncols=nGraphPerRow, figsize=(6\*nGraphPerRow, 5\*nGraphRow))

#loop attributes to plot individual plots

for i, col in enumerate(columnNames):

ax = axes.flat[i]

df\_pct = df2.groupby([columnNames[i],'Risk'])['Risk'].count()/df2.groupby([columnNames[i]])['Risk'].count()

df\_pct.unstack().plot.bar(stacked=True, width=0.8, ax=ax)

ax.set\_ylabel('counts percent %')

ax.set\_title(f'Risk distribution with {columnNames[i]}')

plt.tight\_layout()

plt.show()

Here are the function call and the results,

plotRiskObjColumnDistribution(df, 2)

A screenshot of a computer

Description automatically generated with medium confidence

Chart, treemap chart

Description automatically generated

You are asked to interpret the graph.

# ****Data Preprocessing****

Following the step of "understanding data", we have a full understand of the raw data (data we have imported form the data source). We have identified the data quantity and quality issues. those issues set goals for **"data preprocessing"** to resolve.

**Data preprocessing** is an essential step in any data analytical project. It involves **transforming raw data into a format that is suitable for analysis**, which can involve a range of tasks such as cleaning, normalization, feature selection, and dimensionality reduction. Data preprocessing is critical because it can have a significant impact on the accuracy and reliability of the results obtained from the analysis. By ensuring that the data is properly prepared, data scientists can help to eliminate inconsistencies, reduce errors, and optimize the performance of their analytical models. As such, data preprocessing is a fundamental step in any data analytical project and should be given careful consideration to ensure that the analysis is based on high-quality data.

Data processing will generally produce a new dataset based on the raw dataset. We call a new dataset "df\_credit". it should be more suitable for our prediction model construction later.

Data preprocessing can involves more complicated and tedious tasks. For illustration purpose, we only do the following:

1. We need to convert the target variable **Risk** into a numerical.
2. We need to split df\_credit into predict variable and dependent variable.
3. we do some individual variable **transformation**, **fill missing data**

## 4.1 Prepare for target variable *****Risk*****

In a data prediction model, it is generally necessary to convert the target variable into a numerical variable rather than a categorical variable. This is because most machine learning algorithms require numerical data to work properly and cannot handle categorical data directly.

It is a good idea and common practice that you do data manipulation on a new dataframe and keep the original untouched.

#Importing the data

df = pd.read\_csv("german\_credit\_data.csv", index\_col=0)

#It is a good idea and common practice you do data manipulation on a new dataframe and keep the original untouched

df\_credit = df

Now, let us change “**Risk”** type and check,

#Change Risk column (target variable) into numerical with 0 and 1 values

df\_y = df[['Risk']]

df\_y['Risk'] = np.where(df\_y.loc[ : , 'Risk']=='good', 0, 1)

#Check

df\_y.info()

<class 'pandas.core.frame.DataFrame'>

Int64Index: 1000 entries, 0 to 999

Data columns (total 1 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 Risk 1000 non-null int32

dtypes: int32(1)

memory usage: 11.7 KB

df\_y.head()

Risk

0 0

1 1

2 0

3 0

4 1

## 4.2 Prepare for predictor variable

It is also a common practice to drop target variable form the dataset, so it only contains the predict variables. We will drop the **“Risk”** from df\_credit.

#Drop Risk

df\_credit = df\_credit.drop(["Risk"], axis=1)

df\_credit.head()

Table

Description automatically generated

## 4.3 Deal with missing values

Missing values are a common issue that can occur in many datasets (for example, in our dataset we have 183 missing values in **Saving accounts** and 394 missing values in **Checking account**), and they can have a significant impact on the accuracy and reliability of data analytical models.

There are several methods that can be used to handle missing values, depending on the nature and context of the missing data. One common approach is to impute missing values with estimated or predicted values. This can be done using methods such as mean or median imputation, regression imputation, or machine learning imputation. Another approach is to remove observations with missing values, either entirely or partially, depending on the amount of missing data and the context of the analysis. Alternatively, if the missing values are believed to be related to other variables, methods such as expectation-maximization (EM) or multiple imputation can be used to impute missing data in a more sophisticated manner.

The choice of method will depend on the specific data analytical project and the nature of the missing values, as each approach has its own advantages and limitations. However, it is crucial to handle missing values appropriately to ensure that the analysis is based on high-quality and reliable data.

#Saving accounts has 183 missing values and Checking account 394 missing values

#check Saving accounts

print(df\_credit['Saving accounts'].describe())

#plot the distribution of the existing values

plt.figure(figsize=(12,6))

sns.countplot(x='Saving accounts',data = df\_credit)

plt.show()

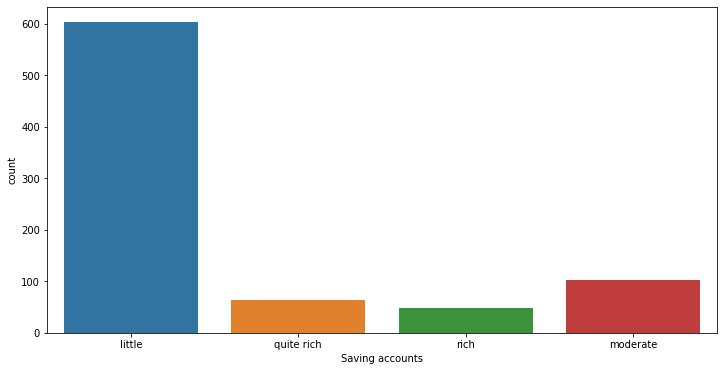
count 817

unique 4

top little

freq 603

Name: Saving accounts, dtype: object

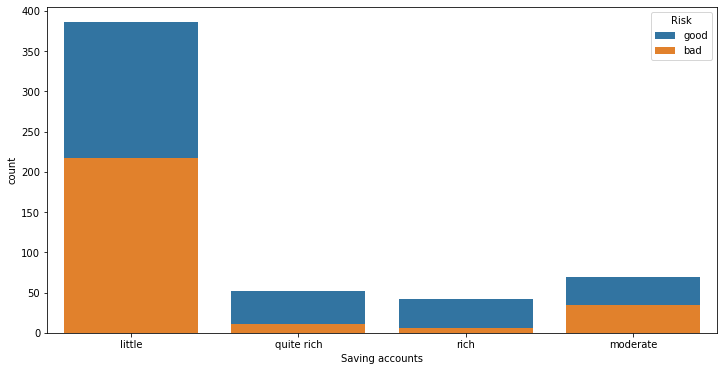


#plot a stacked bar chart for the count of the Risk variable for each value of the Saving accounts variable

plt.figure(figsize=(12,6))

sns.countplot(x='Saving accounts', hue='Risk', data=df\_credit, dodge=False)

plt.show()

