Data Science and Business Analytics

Practice Project II

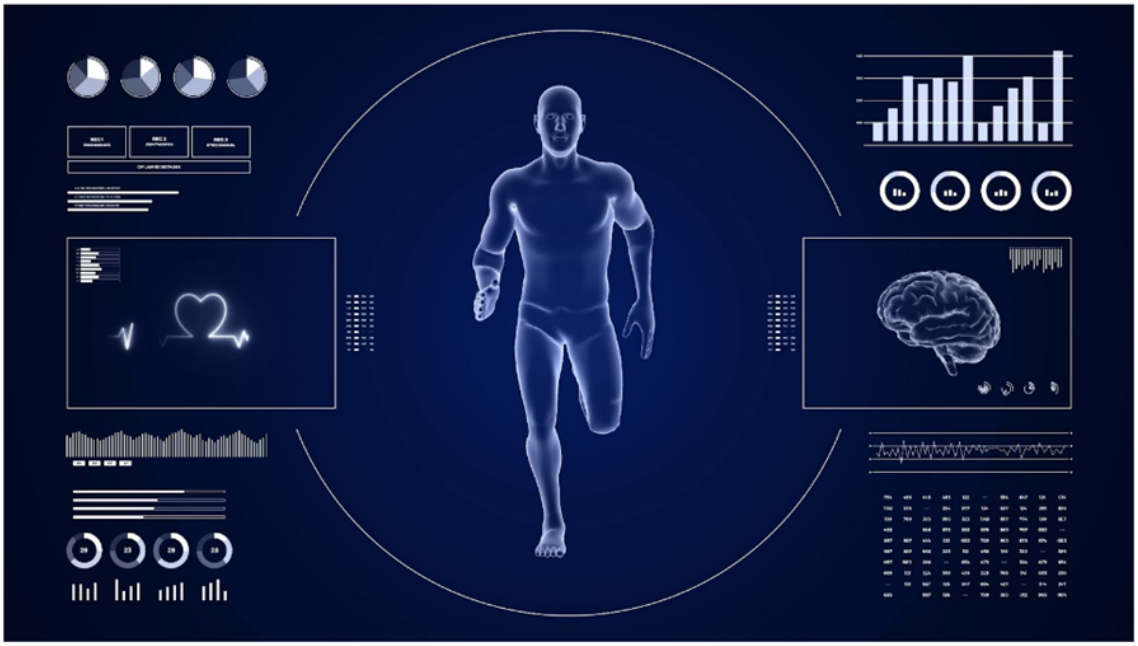
Prediction of Loans Default Based on Applicants Credit History

