predicting-credit-risk-model-pipeline

April 1, 2025

Explorations through some German Credit Risk data to understand their patterns.

1 1. Introduction:

Context

In this dataset, each entry represents a person who takes a credit to a bank. Each person is classified as good or bad credit risks according to the set of attributes. All the observation are describe by the following covariates : 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) Risk (Value target - Good or Bad Risk)

2. Libraries: - Importing Librarys - Importing Dataset

[159]:
```
#Load the libraries
import pandas as pd # manage datasrt
import numpy as np # for linear algebra on tab
import seaborn as sns # Graph library that use matplot in background
import matplotlib.pyplot as plt # to plot figures

# it's a library that we work with plotly
import plotly.offline as py
py.init_notebook_mode(connected=True) # this code, allow us to work with␣
 ↪offline plotly version
import plotly.graph_objs as go # it's like "plt" of matplot
import plotly.tools as tls # It's useful to we get some tools of plotly
import warnings # This library will be used to ignore some warnings
from collections import Counter # To do counter of some features
```

[160]:
```
# import our data
df_credit = pd.read_csv("german_credit_data.csv",index_col=0)
```

2 3. Take a first Look:

- we take a look at some first rows of our dataset

- we are Looking for the type of the covariates.
- we check the proportion of Null values.
- check unique values.

[161]:
```
df_credit.head()
```

[161]:
```
   Age     Sex  Job Housing Saving accounts Checking account  Credit amount  \
0   67    male    2     own             NaN           little           1169
1   22  female    2     own          little         moderate           5951
2   49    male    1     own          little              NaN           2096
3   45    male    2    free          little           little           7882
4   53    male    2    free          little           little           4870

   Duration              Purpose  Risk
0         6             radio/TV  good
1        48             radio/TV   bad
2        12            education  good
3        42  furniture/equipment  good
4        24                  car   bad
```

[162]:
```
#Searching for missings,type of data and also known the shape of data
df_credit.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 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
```

By this we can see that we have 1000 observations observed on the covariates mentioned earlier There are no missing values and most of them are categorical even if there are some categorical values take here as integer.

[163]:
```
#Looking unique values
df_credit.nunique()
```

```
[163]: 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
```

By this we can confirm the distinc values of each covariate with the number of distinc equal to the number of label for the categorical covariates. and the hurge number of distinc values for the numerical ones. This also give us an insight on which are the realy numerical columns.

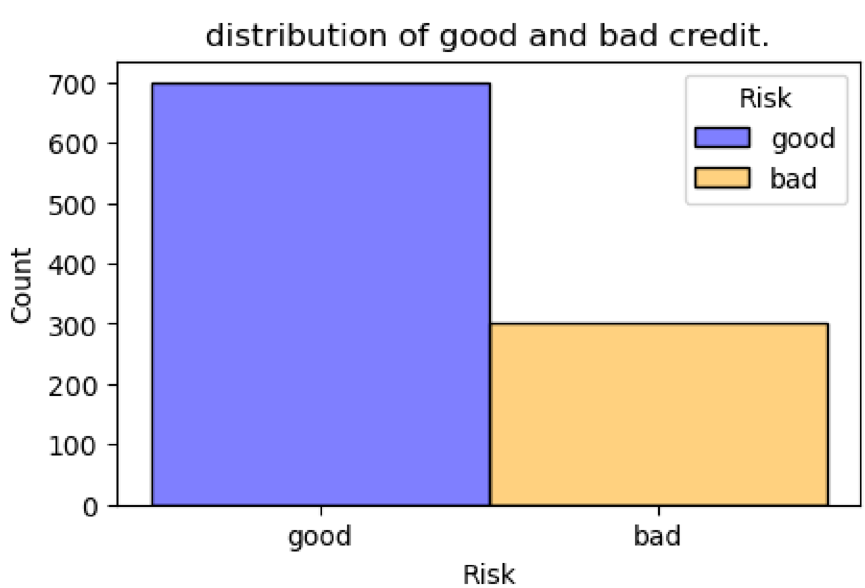
```
[164]: # we handle the fact that the job us considered as an integer column.df
df_credit['Job'] = df_credit["Job"].astype('category')
```

3 4. EDA:

- We check the imbalancenes of the data .
- plot and interpretation of distribution of variables.
- a little bi-variate analysis.

Let's start looking through target variable and their distribuition

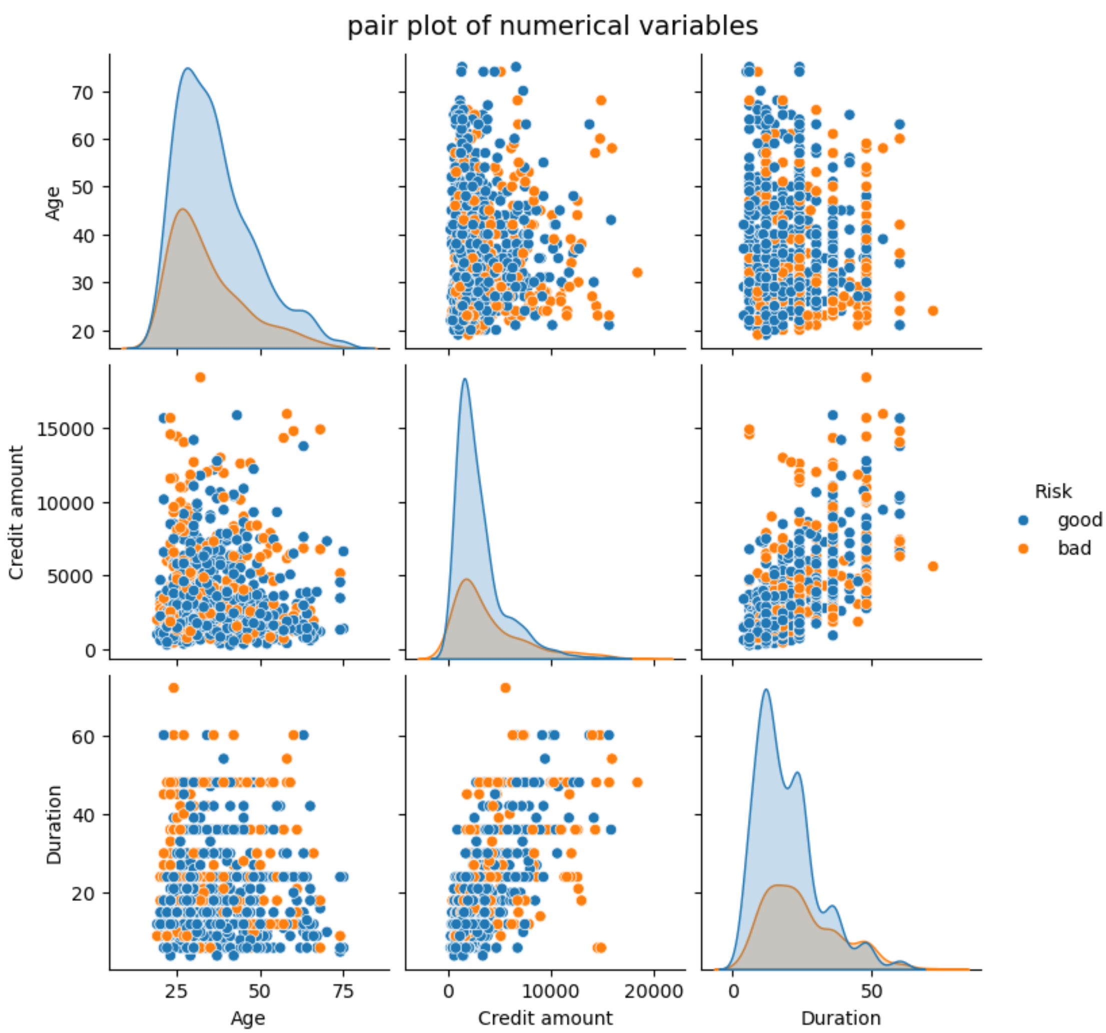
```
[165]: plt.figure(figsize=(5,3))
sns.histplot(data=df_credit, x="Risk", hue="Risk", palette=("blue","orange"))
plt.title("distribution of good and bad credit.")
plt.show()
```



As we can see here the data are imbalance with more good loans than bad ones so we need to implement later some techniques to handle this.