****

**Observations and suggestions:**

#### From above description and plot, we can see that the value of the **saving account** distribution and **Risk** distribution over the saving account:

1. Majority is little, (603/817), the easiest solution is assigning all the missing value as little.

**Discuss its advantage and issues.** Once issue is that simple assign all the missing value as the most frequent value (in this case it is little) will introduce bias to the dataset. This is because the distribution of risk over saving account value "little" is different with the risk distribution over missing value. See the last figure.

1. The best solution would be to use an **imputation method** that takes into account the relationship between the "Saving accounts" and "Risk" variables to fill the missing values. That is to keep risk distribution over saving account unchanged after the value are assigned to the missing values.

Consider how to write python code for this? Here is an example,

import numpy as np

# get the unique values of the "Saving accounts" variable

unique\_values = df\_credit['Saving accounts'].unique()

# loop over the risk levels and impute the missing values for each level

for risk\_level in df\_credit['Risk'].unique():

# get the indices of the missing values for the current risk level

missing\_indices = df\_credit.loc[(df\_credit['Risk'] == risk\_level) & (df\_credit['Saving accounts'].isna()), :].index

# get the frequency of each unique value of "Saving accounts" within the current risk level

freqs = df\_credit.loc[df\_credit['Risk'] == risk\_level, 'Saving accounts'].value\_counts(normalize=True)

# loop over the missing indices and randomly assign one of the unique values based on their frequency

for index in missing\_indices:

imputed\_value = np.random.choice(unique\_values, p=freqs.values)

df\_credit.loc[index, 'Saving accounts'] = imputed\_value

Let us first assign all the missing value as “little” and check.

# Assign missing vlaue with little

df\_credit['Saving accounts'].fillna('little', inplace = True)

df\_credit['Saving accounts'].describe()

count 1000

unique 4

top little

freq 786

Name: Saving accounts, dtype: object

We can see that all the missing value is now filled with “little”. However, this newly added value will likely result in a biased distribution of the "Risk" variable for the "little" category.

Filling missing value is also called “**imputation**” in data science. There are many approaches for imputations.

One possible approach is to use a predictive model to impute the missing values. For example, you can train a classification model with the "Saving accounts" and "Risk" variables as input and output, respectively, and then use the model to predict the "Risk" values for the missing "Saving accounts" values. This way, the imputed values will be based on the actual relationship between the variables, rather than a simple frequency-based imputation.

Another approach is to use a more sophisticated imputation method that considers the joint distribution of the "Saving accounts" and "Risk" variables, such as multivariate imputation by chained equations (MICE) or fully conditional specification (FCS). These methods impute missing values by simulating from the joint distribution of the variables, using a series of regression models.

In any case, it's important to evaluate the performance of the imputation method on a validation set, to ensure that it doesn't introduce bias or affect the performance of downstream analyses.

Let us consider the missing values for checking account.

# Saving accounts has 183 missing values and Checking account 194 misisng values

# checking accounts

print(df\_credit['Checking account'].describe())

plt.figure(figsize=(12,6))

sns.countplot(x='Checking account',data=df\_credit)

plt.show()

count 606

unique 3

top little

freq 274

Name: Checking account, dtype: object

Chart, bar chart

Description automatically generated

**Findings**:

1. We can see that the current distribution is pretty even between **Little** and **Moderate**; and only small amount has value of **rich**
2. The targeted dependent variable **Risk** for the three category distribution are vary a lot.

**Suggestion:** among the many fill missing value methods, one of them may suitable for this situation. **that is create a new category value** like *NoInfo* itself is valuable since the number of missing values is almost the same quantity with any other two most occurrences. 274 for little and 194 missing values.

So, we will fill the missing value of the “checking account” with a category called "No\_info".

# fill the missing value with a category called "No\_info"

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

df\_credit['Checking account'].head()

#df\_credit['Checking account'].describe()

plt.figure(figsize=(12,6))

sns.countplot(x='Checking account',data=df\_credit)

Chart, bar chart

Description automatically generated

Let us consider another important task in data preprocessing that is **Data Transformation**.

## 4.4 Deal with distribution was extremely skewed variables[¶](http://localhost:8888/notebooks/OneDrive/Documents/Work/Teach/BDA%20for%20Risk%20Management/Github%20notebook/Financial-Risk-Management-with-Bank-Loan/predicting-credit-risk-final-gli.ipynb#4.4-Deal-with-distributuion-was-extreamly-skewed-varaibles)

In section 3.1, we have identified that the numerical variables have skewed value distribution. This will affect most the prediction model and make them perform badly when facing non-skewed and randomly distributed data. To over come this problem we need to do a **Data transformation.**

**Data transformation** is an essential step in data preprocessing, where the goal is to convert raw data into a more usable and informative format for subsequent analysis. This can involve converting data into a standard format, such as changing categorical data into numerical data or converting text data into structured data, to make it easier to analyse. Additionally, data transformation can involve rescaling or normalizing data to make it more comparable or reducing the number of features through feature selection or dimensionality reduction techniques. Some examples of data transformation techniques include normalization, standardization, logarithmic transformations, and one-hot encoding. The choice of data transformation techniques largely depends on the type of data, the analysis goals, and the statistical and machine learning models used for analysis. By transforming data, it becomes easier to identify patterns, reduce noise, and improve the accuracy of predictions or models.