By

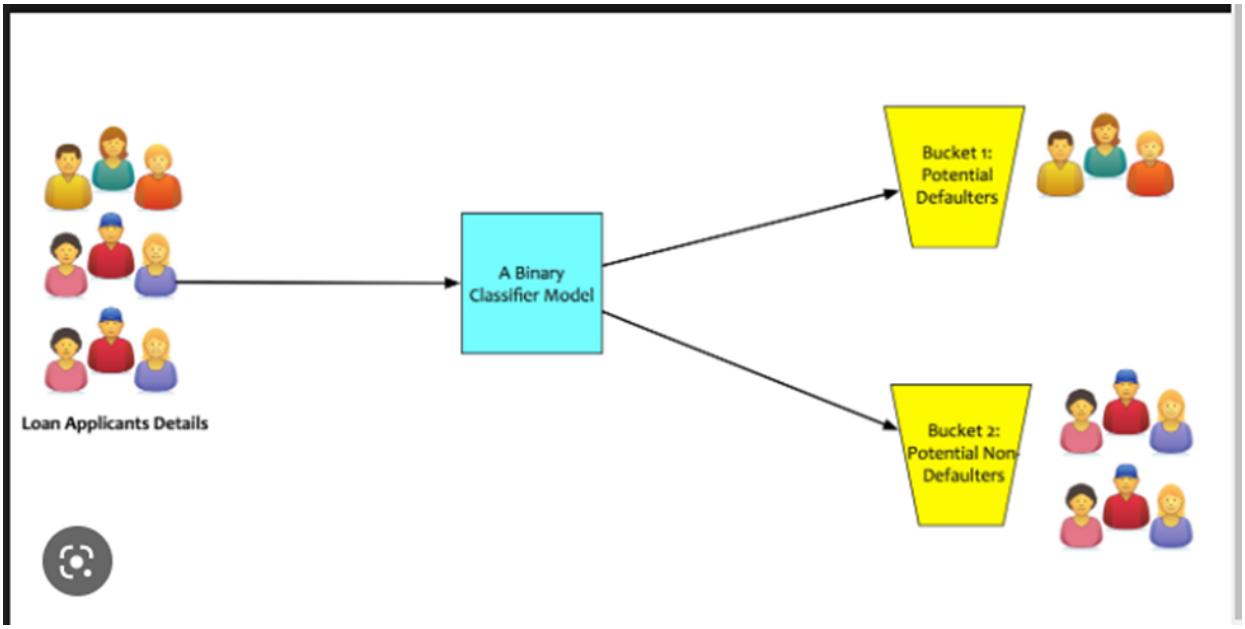
Hayford Osumanu

December 2022

Artificial Intelligence and Machine Learning is the Oxgyen to Today's Business




FICO Credit Score Ranges
FICO® Credit Score
750 - 850
Excellent
Credit Report
760
Excellent
Related searches
credit limit
equifax credit report
loan
Credit Score
Credit Report
540
What is credit history?
Credit history
811
CREDIT REPORT
Your Credit Score is:
730
APPROVED
730+
700-729
670-699
585-669
584 or below
poor credit
CREDIT REPORT
WHAT
FACTORS
Components of a credit report:
Credit Score



Executive Summary

The primary objective of this study is to build machine learning or classification model that will help to prediction of Loans Default based on applicants credit history. To achieve the above-mentioned objective, the analyst used Univariate, Multivariate Analysis, and Bagging & Boosting machine learning decision trees models.

Univariate data analysis was used to explore all the variables and provide observations on the distributions of all the relevant variables in the dataset. Besides, Multivariate data analysis was also used to help explore relationships between the important variables in the dataset. Finally, Decision Tree Models (Bagging and Boosting) was used to identify a relationship between the independent variable(s) and the dependent variable to help predict the dependent/target/response variable using the independent/explanatory/regressor. The primary Statistical/ML software used for the analysis was Python.

From the analysis it was observed that the most important factors for prediction loan default include loan amount, months of loan duration,applicants credit history,checking balance and savings balance.

Problem Statement, Business Objectives and Data Description

Importing all the Relevant Libraries for Data

Analysis

In [9]:
```
# installing the needed upgrade optimum performance
#!pip install nb_black
#!pip install ipython --upgrade
#!pip install xgboost
```

In [10]:
```
# Other important packages for data data analysis
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.stats import pearsonr
from statsmodels.formula.api import ols
from statsmodels.graphics.gofplots import ProbPlot
import statsmodels.api as sm
from scipy import stats;
```

In [11]:
```
#format numeric data for easier readability
pd.set_option(
    "display.float_format", lambda x: "%.2f" % x
)  # to display numbers rounded off to 2 decimal places
```

In [12]:
```
# this will help in making the Python code more structured automatically (good coding p
%load_ext nb_black

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd

# Libraries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns



# split the data into train and test
from sklearn.model_selection import train_test_split

# to build linear regression_model
from sklearn.linear_model import LinearRegression

# to check model performance
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# to build linear regression_model using statsmodels
import statsmodels.api as sm

# to compute VIF
from statsmodels.stats.outliers_influence import variance_inflation_factor

# # Command to tell Python to actually display the graphs
%matplotlib inline
pd.set_option('display.float_format', lambda x: '%.2f' % x) # To supress numerical disp
```

```
# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200);
```

In [13]:

```
# this will help in making the Python code more structured automatically (good coding p
%load_ext nb_black
import warnings

warnings.filterwarnings("ignore")

# Libraries to help with reading and manipulating data
import numpy as np
import pandas as pd
import xgboost as xgb

# Library to split data
from sklearn.model_selection import train_test_split

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)
# Sets the limit for the number of displayed rows
pd.set_option("display.max_rows", 200)


# Libraries different ensemble classifiers
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    AdaBoostClassifier,
    GradientBoostingClassifier,
    StackingClassifier,
)

from xgboost import XGBClassifier
from sklearn.tree import DecisionTreeClassifier

# Libraries to get different metric scores
from sklearn import metrics
from sklearn.metrics import (
    confusion_matrix,
    accuracy_score,
    precision_score,
    recall_score,
    f1_score,
)

# To tune different models
from sklearn.model_selection import GridSearchCV
```

```
The nb_black extension is already loaded. To reload it, use:
```

```
%reload_ext nb_black
```