[166]:
```
# first look on the distribution and relations between numerical variables.
plt.figure(figsize=(7,5))
sns.pairplot(df_credit,hue="Risk")
plt.suptitle("pair plot of numerical variables",y=1.02, fontsize=14)
plt.show()
```

<Figure size 700x500 with 0 Axes>



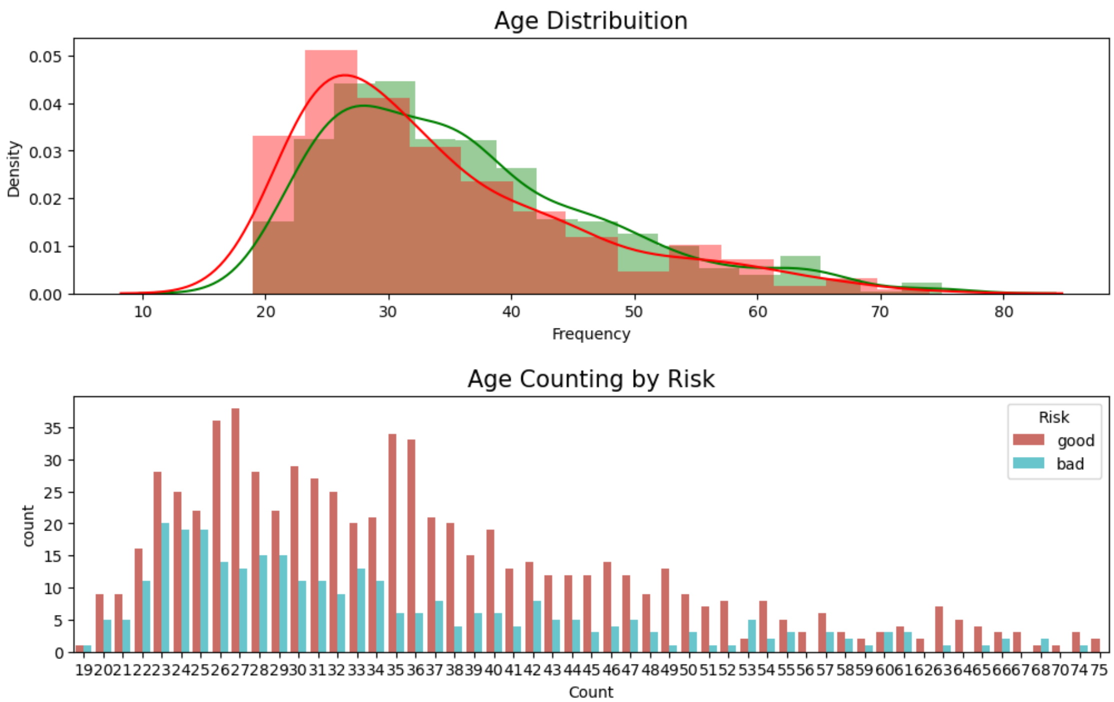
3.0.1 Insights

Here, we observe that age, loan duration, and credit amount are all left-skewed. This suggests that very few people over 50 years old borrow money compared to those under 50. Similarly, compared to those who take a loan under 100K DM, the number of people borrowing more is very low. Additionally, very few people extend their loan duration beyond 50 months, and those who exceed 65 months are almost always bad borrowers. Also we observe a positive relationsheep between the Credit_amount and the Duration which suggest us that people who borrow more tend to get a large deadline which is a good reflect of the reality.

[167]:
```
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()
```

```
/tmp/ipykernel_69562/3165110996.py:7: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


/tmp/ipykernel_69562/3165110996.py:9: UserWarning:
```

```
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```



We categorise the Age

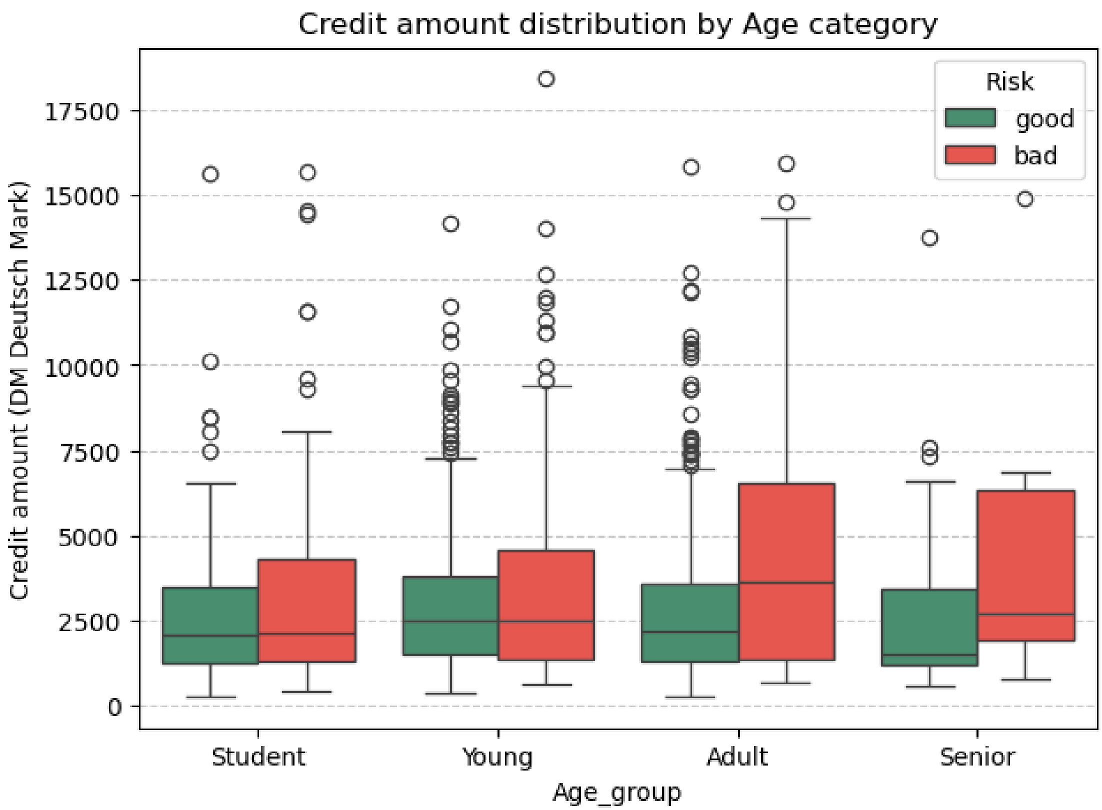
Let us now group the age into differents categories define as:

- [18-25] => Student
- [25-35] => Young
- [35-60] => Adult
- [60-120] => Senior

[168]:
```
# set the intervals
interval = (18, 25, 35, 60, 120)
# set the cathegories
cats = ['Student', 'Young', 'Adult', 'Senior']
# define our new column
df_credit["Age_cat"] = pd.cut(df_credit.Age, interval, labels=cats)
```

[169]:

```
# here we check the bad and good borrower amoung age cthegorie
df_good = df_credit[df_credit["Risk"] == 'good']
df_bad = df_credit[df_credit["Risk"] == 'bad']
df_combined = pd.concat([df_good, df_bad])

# Create the boxplot
plt.figure(figsize=(7, 5))
sns.boxplot(
    x="Age_cat",
    y="Credit amount",
    hue="Risk",
    data=df_combined,
    palette={"#3D9970", "#FF4136"}
)