For the illustration purpose we will perform a **log transformation** on them to overcome the problems.

We firstly reproduce the distribution graph so we can compare after our transformation.

#performming a log transformation on numerical varibles

plotNumColumnDistribution(df, 2)

Histogram

Description automatically generated

# Log transform function

def log\_transform(data, to\_log):

X = data.copy()

for item in to\_log:

# Add 1 to the data to prevent infinity values

X[item] = np.log(1+X[item])

return X

Do the transformation,

# log transfer

df\_credit = log\_transform(df\_credit, ['Age', 'Credit amount', 'Duration'])

plot the result and check,

# check the result

plotNumColumnDistribution(df\_credit, 2)

A picture containing chart

Description automatically generated

We can see that the numerical value is now nearly normally distributed.

## 4.5 Transfer all object and category variables into numerical

For demonstrate purpose, we will convert another important data transform in data pre-processing that is data type transform mainly from categorical to numerical.

In many real-world datasets, categorical data is commonly used to describe non-numeric data such as gender, colour, or country. However, many statistical and machine learning models require numerical data for analysis. Therefore, a necessary step in data preprocessing is to transform categorical data into numerical data. One common method for converting categorical data to numerical data is one-hot encoding. This involves creating a binary variable for each unique category in the original categorical variable, where the value of 1 indicates the presence of the category, and 0 indicates the absence. Another method is label encoding, where each unique category is mapped to a unique integer value. This can be useful for ordinal categorical data, where there is a natural ordering between the categories. A disadvantage of label encoding is that the integer values may imply a natural ordering that may not be present in the data, which may lead to erroneous modelling. Therefore, one-hot encoding is typically preferred. Overall, data type transformation is an essential step in data preprocessing to make the data suitable for subsequent analysis and modelling.

We need to do this for 2 reasons:

1. most of the prediction models need predictor to be numerical.
2. for correlation analysis we need attributes to be numerical too.

We start to define an “one\_hot\_encoder”,

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)

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

return df

then we do the transformation and generate a new dataframe called **“df\_credit\_T”.**

# do transformation on df-credit

df\_credit\_T = one\_hot\_encoder(df\_credit)

print(df\_credit\_T.info())

Table

Description automatically generated

We can see that now our new dataframe has 24 variables and all are numerical type.

After all the df\_credit change into **numerical** we can see the relation between them with the target variable **Risk**. We need to connect them together to form **a training dataset**.

## 4.6 Full training dataset preparation

Any prediction model needs to be trained with a **training dataset** that has several essential characteristics to ensure the model can learn and make accurate predictions. One critical aspect of a training dataset is its structure and size, which should be sufficient to capture the full range of variability and patterns that the prediction model is expected to encounter in real-world data. Additionally, the dataset should be representative of the population or data the model will be operating on, meaning it should include a diverse range of data points that cover the full spectrum of potential inputs and outcomes. The quality of the training dataset is also essential since low-quality data can lead to biased or inaccurate predictions. A balanced training dataset is crucial to prevent the model from being biased toward the majority class, and it should have labelled examples that correspond to the outputs the model is attempting to predict. Overall, the quality and characteristics of the training dataset play a critical role in determining the accuracy and reliability of a prediction model.

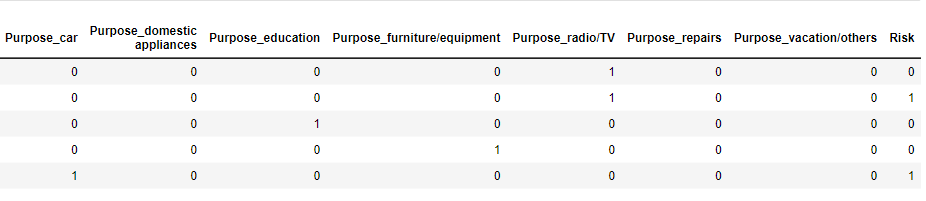
#concat the df\_credit with df\_y

data\_train = pd.concat([df\_credit\_T, df\_y], axis=1)

data\_train.head()

Table

Description automatically generated



5 rows × 26 columns

# check distributions

plotNumColumnDistribution(data\_train, 3)

A picture containing window, large, platform

Description automatically generated

After this basic data preprocessing, we have produced a new dataset ready for to be used for our model construction.

* df\_credit\_T: Dataset only contains independent variables after normalisation.
* df\_y: dataset only has the dependent variable "Risk" and only has 0 and 1.
* data\_train: concatenation of the two.

We are ready to build our prediction models. We also need to assess our model’s performance. That is the next action.

# ****Prediction Model Construction and Evaluation****

Python offers a rich library of machine learning tools, including the widely-used **Sklearn** library. Sklearn's implementation of machine learning models as Python classes allows for a straightforward implementation process. Using pre-processed data, we can instantiate these classes and make an initial assessment of the models' performance.

However, developing a good prediction model typically involves an iterative process, which may require revisiting data processing, such as feature selection or data transformation, to improve data quality for the models.

In order to gain an initial impression of the possible prediction models, we will utilize several popular models offered by Sklearn, including: the Random Forest Classifier, Logistic Regression models, Decision Tree Classifier, K-Nearest Neighbours Classifier, Gaussian Naive Bayes, SVC, and XGBoost, using all default parameters without specification. Random Forest Classifier

* Logistic Regression models
* Decision Tree Classifier
* K-Nearest Neighbours Classifier
* Random Forest Classifier
* Gaussian Naive Bayes
* SVC
* XGBoost

Use all the default parameters without any specification.

## 5.1 Load needed library

# liberary loading

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.naive\_bayes import GaussianNB

from sklearn.svm import SVC

#from xgboost import XGBClassifier

## 5.1 Prepare the 4 datasets

Any prediction models from Sklean needs the 4 datasets for training and testing to assess its performance. They are:

* X\_training: dataset only has all the predict variables’ value for training.
* y\_training: dataset only has values of target variable (Risk) for training.
* X\_test: dataset only has all the predict variables’ value for testing.
* y\_test: dataset only has values of target variable (Risk) for testing.