In [14]:

```
# Libraries to help with reading and manipulating data

import pandas as pd
import numpy as np

# Library to split data
from sklearn.model_selection import train_test_split

# libaries to help with data visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Removes the limit for the number of displayed columns
pd.set_option("display.max_columns", None)

# To build Logistic Regression model for prediction
import statsmodels.stats.api as sms
from statsmodels.stats.outliers_influence import variance_inflation_factor
import statsmodels.api as sm
from statsmodels.tools.tools import add_constant
from sklearn.linear_model import LogisticRegression

# To get diferent metric scores
from sklearn.metrics import (
    f1_score,
    accuracy_score,
    recall_score,
    precision_score,
    confusion_matrix,
    roc_auc_score,
    plot_confusion_matrix,
    precision_recall_curve,
    roc_curve,
    make_scorer,
)

# Libraries to build decision tree classifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import tree

# To tune different models
from sklearn.model_selection import GridSearchCV
```

In [15]:

```
# Import all the Required Statistical Distributions
from scipy import stats
from scipy.stats import ttest_1samp
from numpy import sqrt, abs
from scipy.stats import norm
from scipy.stats import ttest_ind
from scipy.stats import ttest_rel
from statsmodels.stats.proportion import proportions_ztest
from scipy.stats import chi2
from scipy.stats import f
from scipy.stats import chi2_contingency
```

```
from scipy.stats import f_oneway
from statsmodels.stats.multicomp import pairwise_tukeyhsd
from scipy.stats import levene
```

In [16]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import metrics

from sklearn.impute import SimpleImputer

from sklearn.tree import DecisionTreeClassifier
```

Data Analysis Philosophy

In [17]:

```
import this
```

```
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

Reading the Data into a DataFrame

In [18]:

```
# mounting  google drive to enable data reading
# from google.colab import drive
# drive.mount('/content/drive')
```

In [19]:
```
# Reading Dataset from Google Drive
# Loan_data=pd.read_csv("/content/drive/MyDrive/DataBank/Loans.csv")
```

In [20]:
```
# Reading the data from the local drive
Loan_data = pd.read_csv("C:/Users/hayfo/OneDrive/Desktop/Data Publication/Loans.csv")
```

In [21]:
```
# copying data to another variable to avoid any changes to original data
data = Loan_data.copy()
```

Part I: Data Overview

Observation of the DataFrame

The initial steps to get an overview of any dataset is to:

- observe the first few rows of the dataset, to check whether the dataset has been loaded properly or not
- get information about the number of rows and columns in the dataset
- find out the data types of the columns to ensure that data is stored in the preferred format and the value of each property is as expected.
- check the statistical summary of the dataset to get an overview of the numerical columns of the data

In [22]:
```
# Creating  a well readable header label
for header in data.columns:
    header_replace = header.replace(
        " ", "_"
    )  # creates new header with "_" instead of " ".
    data.rename(
        {header: header_replace}, axis=1, inplace=True
    )  # sets new header as header made in line above
```

In [23]:
```
# Extracting the first 10 rows ofthe dataset
data.head(10)
```

Out[23]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance
0	< 0 DM	6	critical	furniture/appliances	1169	unknown

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance
1	1 - 200 DM	48	good	furniture/appliances	5951	< 100 DM
2	unknown	12	critical	education	2096	< 100 DM
3	< 0 DM	42	good	furniture/appliances	7882	< 100 DM
4	< 0 DM	24	poor	car	4870	< 100 DM
5	unknown	36	good	education	9055	unknown
6	unknown	24	good	furniture/appliances	2835	500 - 1000 DM
7	1 - 200 DM	36	good	car	6948	< 100 DM
8	unknown	12	good	furniture/appliances	3059	> 1000 DM
9	1 - 200 DM	30	critical	car	5234	< 100 DM