# Set labels and title
plt.xlabel("Age_group")
plt.ylabel("Credit amount (DM Deutsch Mark)")
plt.title("Credit amount distribution by Age category")

# Display the plot
plt.legend(title="Risk")
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```



Interesting distribuition

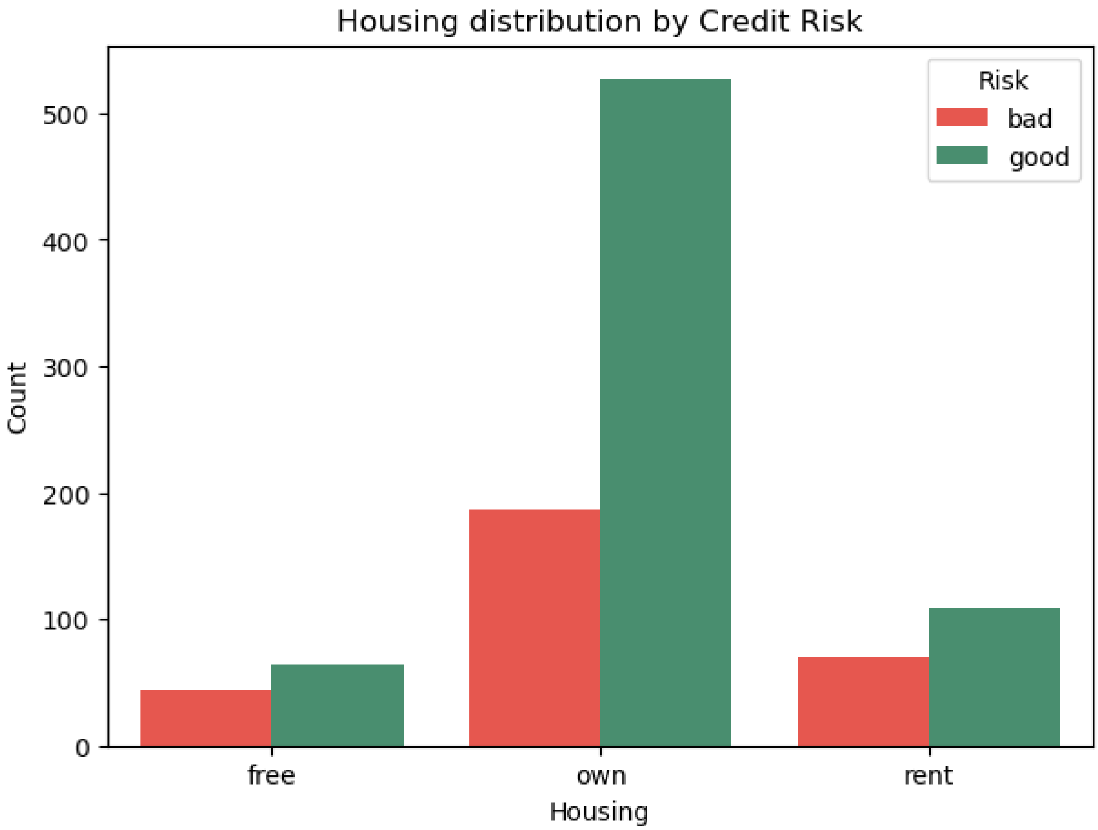
3.0.2 Insights

Here we can see that the proportion of good credit is allmost the same accross the diferent age group. The porportion of bad credit is verry high for the Adult and the senior as compare to the good credit in these categories in all the categories there are more good borrower than bad borrower who borrow more than 70000 DM.
This suggest us many people start care less about their loan over years so we should be more careful with those ones which want to borrow hurge amount of money.

3.0.3 Distribuition of Housing own and rent by Risk

[170]:
```
# Count the occurrences of each Housing category per Risk group
df_housing_counts = df_credit.groupby(["Housing", "Risk"]).size().
↪reset_index(name="Count")

# Create the bar plot
plt.figure(figsize=(7, 5))
sns.barplot(
    x="Housing",
    y="Count",
```

```python
    hue="Risk",
    data=df_housing_counts,
    palette={"good": "#3D9970", "bad": "#FF4136"}
)

# Set labels and title
plt.xlabel("Housing")
plt.ylabel("Count")
plt.title("Housing distribution by Credit Risk")

# Display the legend
plt.legend(title="Risk")

# Show the plot
plt.show()
```



we can see that the own and good risk have a high correlation the majority of borrower are owner the majority of bad credit come also drom house owner as compare to the proportion of the bad credit in the other cathegories.

3.0.4 Now we take a look at the distribution of bad and good credit amoung sex

[171]:
```
# here we try ploting with plotly
#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')
```

```
/home/student/miniconda3/lib/python3.12/site-packages/plotly/tools.py:455:
```

```
DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use
plotly.subplots.make_subplots instead
```

for the Sex distribution (Left Chart - Bar Plot): - Males are significantly more represented in the dataset compared to females. - More males have good credit than bad credit. - Similarly, more females have good credit, but their total numbers are lower than males.

for the Credit amount by Sex (Right Chart - Box Plot) - The green box plots represent good credit, while the purple box plots represent bad credit. - The median credit amount is higher for those with bad credit compared to those with good credit. - The spread (IQR) of credit amounts is larger for individuals with bad credit, meaning there is more variability in credit amounts. - There are outliers (dots above the whiskers), indicating that some individuals have exceptionally high credit amounts.

3.0.5 Insights

The dataset has a gender imbalance, with more male applicants than female applicants.Regardless of gender, a higher number of people in the dataset have good credit compared to bad credit. since the dataset is also imbalance reagrding the Risk covariate, also People with bad credit tend to take larger credit amounts compared to those with good credit. The higher variability in bad credit cases suggests that some individuals may have taken excessive loans, leading to credit risk. Both males and females follow the same trend, but since more males are in the dataset, the pattern is clearer for them.

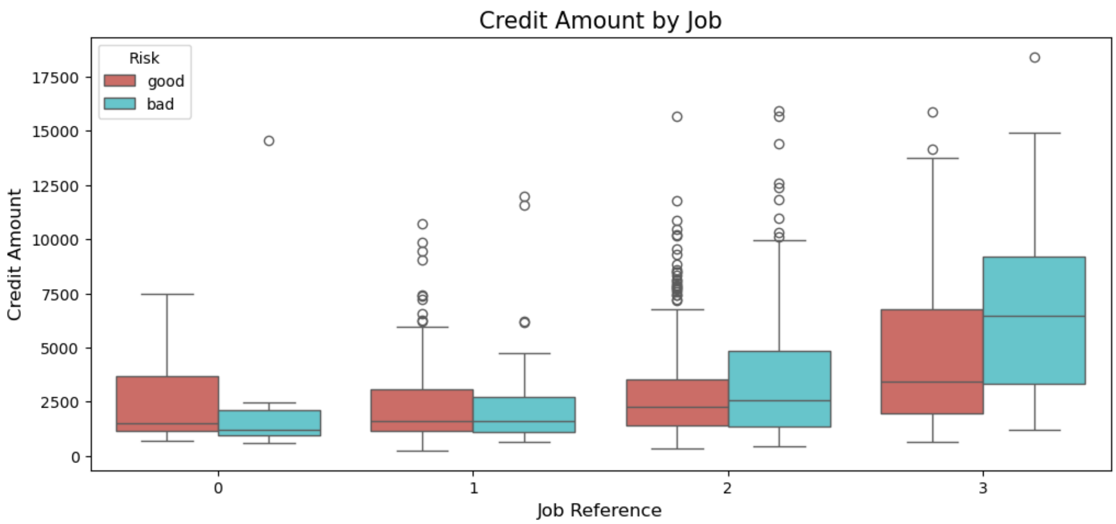
Now we do some explorations through the Job - Distribuition - Crossed by Credit amount