Those are generated by a split factor traing: testing. Most commonly used proportion is 80:20, 75:25; and 70:30.

# prepare for training and testing datasets for Sklearn models

X = df\_credit\_T

y = df\_y

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

## 5.2 Train and assess prediction models

# to feed the random state

seed = 42

# prepare models

models = []

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

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

models.append(('DT', 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 = []

for name, model in models:

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

pre\_results = model.predict(X\_test)

model\_acc = accuracy\_score(y\_test, pre\_results)

names.append(name)

results.append(model\_acc)

msg = "Modle %s has: %0.3f accuracy" % (name, model\_acc)

print(msg)

#print(names)

#print(results)

# boxplot algorithm comparison

results\_df = pd.DataFrame({'Name':names, 'Results':results})

#print(results\_df)

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

fig.suptitle('Algorithm Comparison')

results\_df.plot.bar(x='Name',rot=45)

plt.show()

Model LR has accuracy: 0.736

Model KNN has accuracy: 0.736

Model DT has accuracy: 0.644

Model NB has accuracy: 0.660

Model RF has accuracy: 0.752

Model SVM has accuracy: 0.748

Model XGB has accuracy: 0.752

A picture containing icon

Description automatically generated

We can see how easy the model can be constructed and how the results can be predicted. However, we don’t know the results are best or not. This can be conformed or improved after the models fine tune.

**Findings:**

* All models have pretty high/low accuracy.
* Some models perform a little better than others.

# ****Prediction model fine tune****

Fine-tuning a prediction model is crucial to achieving optimal performance, particularly when training the model using our own data. Without a pre-trained model, we must start from scratch and find the optimal set of model parameters and hyperparameters to fit our unique data. In addition, fine-tuning requires experimentation with various feature combinations, data pre-processing techniques, and other factors that can significantly impact model performance. Each of these factors is specific to the data we are using, making fine-tuning an iterative process that requires careful consideration and experimentation. To fine-tune the model, we may use general methods such as grid search, random search, or Bayesian optimization to explore the parameter space and identify the best performing model. However, it is important to keep in mind that fine-tuning is a time-consuming process that requires patience and expertise, as even small adjustments can have a significant impact on model performance. In the end, taking the time to fine-tune the model leads to better performance, more accurate predictions.

There are two major tasks involved in fine-tuning a prediction model:

1. **Improving the quality of training samples**. The samples used for model training have a significant effect on the model's performance. Proper selection of the right number and type of attributes can increase the model's accuracy but also increase training costs. The two most popular methods for attribute selection are:
   1. correlation analysis and
   2. K-fold cross-validation.
2. **Adjusting model parameters.** Each prediction model is defined by its model parameters. However, during the training process, we must select optimal hyperparameters that the learning algorithm will use to learn the best parameters for mapping the input attributes (independent variables) to the labels or targets (dependent variable). Hyperparameters vary depending on the prediction model used. Various methods can be used to fine-tune these hyperparameters, depending on the specific model being used.
   1. **K-fold cross validation**. There are many methods to increase sample numbers. Such as upsampling, it can overcome imbalance class. However, upsampling can cause overfitting. Thus, k-fold cross validation will be used because this method is more superior than the simple train/test split to prevent overfitting. In addition, K=5 is chosen because empirical data showed that this value produces lower error rate and has a balanced bias vs variance (An Introduction to Statistical Learning, 2017).
   2. **Stratification.** Due to class imbalance, for k-fold cross validation, we might end up with folds which has no representation from the minority class (Jason Brownlee, 2020). Hence, stratification will be applied to ensure that the original class distribution is maintained in each fold.
   3. **SMOTE method.** If upsampling is done before cross validation, we risk data leak traps since the same upsampled data might appear in both the training and validation folds. Hence, SMOTE method will be used to segregate the data into its respective folds before up-sampling is done in each fold. SMOTE is selected because of its low chances of overfitting and information loss. (Nitesh, 2002)
   4. **Log transformation and Data scaling**. Features in our dataset are skewed and have high variance. Such high variance will negatively impact the accuracy of the model. Thus, log transformation is done to reduce features’ variance. For features with minimum value of zero, 1 is added before taking log to prevent infinity values. Thereafter, Z-score normalization is done so that features are compared on a similar scale.
   5. **Hyper-parameter tunning**. Hyper-parameters must be tuned to overcome bias-variance trade off in the models. For example, in Random Forest, when the number of trees is low at the extreme left, underfitting results in high bias, thus high errors. As number of trees is increased towards the right side of the graph, overfitting sets in, resulting in low training error but increasing validation error. Thus, hyper-parameters must be fine-tuned to select the best param which prevent overfitting and underfitting. The hyper-parameters will be fine-tunned using GridSearchCV.

## ****6.1 Features selection****

For any prediction models, features selection is important. It affects the model's performance by

1. the number of the properties used (not the more the better).
2. Which property to use (not use the ones are correlated).

The two general methods used are:

* Correlation analysis
* PCA analysis

### Correlation Analysis

# Correlation analysis of on the data\_train rather than df\_credit\_T

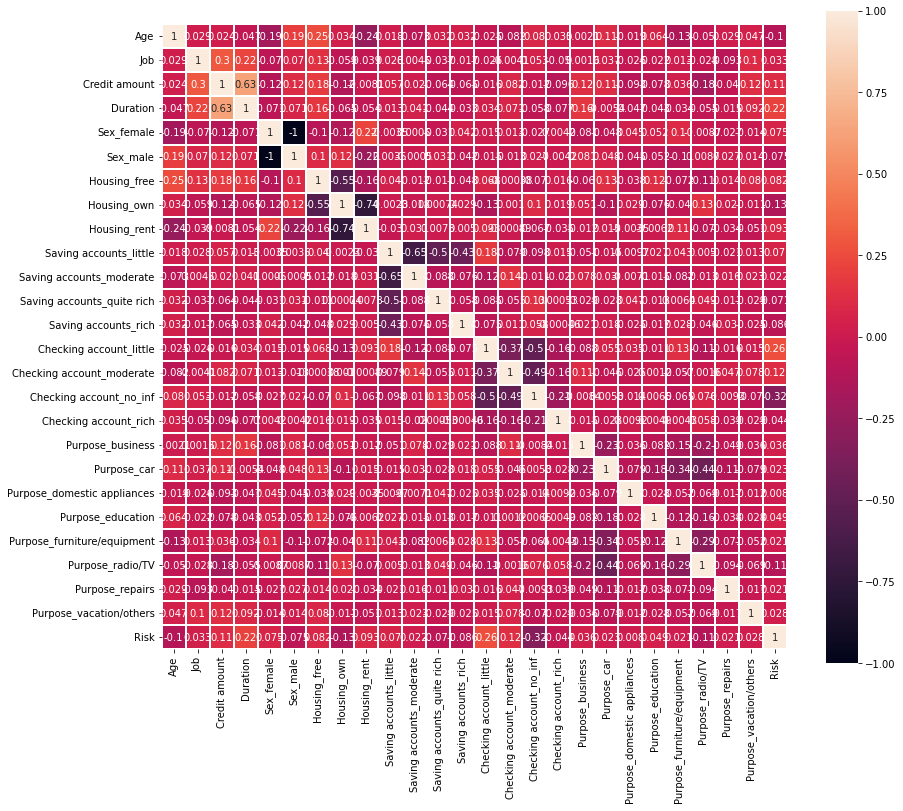
# Because we want see the correlation with Risk

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

sns.heatmap(data\_train.astype(float).corr(),linewidths=0.1,vmax=1.0,

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

plt.show()



Above correlation heatmap shows **the correlations among the attributes**. Apart from the correlations between attributes, Our attention should focuse on the correlation between the predictor variables and also shows the correlations between predictors and the target variable **Risk**.

**Findings:**

1. The most positive correlation with **Risk** are: **checking account\_little, Duration, saving account little, house\_rent and credit\_amount**.
2. **Checking account\_no\_info** has a negative on the **Risk**.
3. Correlation analysis provides basic information on the connections between predictors and the target variable.

### The importance of the independent (predictors) variable with target variable in order

# show the most influenced variables

var\_importance = data\_train.corrwith(data\_train["Risk"]).abs()

var\_importance.sort\_values(ascending=False)

Table

Description automatically generated with medium confidence

**Findings:**

With this order we can decide how many and which parameters to use.

### PCA Analysis

With PCA, we can also find **the number of features required** to explain the variance in all candidate features. We have 25 features that all can potentially be predictors. Let us see their prediction power over target variable.

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

n=25

columns=['PCA\_1', 'PCA\_2', 'PCA\_3', 'PCA\_4', 'PCA\_5', 'PCA\_6', 'PCA\_7',

'PCA\_8', 'PCA\_9', 'PCA\_10', 'PCA\_11', 'PCA\_12','PCA\_13','PCA\_14',

'PCA\_15', 'PCA\_16', 'PCA\_17', 'PCA\_18', 'PCA\_19', 'PCA\_20', 'PCA\_21',

'PCA\_22', 'PCA\_23', 'PCA\_24', 'PCA\_25'

]

PCA is affected by scale. So we need to scale the features in our data before applying PCA. The scale is done by using **StandardScaler**. It can standardize the dataset’s features onto unit scale (mean = 0 and variance = 1) which is a requirement for the optimal performance of many machine learning algorithms.

# Standardizing the features

# df\_credit\_T is the origianl dataset applied

X = StandardScaler().fit\_transform(df\_credit\_T)

### Using PCA ****.fit\_transform()**** to find PCs

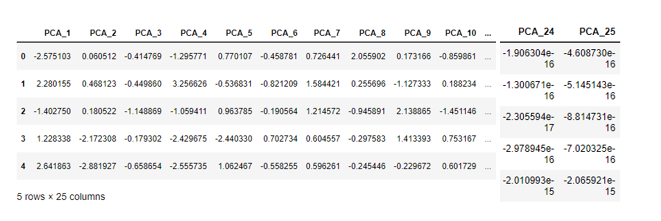
# Create the PCA instance and fit and transform the data with pca

pca = PCA(n\_components=n)

pc = pca.fit\_transform(X)

df\_pc = pd.DataFrame(pc, columns=columns)

df\_pc.head()



The DataFrame **df\_pc** shows the **explained variance**, which is the information (variance) attributed by each of the principal components. This important that we know how much information contribution can be accumulated by a number of the principal components together. PCA.explained\_variance\_ratio\_ tell us the percentage (importance) of each PC. Notice that **this analysis does not link predictors with the target variable**.

pca.explained\_variance\_ratio\_

Text, letter

Description automatically generated

**Finding:**

**explained\_variance\_ratio\_**tells us that the first principal component contains 9.98% of the variance and the second principal component contains 9.26% of the variance. Together, the two components contain nearly 20% of the information.

### Plot the (variance) of PCA information contribution in a bar chart

# Create a DataFrame for visulisation

pca\_df = pd.DataFrame({'Variance Explained':pca.explained\_variance\_ratio\_,

'PC':df\_pc.columns})

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

plt.title('PCA - explained variance ratio')

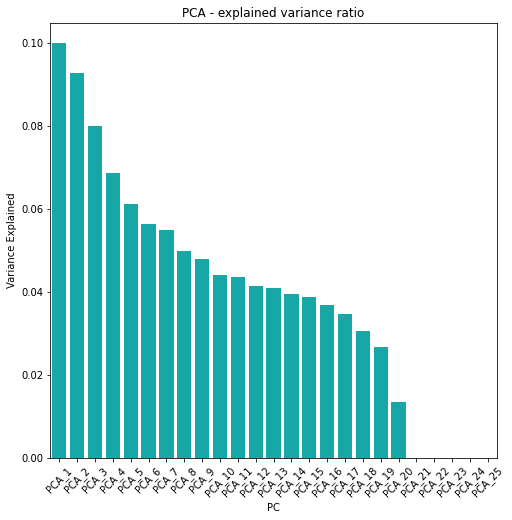
# Plot DataFrame

sns.barplot(x='PC',y='Variance Explained',

data=pca\_df, color="c")

plt.xticks(rotation=45)

plt.show()