In [24]:
```
# observing the last 10 rows of the dataset
data.tail(10)
```

Out[24]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balanc
990	unknown	12	critical	education	3565	unknow
991	unknown	15	very good	furniture/appliances	1569	100 - 500 DI
992	< 0 DM	18	good	furniture/appliances	1936	unknow
993	< 0 DM	36	good	furniture/appliances	3959	< 100 DI
994	unknown	12	good	car	2390	unknow
995	unknown	12	good	furniture/appliances	1736	< 100 DI
996	< 0 DM	30	good	car	3857	< 100 DI
997	unknown	12	good	furniture/appliances	804	< 100 DI
998	< 0 DM	45	good	furniture/appliances	1845	< 100 DI
999	1 - 200 DM	45	critical	car	4576	100 - 500 DI

In [25]:
```
# let's view a random sample of 10  observations of the entire dataset
data.sample(n=10, random_state=1)
```

Out[25]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balanc
507	1 - 200 DM	15	very good	car	6850	100 - 500 DI
818	< 0 DM	36	good	car0	15857	< 100 DI
452	unknown	12	perfect	furniture/appliances	2759	< 100 DI
368	< 0 DM	36	good	furniture/appliances	3446	< 100 DI

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balanc
242	< 0 DM	48	perfect	car	4605	< 100 DI
929	< 0 DM	12	poor	car	1344	< 100 DI
262	< 0 DM	18	critical	car	5302	< 100 DI
810	1 - 200 DM	8	good	business	907	< 100 DI
318	unknown	12	critical	education	701	< 100 DI
49	unknown	12	good	furniture/appliances	2073	100 - 500 DI

In [26]:
```
# let's view a random sample of 10  observations of the entire dataset
data.sample(n=10, random_state=2)
```

Out[26]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balanc
37	> 200 DM	18	good	furniture/appliances	2100	< 100 DI
726	unknown	15	critical	furniture/appliances	1316	500 - 1000 DI
846	unknown	18	good	car	6761	unknow
295	1 - 200 DM	48	good	furniture/appliances	9960	< 100 DI
924	< 0 DM	24	very good	furniture/appliances	6872	< 100 DI
658	1 - 200 DM	30	perfect	business	4221	< 100 DI
682	unknown	15	poor	furniture/appliances	1478	< 100 DI
286	< 0 DM	48	good	car	4788	< 100 DI
880	unknown	24	good	car	7814	< 100 DI
272	1 - 200 DM	48	very good	car	12169	unknow

Columns/Variable/Features of the Dataset

In [27]:
```
# Extracting the columns/variables of the dataset
data.columns
```

Out[27]:
```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

Dimension of the Dataset

In [28]:
```
# checking the shape of the data
print(f"There are {data.shape[0]} rows and {data.shape[1]} columns.")
```

```
There are 1000 rows and 17 columns.
```

In [29]:
```
# Checking the dimension (number of observations/rows and variables/columns of the Data
print("There are", data.shape[0], "rows and", data.shape[1], "columns.")
```

```
There are 1000 rows and 17 columns.
```

Data Types of the Dataset

In [30]:
```
# Checking the data types of the variables/columns for the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64 
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64 
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64 
 8   years_at_residence    1000 non-null   int64 
 9   age                   1000 non-null   int64 
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64 
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64 
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

Checking Values Equal to Zero or Less

In [31]:
```
# Extracting Values Equal to Zero
(data == 0)
```

Out[31]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	employ
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	employ
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
...	...	...	...	...	...	...	
995	False	False	False	False	False	False	
996	False	False	False	False	False	False	
997	False	False	False	False	False	False	
998	False	False	False	False	False	False	
999	False	False	False	False	False	False	

1000 rows × 17 columns

In [32]:
```
# Extracting the Actual Number of Values Equal to Zero
(data == 0).shape[0]
```

Out[32]:
```
1000
```

Observation

Checking the Missing Values of the Dataset

In [33]:
```
# Checking for missing values in the dataset
data.isnull().sum()
```

Out[33]:
```
checking_balance         0
months_loan_duration     0
credit_history           0
purpose                  0
amount                   0
savings_balance          0
employment_duration      0
percent_of_income        0
years_at_residence       0
age                      0
other_credit             0
housing                  0
existing_loans_count     0
job                      0
dependents               0
phone                    0
default                  0
dtype: int64
```