[172]:
```python
#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')
```

[173]:
```
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')
```

[174]:
```
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()
```



Credit amount by Job (Top Box Plot): - The x-axis represents different Job References (0 - unskilled and non-resident, 1 - unskilled and resident, 2 - skilled, 3 - highly skilled). - The y-axis represents the Credit Amount. - The red boxes represent individuals with good credit, and the blue boxes represent individuals with bad credit. - Unskilled and non-resident : tend to have the lowest credit amounts, and those with good credit have slightly higher median credit amounts than those with bad credit. - (Unskilled and resident) and skilled : show a more balanced credit distribution between good and bad credit holders, with a small increase in credit amounts. - highly skilled : has the highest median and most dispersed credit amounts, with bad credit holders having a higher spread of credit amounts than good credit holders. - Across all job types, bad credit holders show more variability and higher outliers, suggesting some individuals take very large loans but struggle to repay them.

for the job type Reference x Age (Bottom Violin Plot): - The red violin plots represent individuals with good credit, and the blue violin plots represent individuals with bad credit. - The width of the violin plot shows the density of individuals at a given age. - For the Unskilled and non-resident (reference 0) : - The age distribution is wider, with a significant number of younger individuals. - Bad credit cases are more frequent among younger individuals. - for the (Unskilled and resident) and skilled workers Reference to 1 & 2: - The age distribution is more concentrated, meaning these job types attract people of within [25-45]years olds. - Both good and bad credit are evenly spread.

- In the high skilled group:
  - The distribution of bad credit cases is spread across different ages, meaning that age is not necessarily a strong predictor of bad credit for this job type.

3.0.6 Insights

Younger individuals (especially in Job 0) tend to have more bad credit cases, this could be due to lack of financial experience or unstable income.those with more stable job types (Job 1 & 2) show balanced credit risk across ages and finaly for Job 3, both younger and older individuals are at risk, suggesting that income stability or financial habits may be more important factors than age alone. The more people are skilled the more they tend to have higher credit amounts, but they also have more bad credit cases. Which suggest us that higher-income jobs come with higher borrowing but also greater credit risk. So lenders may need to closely assess creditworthiness for higher job levels.

3.0.7 we look at the distribuition of Credit Amont

[ ]:

```

```

[175]:

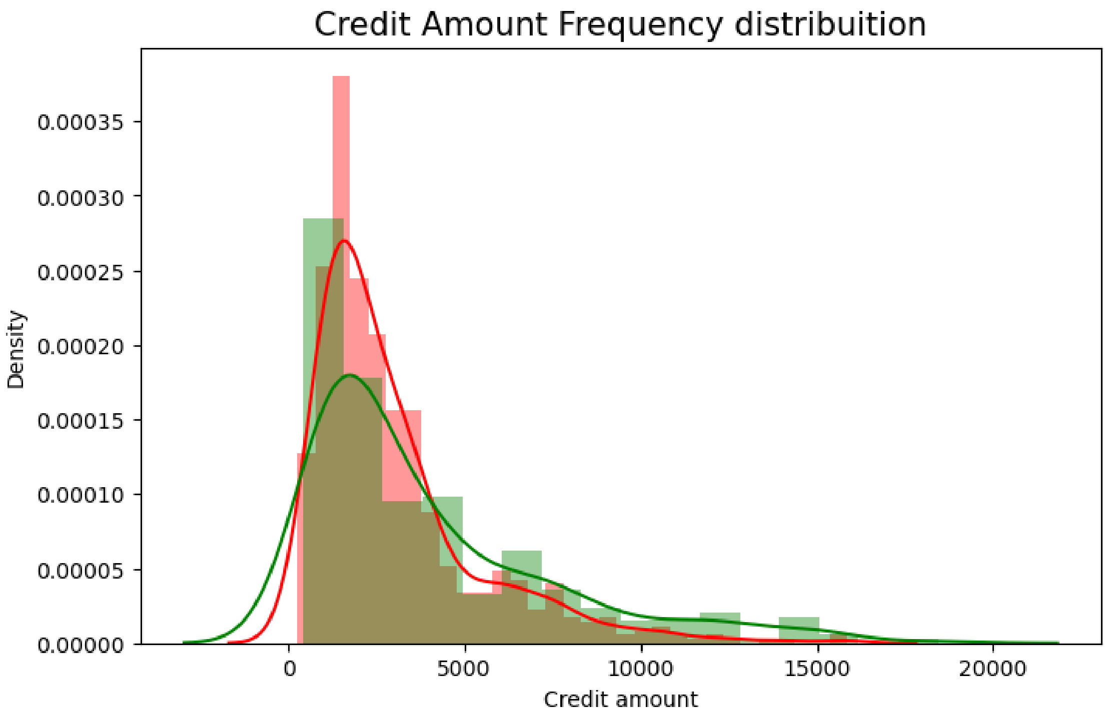
```
import plotly.figure_factory as ff

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')
```

[176]:

```
#Ploting the good and bad dataframes in distplot
plt.figure(figsize = (8,5))

g= sns.distplot(df_good['Credit amount'], color='r')
```

```python
g = sns.distplot(df_bad["Credit amount"], color='g')
g.set_title("Credit Amount Frequency distribuition", fontsize=15)
plt.show()
```

```
/tmp/ipykernel_69562/98572243.py:4: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


/tmp/ipykernel_69562/98572243.py:5: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```



Distruibution of Saving accounts by Risk

[177]:
```
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(
```

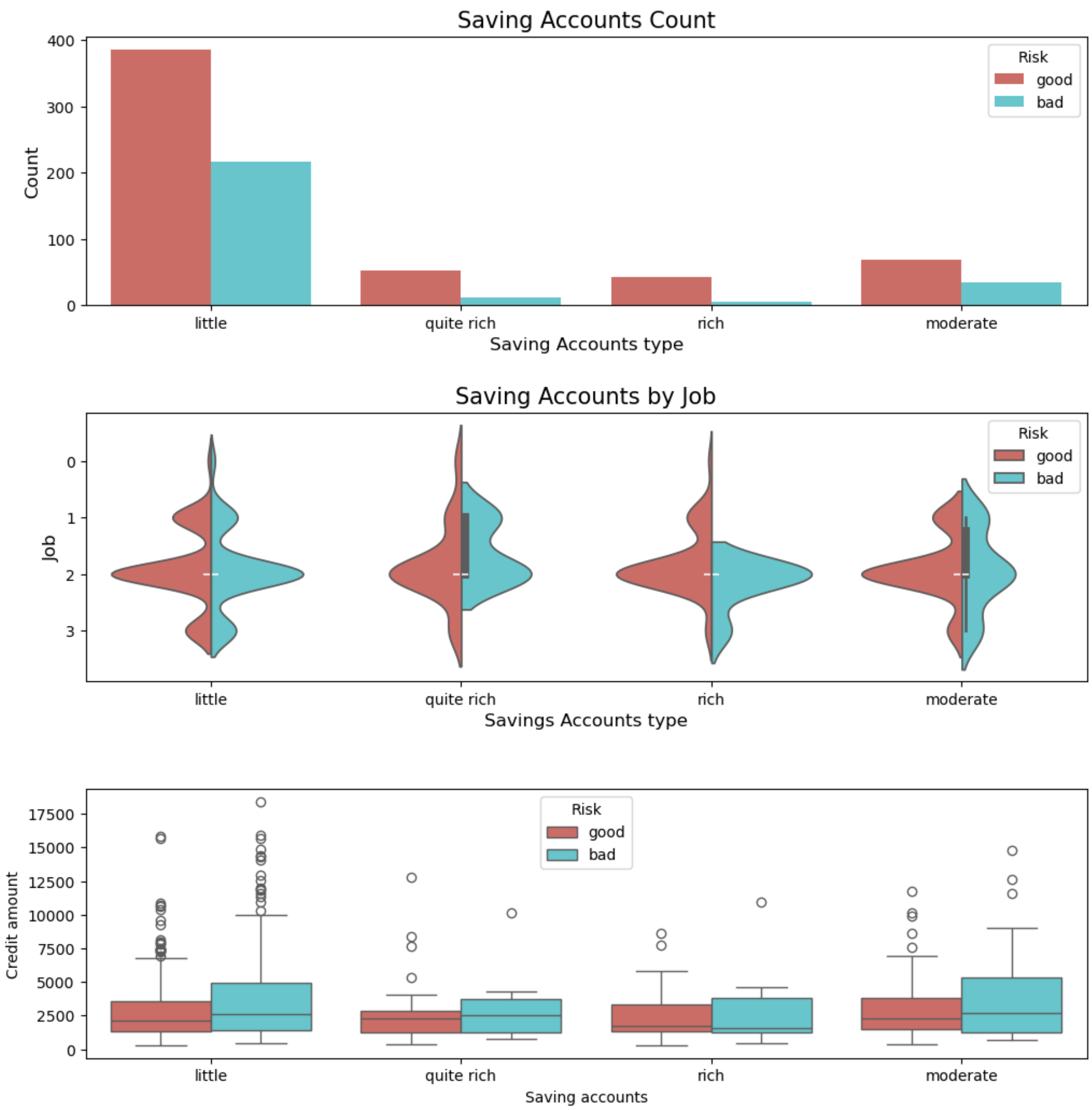
```
    x=df_bad["Saving accounts"],
    y=df_bad["Credit amount"],
    name='Bad credit'
)