### Plot the cumulated (variance) information contribution

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

# Instantiate, fit and transform

var = pca.explained\_variance\_ratio\_

# Plot cumulative variance

cumulative\_var = np.cumsum(var)\*100

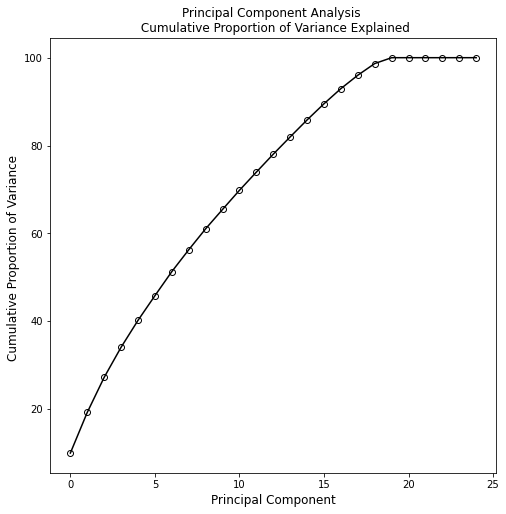
plt.plot(cumulative\_var,'k-o',markerfacecolor='None',markeredgecolor='k')

plt.title('Principal Component Analysis \n Cumulative Proportion of Variance Explained',fontsize=12)

plt.xlabel("Principal Component",fontsize=12)

plt.ylabel("Cumulative Proportion of Variance ",fontsize=12)

plt.show()



**Finding:**

From the plots above, we see that approximately **17 features** are needed to explain 90% of the variance in the dataset. This gives us a good intuition of the number of features required in our model.

For the minimum numbers of predictor to achieve the best performance, the number of predictors are 20. With correlation analysis, we can choice the 20 predictors.

### Assess models' performance after features selection

# Data preparation

X = df\_credit\_T.drop(['Purpose\_car', 'Saving accounts\_moderate', 'Purpose\_furniture/equipment', 'Purpose\_repairs', 'Purpose\_domestic appliances'], axis=1)

y = df\_y

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# to feed the random state

seed = 42

# prepare models

models = []

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

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

models.append(('DT', 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 = []

for name, model in models:

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

pre\_results = model.predict(X\_test)

model\_acc = accuracy\_score(y\_test, pre\_results)

names.append(name)

results.append(model\_acc)

msg = "Modle %s has: %0.3f accuracy" % (name, model\_acc)

print(msg)

#print(names)

#print(results)

# boxplot algorithm comparison

results\_df = pd.DataFrame({'Name':names, 'Results':results})

#print(results\_df)

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

fig.suptitle('Algorithm Comparison')

results\_df.plot.bar(x='Name',rot=45)

plt.show()

Model LR has accuracy of: 0.736

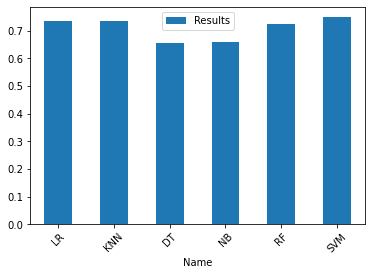
Model KNN has accuracy of: 0.736

Model DT has accuracy of: 0.656

Model NB has accuracy of: 0.660

Model RF has accuracy of: 0.724

Model SVM has accuracy of: 0.748



**Findings:**

We can see after select predictors the noise has been reduced, so all models' performance has been improved.

## 6.2 Models' performance measure using Cross-Validation

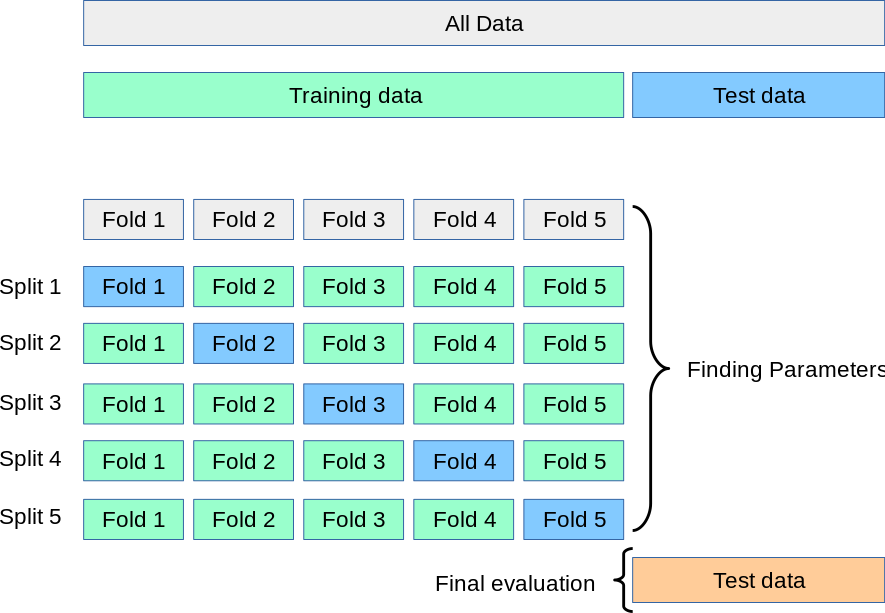
**Cross-validation** (CV) is a technique used in machine learning and prediction model construction to evaluate the performance of a model on new, unseen data. It involves splitting the data into a training set and a validation set. The training set is used to train the model, while the validation set is used to evaluate the performance of the trained model. This process is repeated multiple times, with different subsets of the data used for training and validation each time.

There are several different types of cross-validation techniques, including k-fold cross-validation, leave-one-out cross-validation, and stratified cross-validation. Each of these techniques has its own advantages and disadvantages, and the choice of which technique to use depends on the specific requirements of the problem at hand.