In [34]:
```
# Checking the total number of missing values in the dataset
```

```
data.isnull().sum().sum()
```

Out[34]:
```
0
```

Observation

- There are no missing values in the dataset
- However, we need to fill in the zero values with the median values of each column

---

Checking the Duplicates in the Dataset

In [35]:
```
# checking for duplicate values
print("There are about: ", data.duplicated().sum(), "dupplicates in the dataset")
```

```
There are about:  0 dupplicates in the dataset
```

Removing Duplicates from the Dataset

In [36]:
```
# dropping duplicate entries from the data
data.drop_duplicates(inplace=True)

# resetting the index of data frame since some rows will be removed
data.reset_index(drop=True, inplace=True)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64 
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64 
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64 
 8   years_at_residence    1000 non-null   int64 
 9   age                   1000 non-null   int64 
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64 
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64 
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

There are no duplicates in the data

Data Sanity Checks: Deep Checking/scrutinity of the the dataset before EDA

Data Types of the Dataset

In [37]:
```
# Checking the datatypes of the dataset
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64 
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64 
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64 
 8   years_at_residence    1000 non-null   int64 
 9   age                   1000 non-null   int64 
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64 
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64 
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

Overview of Categorical/Object Variables

In [38]:
```
# Making a list of all catrgorical variables
cat_col = list(data.select_dtypes("object").columns)

# Printing number of count of each unique value in each column
for column in cat_col:
    print(data[column].value_counts())
    # print("cat_col")
```

```
unknown       394
< 0 DM        274
1 - 200 DM    269
> 200 DM       63
Name: checking_balance, dtype: int64
good        530
critical    293
```

```
poor          88
very good     49
perfect       40
Name: credit_history, dtype: int64
furniture/appliances    473
car                     337
business                 97
education                59
renovations              22
car0                     12
Name: purpose, dtype: int64
< 100 DM         603
unknown          183
100 - 500 DM     103
500 - 1000 DM     63
> 1000 DM         48
Name: savings_balance, dtype: int64
1 - 4 years    339
> 7 years      253
4 - 7 years    174
< 1 year       172
unemployed      62
Name: employment_duration, dtype: int64
none     814
bank     139
store     47
Name: other_credit, dtype: int64
own      713
rent     179
other    108
Name: housing, dtype: int64
skilled       630
unskilled     200
management    148
unemployed     22
Name: job, dtype: int64
no     596
yes    404
Name: phone, dtype: int64
no     700
yes    300
Name: default, dtype: int64
```

Observation

- There are no categorical variables in the dataset

Part II: Exploratory Data Analysis (EDA)

Numerical Analysis of the Dataset

Statistical summary of the numerical columns in both

train and test dataset

Statistical Summary of the Dataset

In [39]:
```
# let's view the statistical summary of minimum numerical columns in the data
data.describe(include=np.number).T.style.highlight_min(color="green", axis=0)
```

Out[39]:

	count	mean	std	min	25%	50%	
months_loan_duration	1000.000000	20.903000	12.058814	4.000000	12.000000	18.000000	2
amount	1000.000000	3271.258000	2822.736876	250.000000	1365.500000	2319.500000	397
percent_of_income	1000.000000	2.973000	1.118715	1.000000	2.000000	3.000000	
years_at_residence	1000.000000	2.845000	1.103718	1.000000	2.000000	3.000000	
age	1000.000000	35.546000	11.375469	19.000000	27.000000	33.000000	4
existing_loans_count	1000.000000	1.407000	0.577654	1.000000	1.000000	1.000000	
dependents	1000.000000	1.155000	0.362086	1.000000	1.000000	1.000000	

In [40]:
```
# let's view the statistical summary of maximum numerical columns in the data
data.describe(include=np.number).T.style.highlight_max(color="indigo", axis=0)
```

Out[40]:

	count	mean	std	min	25%	50%	
months_loan_duration	1000.000000	20.903000	12.058814	4.000000	12.000000	18.000000	2
amount	1000.000000	3271.258000	2822.736876	250.000000	1365.500000	2319.500000	397
percent_of_income	1000.000000	2.973000	1.118715	1.000000	2.000000	3.000000	
years_at_residence	1000.000000	2.845000	1.103718	1.000000	2.000000	3.000000	
age	1000.000000	35.546000	11.375469	19.000000	27.000000	33.000000	4
existing_loans_count	1000.000000	1.407000	0.577654	1.000000	1.000000	1.000000	
dependents	1000.000000	1.155000	0.362086	1.000000	1.000000	1.000000	