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')
```

```
/home/student/miniconda3/lib/python3.12/site-packages/plotly/tools.py:455:
DeprecationWarning:

plotly.tools.make_subplots is deprecated, please use
plotly.subplots.make_subplots instead
```

How can I better configure the legends? I am trying to substitute the graph below, so how can I use the violinplot on subplots of plotly?

[178]:
```
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()
```

```
Description of Distribuition Saving accounts by Risk:
Risk             bad  good
Saving accounts
little           217   386
moderate          34    69
quite rich        11    52
rich               6    42
```



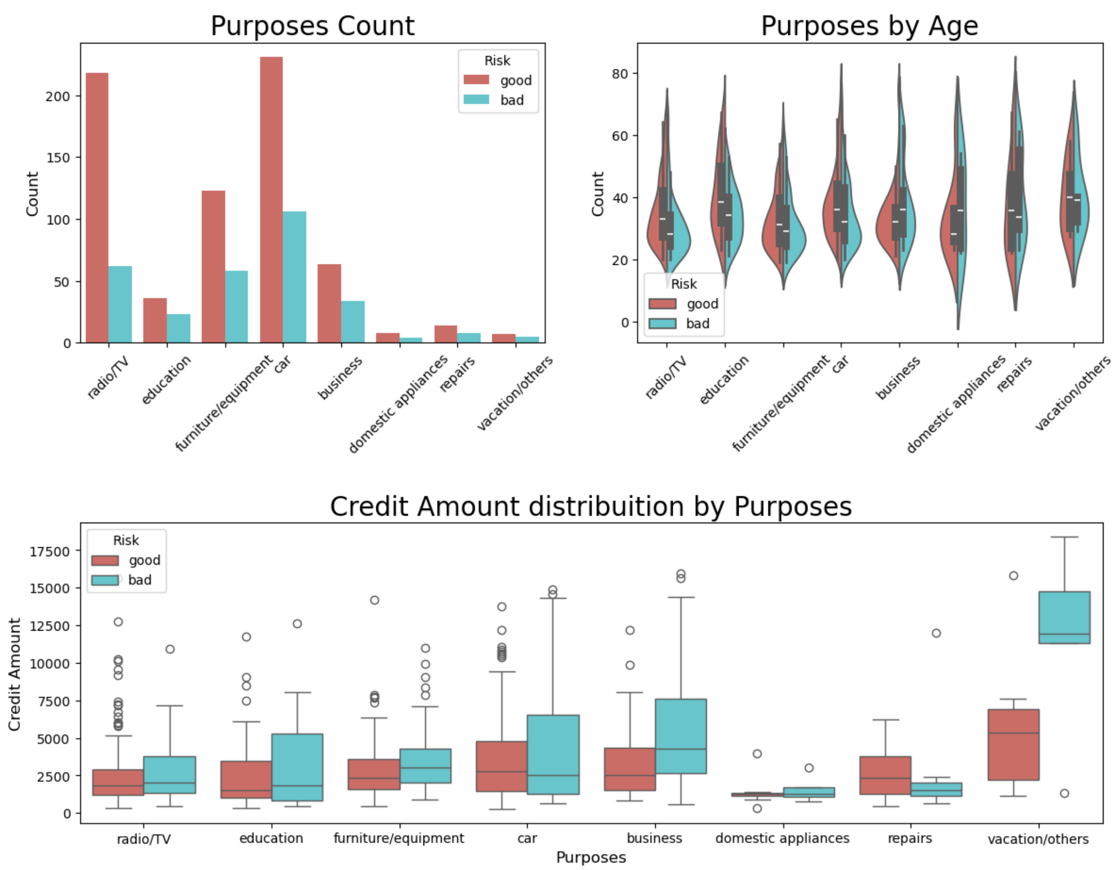
Pretty and interesting distribution...

[179]:
```
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)
```

```python
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()
```

```
Values describe:
Risk                 bad  good
Purpose
business              34    63
car                  106   231
domestic appliances    4     8
education             23    36
furniture/equipment   58   123
radio/TV              62   218
repairs                8    14
vacation/others        5     7
```

```
/tmp/ipykernel_69562/3179296861.py:9: UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.

/tmp/ipykernel_69562/3179296861.py:17: UserWarning:

set_ticklabels() should only be used with a fixed number of ticks, i.e. after
set_ticks() or using a FixedLocator.
```

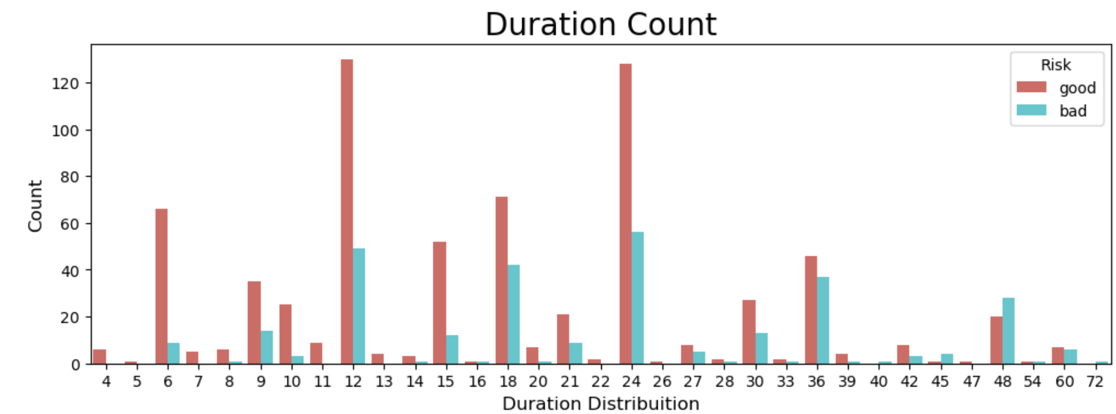


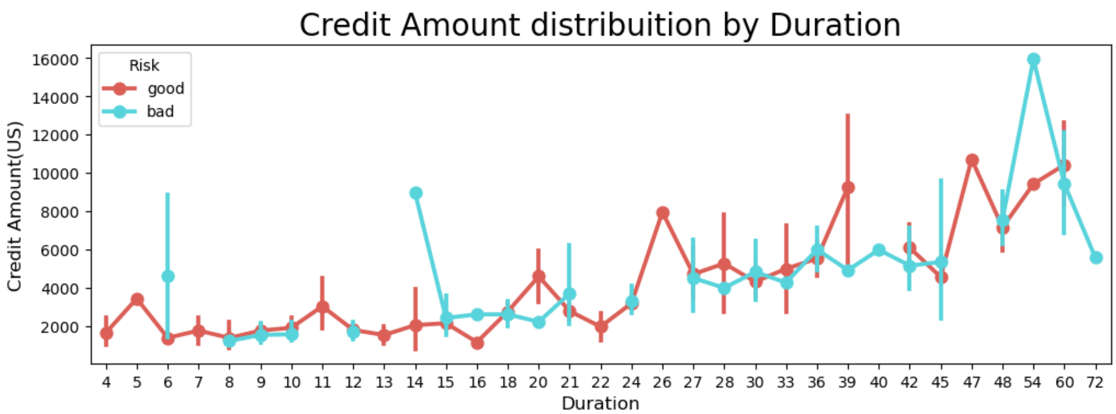
Duration of the loans distribuition and density

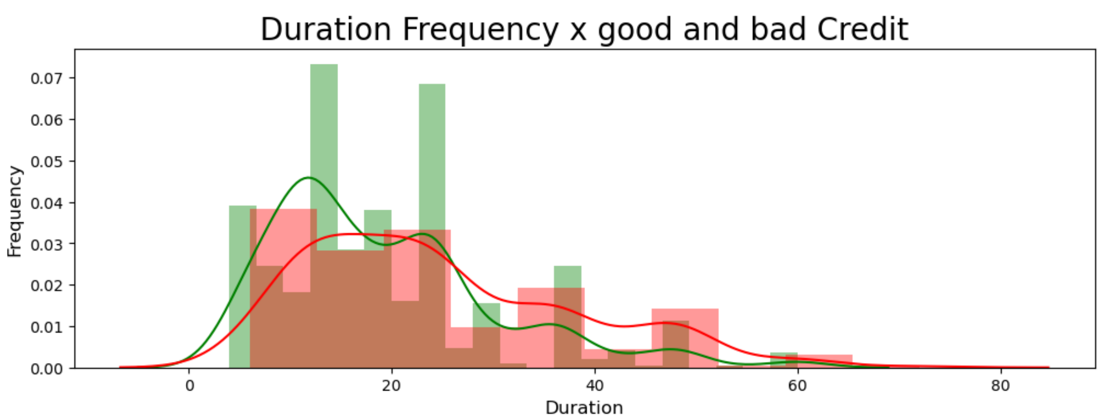
[180]:
```
plt.figure(figsize = (12,14))

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()
```

```
/tmp/ipykernel_69562/3076791316.py:18: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751


/tmp/ipykernel_69562/3076791316.py:19: UserWarning:



`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751
```







Interesting, we can see that the highest duration have the high amounts. The highest density is between [12 ~ 18 ~ 24] months It all make sense.

Checking Account variable

First, let's look the distribuition

[181]:
```
#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)
```

Now, we will verify the values through Checking Accounts

[182]:
```
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')
```

The old plot that I am trying to substitute with interactive plots

[183]:
```
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()
```

```
Total values of the most missing variable:
Checking account
little      274
moderate    269
rich         63
Name: Checking account, dtype: int64
```

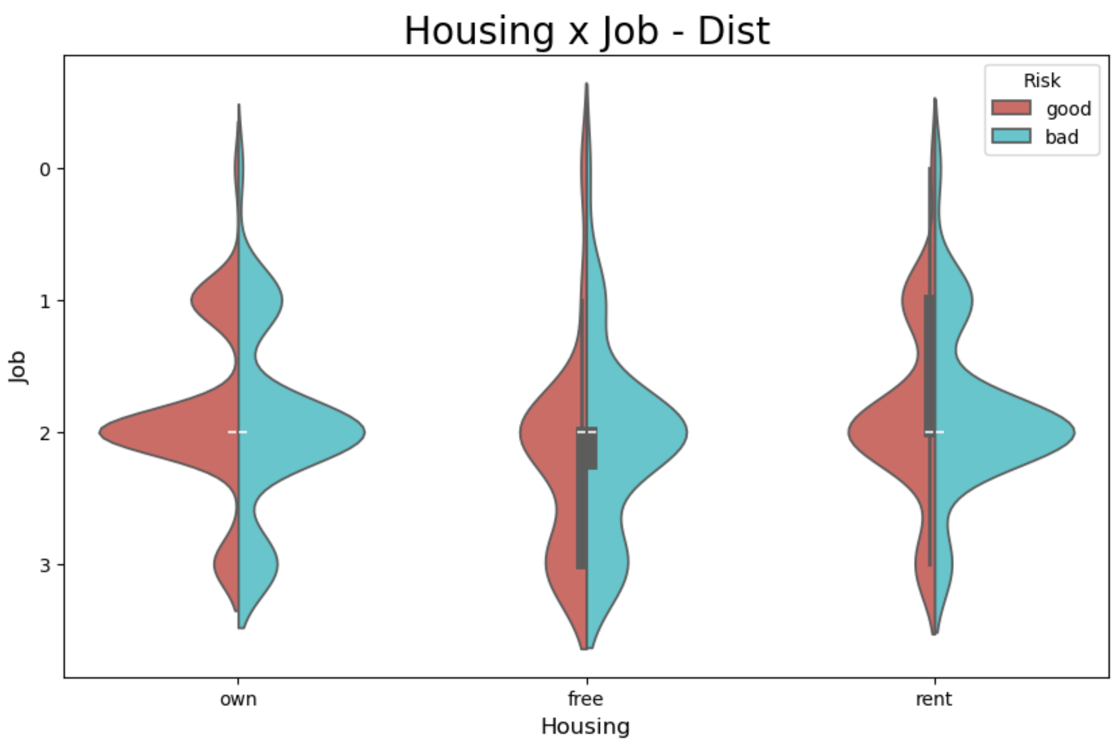
Crosstab session and anothers to explore our data by another metrics a little deep

[184]:
```
print(pd.crosstab(df_credit.Sex, df_credit.Job))
```

```
Job     0    1    2    3
Sex
```

```
female  12   64  197   37
male    10  136  433  111
```

[185]:
```
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()
```



[186]:
```
print(pd.crosstab(df_credit["Checking account"],df_credit.Sex))
```

```
Sex               female  male
Checking account
little                88   186
moderate              86   183
rich                  20    43
```

[187]:
```
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)
```

[187]: <pandas.io.formats.style.Styler at 0x7fb90da0e0f0>

3.1 Looking the total of values in each categorical feature

[188]:
```
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())
```