Cross-validation can be used to tune the hyperparameters of the model and provides accurate and reliable estimates of a model's performance. [scikit-learn](https://scikit-learn.org/stable/modules/cross_validation.html) has provided a good introduction on CV.

**K-fold cross-validation** is a commonly used method of cross-validation in machine learning. In k-fold cross-validation, the available data is divided into k equal-sized subsets, or "folds". The model is then trained on k-1 of these folds, and the remaining fold is used as the validation set to evaluate the performance of the model.

This process is repeated k times, with a different fold used for validation each time. The results of each validation can then be averaged to provide an overall estimate of the model's performance.



One advantage of k-fold cross-validation is that it allows for a more accurate estimate of the model's performance than a simple train-test split, as it uses all available data for training and validation. K-fold cross-validation can also be used to tune the hyperparameters of the model, by evaluating the performance of the model for different values of the hyperparameters.

The choice of k in k-fold cross-validation depends on the size of the available data and the computational resources available. Typically, values of k between 5 and 10 are used, as these provide a good balance between bias and variance.

Overall, k-fold cross-validation is a powerful tool in machine learning model construction, as it allows for an accurate estimate of the model's performance and can be used to optimize the hyperparameters of the model for improved performance.

The following is a simple example of using k-fold cross-validation to measure models' performance. Here, we only use accuracy as the measure. CV can use many other measures such as F1-score.

# Data preparation

X = df\_credit\_T.drop(['Purpose\_car', 'Saving accounts\_moderate', 'Purpose\_furniture/equipment', 'Purpose\_repairs', 'Purpose\_domestic appliances'], axis=1)

y = df\_y

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.25, random\_state=42)

# to feed the random state

seed = 42

# prepare models

models = []

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

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

models.append(('DT', 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 = []

validates = []

names = []

scoring = 'accuracy'

for name, model in models:

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

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

results.append(cv\_results)

names.append(name)

msg = "%s accurcy= %0.3f and sd = %0.3f)" % (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()

LR accurcy= 0.732 and sd = 0.042)

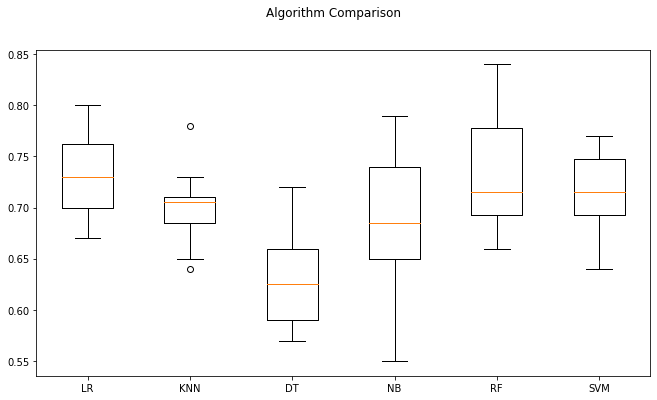
KNN accurcy= 0.701 and sd = 0.038)

DT accurcy= 0.633 and sd = 0.053)

NB accurcy= 0.691 and sd = 0.068)

RF accurcy= 0.735 and sd = 0.056)

SVM accurcy= 0.713 and sd = 0.042)



**Finding:**

* All models' performance has been accurately measured.
* The best performance is achieved when split data set with K=10 fold.
* A good explanation on K-fold CV is in <https://scikit-learn.org/stable/modules/cross_validation.html>

## 6.3 Overfitting detection with K-fold CV

**Overfitting** is common problem in model construction. It occurs when a model is trained to fit the training data too well, which can lead to poor performance on new, unseen data.

Overfitting can be detected using k-fold cross-validation, we can compare the performance of the model on the training set and the validation set. If the model performs significantly better on the training set than on the validation set, it is likely overfitting the training data. In this case, you may need to adjust the hyperparameters of the model, such as regularization strength or number of features, to prevent overfitting.

The following code demonstrates the use of CV for overfitting detection. We can export the taining and the testing accuracy. The training accuracy is the mean accuracy on the X\_train and y\_train, and the test accuracy is the accuracy on X\_test and y\_test. If training sccuracy is significantly higher than testing accuracy, we know the model is overfitted with the training dataset.

from sklearn.model\_selection import cross\_validate

# to feed the random state

seed = 42

# prepare models

models = []

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

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

models.append(('DT', 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 = []

validates = []

names = []

scoring = 'accuracy'

for name, model in models:

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

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

CV\_results = cross\_validate(model, X, y, cv=kfold, scoring=scoring, return\_train\_score=True)

#print(CV\_results.keys())

model\_acc = CV\_results['train\_score']

model\_val = CV\_results['test\_score']

validates.append(model\_val)

names.append(name)

results.append(model\_acc)

#print(CV\_results.keys())

#print(model\_acc)

msg = "Modle %s has: %0.3f train accuracy, %0.3f test accuracy" % (name, model\_acc.mean(), model\_val.mean())

#msg = "Modle %s has: %0.3f test accuracy" % (name, model\_val.mean())

print(msg)

# boxplot algorithm comparison

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

fig.suptitle('Algorithm Comparison')

ax = fig.add\_subplot(111)

plt.boxplot(validates)

ax.set\_xticklabels(names)

plt.show()

Model LR has: 0.757 train accuracy, 0.739 test accuracy

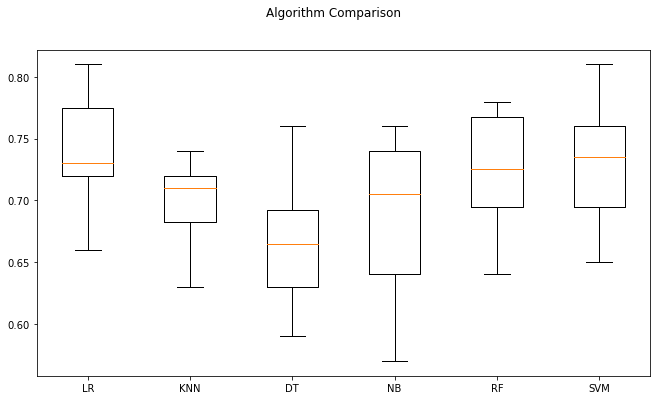
Model KNN has: 0.805 train accuracy, 0.700 test accuracy