In [41]:
```
# Extracting the Quantiles of the dataset
data.quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T.style.highlight_max(
    color="purple", axis=0
)
```

Out[41]:

	0.25	0.5	0.6	0.75	0.9	0.95	
months_loan_duration	12.000000	18.000000	24.000000	24.000000	36.000000	48.000000	
amount	1365.500000	2319.500000	2852.400000	3972.250000	7179.400000	9162.700000	14
percent_of_income	2.000000	3.000000	4.000000	4.000000	4.000000	4.000000	

	0.25	0.5	0.6	0.75	0.9	0.95
years_at_residence	2.000000	3.000000	4.000000	4.000000	4.000000	4.000000
age	27.000000	33.000000	36.000000	42.000000	52.000000	60.000000
existing_loans_count	1.000000	1.000000	1.000000	2.000000	2.000000	2.000000
dependents	1.000000	1.000000	1.000000	1.000000	2.000000	2.000000

In [42]:
```
# Extracting the Quantiles of the dataset
data.quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T.style.highlight_min(
    color="red", axis=0
)
```

Out[42]:

	0.25	0.5	0.6	0.75	0.9	0.95
months_loan_duration	12.000000	18.000000	24.000000	24.000000	36.000000	48.000000
amount	1365.500000	2319.500000	2852.400000	3972.250000	7179.400000	9162.700000
percent_of_income	2.000000	3.000000	4.000000	4.000000	4.000000	4.000000
years_at_residence	2.000000	3.000000	4.000000	4.000000	4.000000	4.000000
age	27.000000	33.000000	36.000000	42.000000	52.000000	60.000000
existing_loans_count	1.000000	1.000000	1.000000	2.000000	2.000000	2.000000
dependents	1.000000	1.000000	1.000000	1.000000	2.000000	2.000000