```
Purpose :  ['radio/TV' 'education' 'furniture/equipment' 'car' 'business'
 'domestic appliances' 'repairs' 'vacation/others']
Sex :  ['male' 'female']
Housing :  ['own' 'free' 'rent']
Saving accounts :  [nan 'little' 'quite rich' 'rich' 'moderate']
Risk :  ['good' 'bad']
Checking account :  ['little' 'moderate' nan 'rich']
Aget_cat :  ['Senior', 'Student', 'Adult', 'Young']
Categories (4, object): ['Student' < 'Young' < 'Adult' < 'Senior']
```

3.2 Let's do some feature engineering on this values and create variable Dummies of the values

[189]:
```
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
```

3.3 Transforming the data into Dummy variables

[190]:
```
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)
```

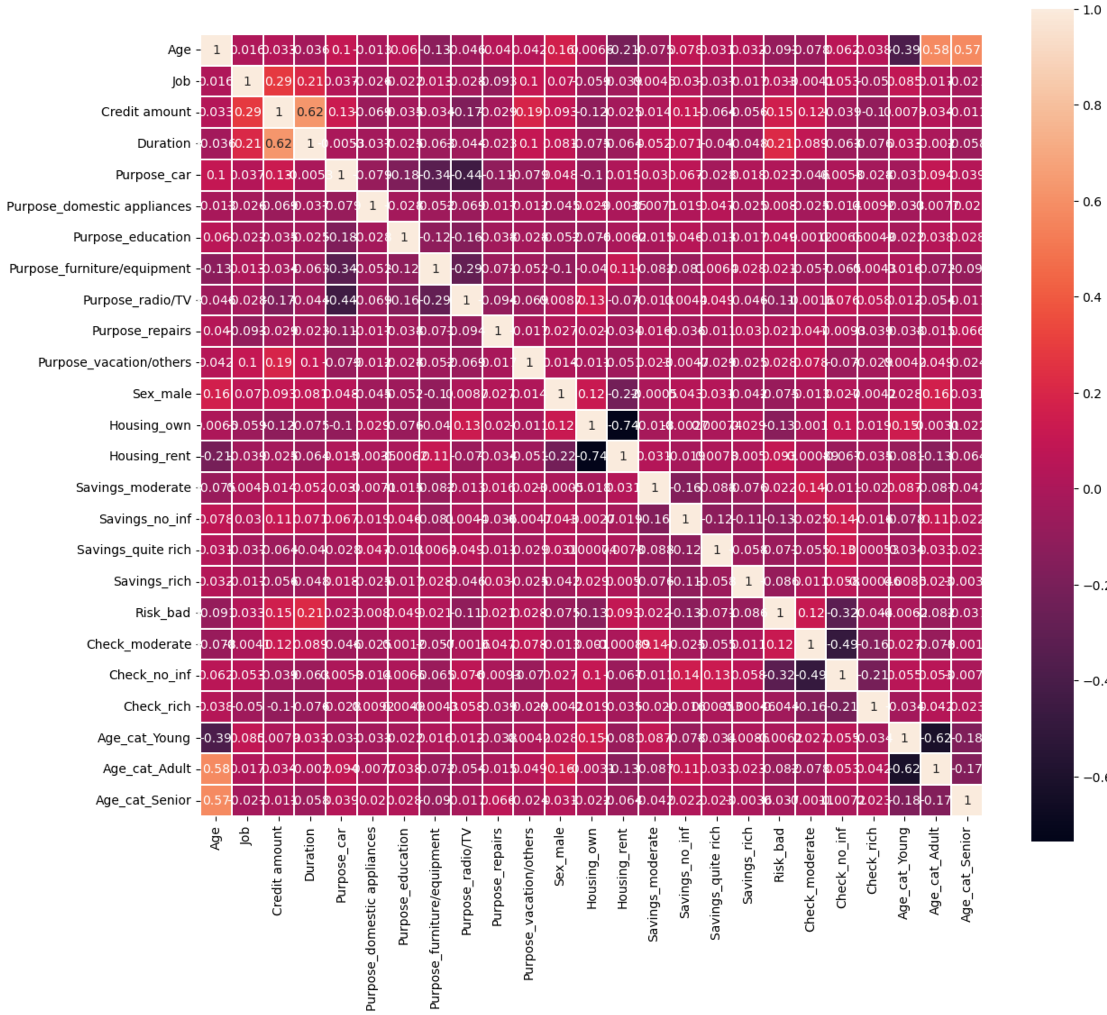
3.4 Deleting the old features

[191]:
```
#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']
```

4 5. Correlation:

Looking the correlation of the data

[192]:
```
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()
```



5 6. Preprocessing:

- Importing ML librarys
- Setting X and y variables to the prediction
- Splitting Data

[193]:
```
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
```

```python
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
```

[194]:

```python
df_credit['Credit amount'] = np.log(df_credit['Credit amount'])
```

[195]:

```python
#Creating the X and y variables
X = df_credit.drop(columns='Risk_bad').values
y = df_credit["Risk_bad"].values

# Spliting X and y into train and test version
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25,␣
 ↪random_state=42)
```

[211]:

```python
# Import necessary libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier  # Ensure XGBoost is installed

# Set random seed
seed = 7

# Define models
models = [
    ('LR', LogisticRegression()),
    ('LDA', LinearDiscriminantAnalysis()),
    ('KNN', KNeighborsClassifier()),
    ('CART', DecisionTreeClassifier()),
    ('NB', GaussianNB()),
    ('RF', RandomForestClassifier()),
    ('SVM', SVC(gamma='auto'))
]
```

```python
# Try adding XGBClassifier only if XGBoost is installed correctly
try:
    models.append(('XGB', XGBClassifier(use_label_encoder=False,␣
 ↪eval_metric='logloss')))
except Exception as e:
    print("Error with XGBClassifier:", e)

# Initialize lists for results
results = []
names = []
scoring = 'f1'  # Updated to F1-score

# Evaluate each model
for name, model in models:
    print(f"Processing model: {name}")
    kfold = KFold(n_splits=10, shuffle=True, random_state=seed)

    try:
        cv_results = cross_val_score(model, X_train, y_train, cv=kfold,␣
 ↪scoring=scoring)
        results.append(cv_results)
        names.append(name)
        print(f"{name}: {cv_results.mean():.4f} ({cv_results.std():.4f})")
    except Exception as e:
        print(f"Error evaluating {name}: {e}")

# Boxplot of algorithm comparison
plt.figure(figsize=(11, 6))
plt.suptitle('Algorithm Comparison (F1-Score)')
sns.boxplot(data=results)
plt.xticks(ticks=np.arange(len(names)), labels=names)
plt.ylabel("F1-Score")  # Updated label
plt.show()
```

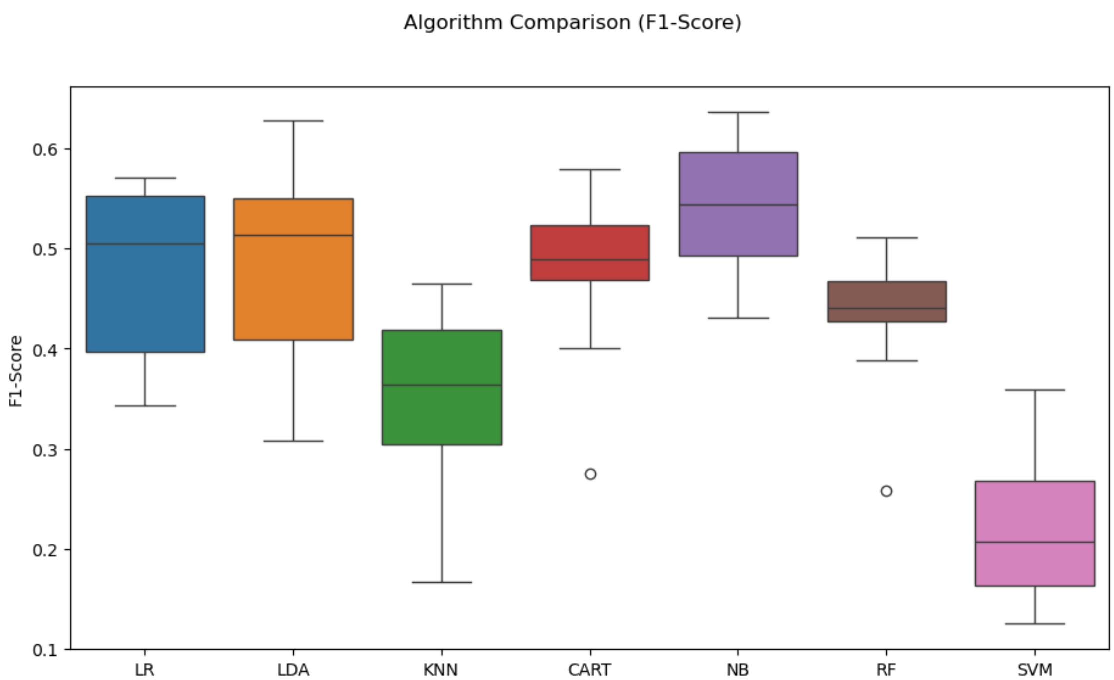
```
Processing model: LR
```