Model DT has: 1.000 train accuracy, 0.671 test accuracy

Model NB has: 0.709 train accuracy, 0.689 test accuracy

Model RF has: 1.000 train accuracy, 0.732 test accuracy

Model SVM has: 0.817 train accuracy, 0.729 test accuracy



**Finding:**

We can see that all models have a higher accuracy on the training and a lower accuracy on test. It shows all our models have certain degree of **"overfitting”.**

## ****6.4 Fine tune RF Hyperparameters with K-fold CV****

All models we used with their default parameters. It is a common practice that fine tune hyperparameters for models used in the final prediction. In this tutorial we use **Random Forest (RF)** model as an example to demonstrate the process and technologies used in k-fold CV for model fine tune.

With scikit-learn, tuning a classifier for recall can be achieved in (at least) two main steps.

1. Using GridSearchCV to tune your model by searching for the best hyperparameters and keeping the classifier with the highest recall score.
2. Adjust the decision threshold using the precision-recall curve and the roc curve, which is a more involved method that I will walk through.

Random Forest (RF) Hyperparameters that we’ll be looking at:

* **max\_depth:** The max\_depth of a tree in Random Forest is defined as the longest path between the root node and the leaf node.
* **min\_sample\_split:** default is 2. This means that if any terminal node has more than two observations and is not a pure node, we can split it further into subnodes.
* **max\_leaf\_nodes:** sets a condition on the splitting of the nodes in the tree and hence restricts the growth of the tree. If
* **min\_samples\_leaf:** specifies the minimum number of samples that should be present in the leaf node after splitting a node.
* **n\_estimators:** how many trees should we consider
* **max\_sample (bootstrap sample):** determines what fraction of the original dataset is given to any individual tree.
* **max\_features:** the number of maximum features provided to each tree in a random forest.
* **criterion:** the measures of splitting of a node of a decision tree (Gini, Entropy and Error rate).

Among these Hyperparameters, three categories are covered: Tree size (terminate conditions), sample and features. So the gris is also normally specifying these three.

#### Basic Techniques:

* RandomizedSearchCV
* GridSearchCV

#Seting the Hyper Parameters

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

"n\_estimators":[10,15,50,100],

"max\_features": [10,15, 20]}

#Creating the classifier

model = RandomForestClassifier(random\_state = 2)

grid\_search = GridSearchCV(model, param\_grid = param\_grid, cv=5, scoring='recall', verbose=0)

# we use recall rather than accurate because in our application we care more of correct prediciton rate on positive results.

# verbose shows the msg to display:

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

GridSearchCV(cv=5, estimator=RandomForestClassifier(random\_state=2),

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

'max\_features': [10, 15, 20],

'n\_estimators': [10, 15, 50, 100]},

scoring='recall')

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

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

0.503864734299517

{'max\_depth': None, 'max\_features': 20, 'n\_estimators': 15}

rf = RandomForestClassifier(max\_depth=None, max\_features=20, n\_estimators=15, random\_state=2)

#rf = RandomForestClassifier(max\_depth=None, max\_features=15, n\_estimators=15, random\_state=42)

#rf = RandomForestClassifier(max\_depth=None, max\_features=15, n\_estimators=25, random\_state=42)

#rf = RandomForestClassifier(max\_depth=None, max\_features=15, n\_estimators=500, random\_state=42)

#rf = RandomForestClassifier(max\_depth=3, max\_features=15, n\_estimators=100, random\_state=42)

#rf = RandomForestClassifier(random\_state=42)

#trainning with the best params

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

RandomForestClassifier(max\_features=20, n\_estimators=15, random\_state=2)

Finally, we use suggested hyperparameters to refit our RF model, and use it to make predictions.

We check our prediction with a confusion matrix!

#Testing the model

#Predicting using our model

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

# Verificaar os resultados obtidos

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

print("\n")

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

0.76

[[157 21]

[ 39 33]]

**Finding:** with Grid search we can find the best hyperparameters. Use them we can improve the model's performance.

from **71.6** to **76%**

Can you interpret what is this mean?

结束语

这是一个基于实际风险评估的具体实例完整演示了如何使用大数据分析方法给出答案的全部过程。

这个过程包括：

* 1. 寻找数据源并导入数据：可以从不同的单一或者多个数据源导入数据。
  2. 读数据进行初步观察和理解。 使用描述性数据分析方法（）DDA）或者探索性数据分析（）EDA）方法了解数据的分布，识别异常值和缺失值，并确定要使用哪些特征。
  3. 对数据进行预处理：转换类型， 填充遗失数据，数据特征工程，包括特征选择、特征变换和特征生成等，以提高模型的性能。等
  4. 模型选择和训练：根据问题的特点，选择适当的模型，并使用训练数据进行模型训练。常见的模型包括线性回归、决策树、随机森林、神经网络等。
  5. 模型表现进行评估。
  6. 对已构建的模型进行精细调试，包括训练数据的调整（数量和属性的选择）和模型固有参数的调整，确定模型的性能并进行必要的调整，以提高模型的泛化性能。

在数据分析项目中，各个任务都需要使用不同的工具，例如Python中的Pandas、Numpy、Scikit-learn、Matplotlib、Seaborn等等，这些工具可以帮助我们有效地处理数据和建模。

最后，需要指出的是，没有最好的模型。在模型的训练和调整过程中，需要投入大量的时间和精力，并且可能需要对多个模型进行比较。需要根据具体问题的特点，不断尝试和改进模型，才能取得更好的预测效果。因此，在进行数据分析和预测模型构建时，需要有耐心和恒心，并且不断学习和探索新的技术和方法。

李刚民

2023年2月16日

剑桥, 英国