```
/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression

/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
```

```
/home/student/miniconda3/lib/python3.12/site-
packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning:

lbfgs failed to converge (status=1):
STOP: TOTAL NO. OF ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression


LR: 0.4786 (0.0862)
Processing model: LDA
LDA: 0.4862 (0.0996)
Processing model: KNN
KNN: 0.3444 (0.0950)
Processing model: CART
CART: 0.4772 (0.0831)
Processing model: NB
NB: 0.5418 (0.0646)
Processing model: RF
RF: 0.4326 (0.0674)
Processing model: SVM
SVM: 0.2194 (0.0774)
Processing model: XGB
Error evaluating XGB: 'super' object has no attribute '__sklearn_tags__'
```



5.1 Key Observations:

- **Naïve Bayes (NB)** has the **highest median f1-score** and the **smallest variability**, making it the most consistent performer.
- **LDA and LR** follows closely, with a high f1-score but slightly larger variability.

Very interesting. Almost all models shows a low value to f1-score.

We can observe that our best results was with LDA, NB and LR. I will implement some models and try to do a simple Tunning on them

6 7.1 Model 1 :

- Using Random Forest to predictict the credit score
- Some of Validation Parameters

[ ]:
```
#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)
```

[218]:
```
print(grid_search.best_score_)
print(grid_search.best_params_)
```

```
0.508792270531401
{'max_depth': None, 'max_features': 20, 'n_estimators': 3}
```

[219]:
```
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)
```

[219]:
```
RandomForestClassifier(max_features=10, n_estimators=15, random_state=2)
```

[224]:
```
#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_))
print("\n")
print(fbeta_score(y_test, y_pred_, beta=2))
print("\n")
print(classification_report(y_test, y_pred_))
```

```
0.736


[[158  20]
 [ 46  26]]


0.38922155688622756


              precision    recall  f1-score   support

       False       0.77      0.89      0.83       178
        True       0.57      0.36      0.44        72

    accuracy                           0.74       250
   macro avg       0.67      0.62      0.63       250
weighted avg       0.71      0.74      0.72       250
```

Very sucks results! How can I increase my model?

7 7.2 Model 2:

[221]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import GridSearchCV

# Define the hyperparameter grid
param_grid = {
    "var_smoothing": [1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3]  # Smoothing␣
 ↪parameter for GaussianNB
}

# Create the classifier
model = GaussianNB()

# Perform GridSearchCV
```

```python
grid_search = GridSearchCV(model, param_grid=param_grid, cv=5,␣
 ↪scoring='recall', verbose=4)
grid_search.fit(X_train, y_train)

# Print best parameters
print("Best parameters:", grid_search.best_params_)
print("Best recall score:", grid_search.best_score_)
```

```
Fitting 5 folds for each of 7 candidates, totalling 35 fits
[CV 1/5] END …var_smoothing=1e-09;, score=0.644 total time=   0.0s
[CV 2/5] END …var_smoothing=1e-09;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=1e-09;, score=0.696 total time=   0.0s
[CV 4/5] END …var_smoothing=1e-09;, score=0.717 total time=   0.0s
[CV 5/5] END …var_smoothing=1e-09;, score=0.565 total time=   0.0s
[CV 1/5] END …var_smoothing=1e-08;, score=0.644 total time=   0.0s
[CV 2/5] END …var_smoothing=1e-08;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=1e-08;, score=0.696 total time=   0.0s
[CV 4/5] END …var_smoothing=1e-08;, score=0.717 total time=   0.0s
[CV 5/5] END …var_smoothing=1e-08;, score=0.565 total time=   0.0s
[CV 1/5] END …var_smoothing=1e-07;, score=0.644 total time=   0.0s
[CV 2/5] END …var_smoothing=1e-07;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=1e-07;, score=0.696 total time=   0.0s
[CV 4/5] END …var_smoothing=1e-07;, score=0.717 total time=   0.0s
[CV 5/5] END …var_smoothing=1e-07;, score=0.565 total time=   0.0s
[CV 1/5] END …var_smoothing=1e-06;, score=0.644 total time=   0.0s
[CV 2/5] END …var_smoothing=1e-06;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=1e-06;, score=0.696 total time=   0.0s
[CV 4/5] END …var_smoothing=1e-06;, score=0.717 total time=   0.0s
[CV 5/5] END …var_smoothing=1e-06;, score=0.565 total time=   0.0s
[CV 1/5] END …var_smoothing=1e-05;, score=0.644 total time=   0.0s
[CV 2/5] END …var_smoothing=1e-05;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=1e-05;, score=0.696 total time=   0.0s
[CV 4/5] END …var_smoothing=1e-05;, score=0.717 total time=   0.0s
[CV 5/5] END …var_smoothing=1e-05;, score=0.565 total time=   0.0s
[CV 1/5] END …var_smoothing=0.0001;, score=0.578 total time=   0.0s
[CV 2/5] END …var_smoothing=0.0001;, score=0.533 total time=   0.0s
[CV 3/5] END …var_smoothing=0.0001;, score=0.674 total time=   0.0s
[CV 4/5] END …var_smoothing=0.0001;, score=0.674 total time=   0.0s
[CV 5/5] END …var_smoothing=0.0001;, score=0.587 total time=   0.0s
[CV 1/5] END …var_smoothing=0.001;, score=0.444 total time=   0.0s
[CV 2/5] END …var_smoothing=0.001;, score=0.400 total time=   0.0s
[CV 3/5] END …var_smoothing=0.001;, score=0.565 total time=   0.0s
[CV 4/5] END …var_smoothing=0.001;, score=0.522 total time=   0.0s
[CV 5/5] END …var_smoothing=0.001;, score=0.413 total time=   0.0s
Best parameters: {'var_smoothing': 1e-09}
Best recall score: 0.6312077294685989
```

[ ]:

[222]:
```python
# Fitting with train data
model = model.fit(X_train, y_train)

# Printing the Training Score
print("Training score data: ")
print(model.score(X_train, y_train))
```

```
Training score data:
0.7053333333333334
```

[223]:
```python
y_pred = model.predict(X_test)

print(accuracy_score(y_test,y_pred))
print("\n")
print(confusion_matrix(y_test, y_pred))
print("\n")
print(classification_report(y_test, y_pred))
```

```
0.648


[[124  54]
 [ 34  38]]


              precision    recall  f1-score   support

       False       0.78      0.70      0.74       178
        True       0.41      0.53      0.46        72

    accuracy                           0.65       250
   macro avg       0.60      0.61      0.60       250
weighted avg       0.68      0.65      0.66       250
```

With the Gaussian Model we got a lower f1-score.

7.1 Let's verify the ROC curve

[228]:
```python
from sklearn.metrics import roc_curve
import matplotlib.pyplot as plt

# Predict probabilities
y_pred_prob = model.predict_proba(X_test)[:, 1]  # For first model
rf_y_pred_prob = rf.predict_proba(X_test)[:, 1]  # For RF model

# Generate ROC curve values
```

```
fpr, tpr, _ = roc_curve(y_test, y_pred_prob)  # First model
fpr_rf, tpr_rf, _ = roc_curve(y_test, rf_y_pred_prob)  # Random Forest

# Plot both ROC curves
plt.figure(figsize=(8, 6))
plt.plot([0, 1], [0, 1], 'k--', label="Random Guess")  # Diagonal line
plt.plot(fpr, tpr, label="NB ROC Curve")  # First model
plt.plot(fpr_rf, tpr_rf, label="RF ROC Curve")  # Random Forest model

# Labels and title
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve Comparison')
plt.legend()
plt.show()
```