In [ ]:
```
```

Graphical Univariate Analysis

In [43]:
```
len(data.columns)
```

Out[43]:
```
17
```

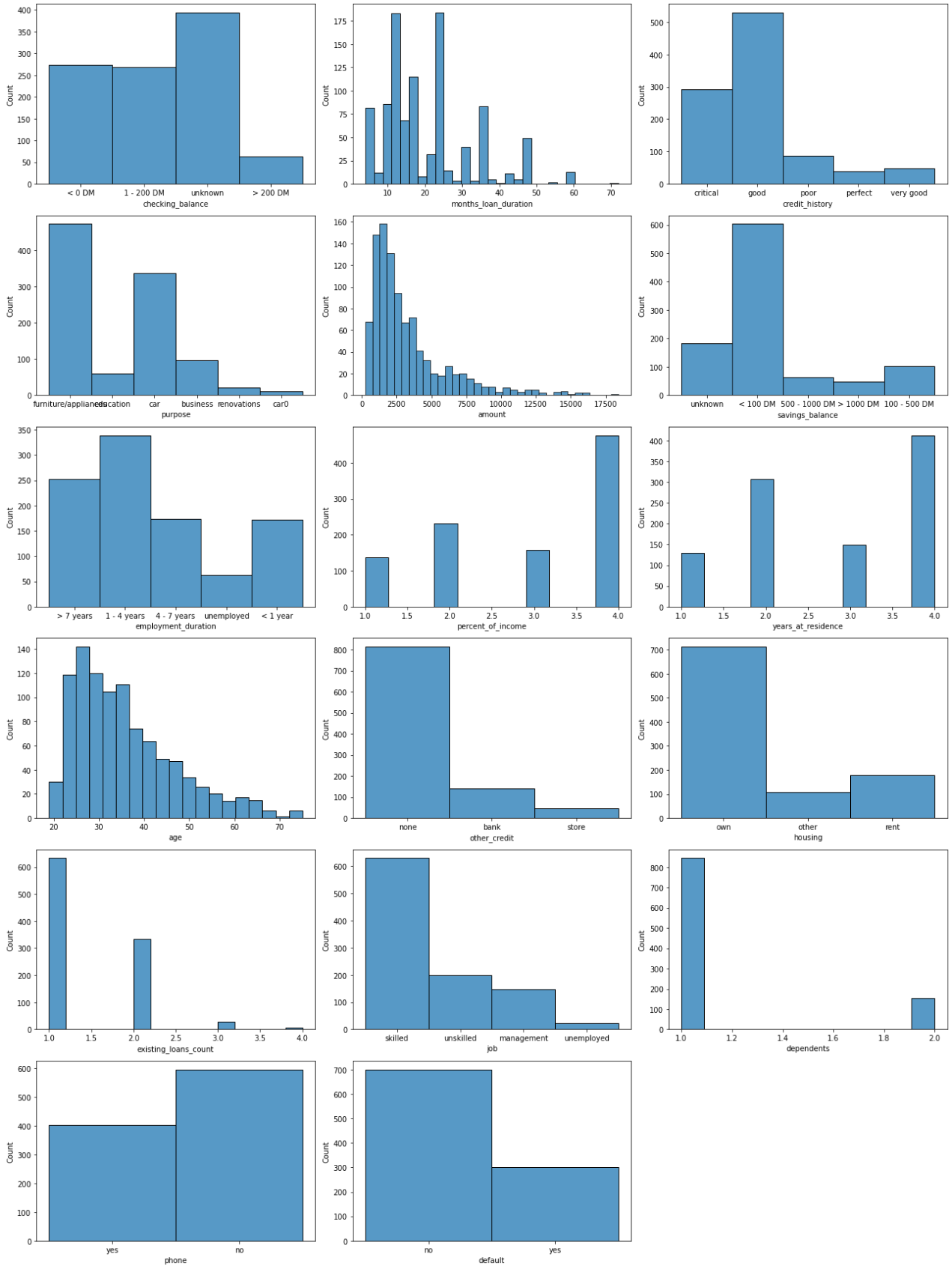
In [44]:
```
# Checking the histogram plot  of the entire dataset
cols = 5
rows = 5
num_cols = data.select_dtypes(include="category").columns
fig = plt.figure(figsize=(cols * 5, rows * 5))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.histplot(x=data[col], ax=ax)
```

```
fig.tight_layout()
plt.show()
```

```
<Figure size 1800x1800 with 0 Axes>
```

In [45]:

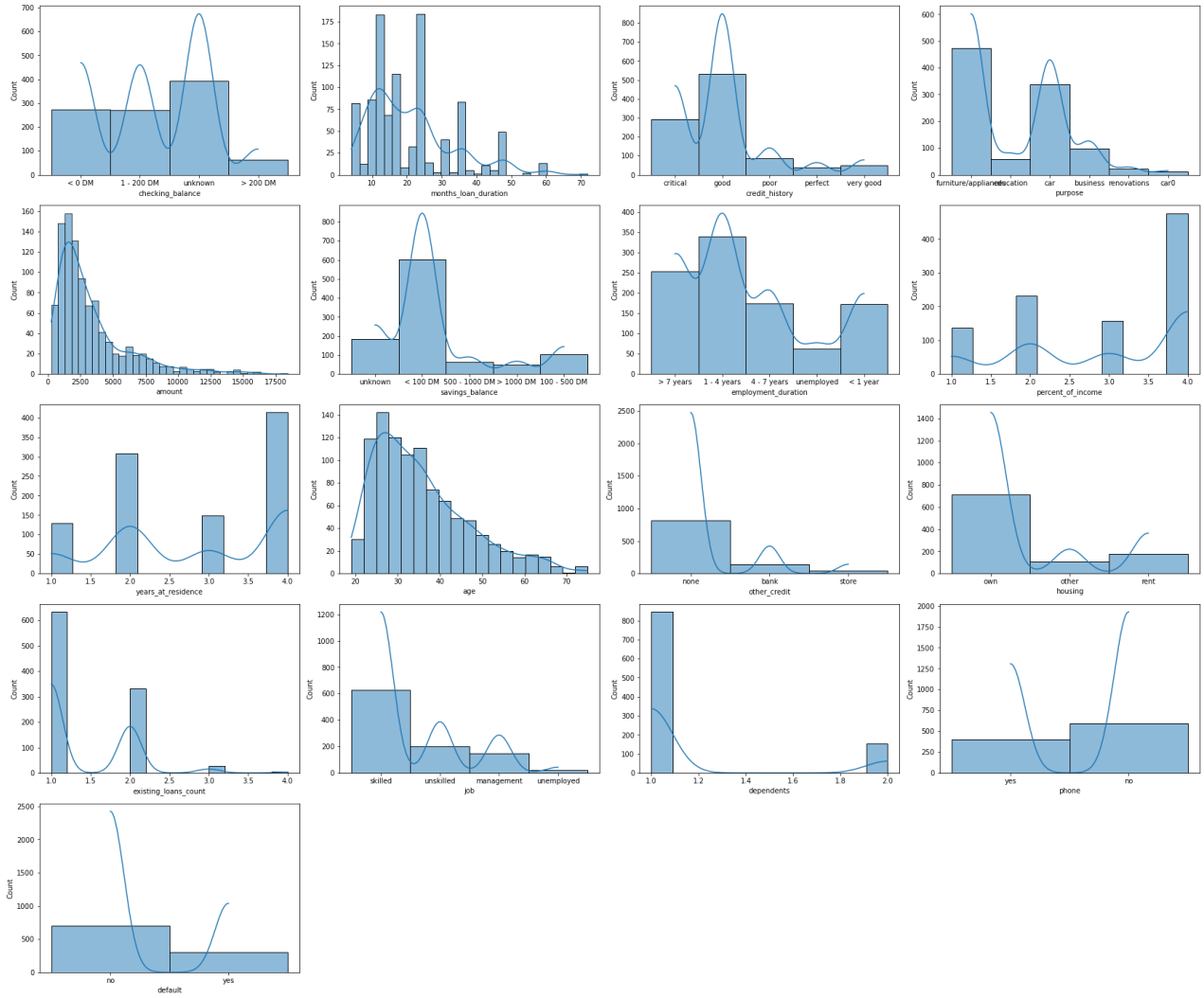
```
# Checking the histogram plot of numerical variables of the entire dataset
cols = 3
rows = 6
num_cols = data.select_dtypes(exclude="category").columns
fig = plt.figure(figsize=(cols * 6, rows * 4))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.histplot(x=data[col], ax=ax)

fig.tight_layout()
plt.show()
```



In [46]:

```
# Checking the histogram plot of numerical variables of the entire dataset
cols = 4
rows = 5
```

```
num_cols = data.select_dtypes(exclude="category").columns
fig = plt.figure(figsize=(cols * 6, rows * 4))
for i, col in enumerate(num_cols):

    ax = fig.add_subplot(rows, cols, i + 1)

    sns.histplot(x=data[col], kde=True, ax=ax)

fig.tight_layout()
plt.show()
```



Detailed Univariate Analysis

In [ ]:

Univariate analysis

In [47]:

```
# function to plot a boxplot and a histogram along the same scale.
```

```
def histogram_boxplot(data, feature, figsize=(18, 8), kde=True, bins=None):
    """
    Boxplot and histogram combined

    data: dataframe
    feature: dataframe column
    figsize: size of figure (default (12,7))
    kde: whether to the show density curve (default False)
    bins: number of bins for histogram (default None)
    """
    f2, (ax_box2, ax_hist2) = plt.subplots(
        nrows=2,  # Number of rows of the subplot grid= 2
        sharex=True,  # x-axis will be shared among all subplots
        gridspec_kw={"height_ratios": (0.25, 0.75)},
        figsize=figsize,
    )  # creating the 2 subplots
    sns.boxplot(
        data=data, x=feature, ax=ax_box2, showmeans=True, color="red"
    )  # boxplot will be created and a star will indicate the mean value of the column
    sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2, bins=bins, palette="winter"
    ) if bins else sns.histplot(
        data=data, x=feature, kde=kde, ax=ax_hist2
    )  # For histogram
    ax_hist2.axvline(
        data[feature].mean(), color="green", linestyle="--"
    )  # Add mean to the histogram
    ax_hist2.axvline(
        data[feature].median(), color="indigo", linestyle="-"
    )  # Add median to the histogram
```

In [48]:
```
data.columns
```

Out[48]:
```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

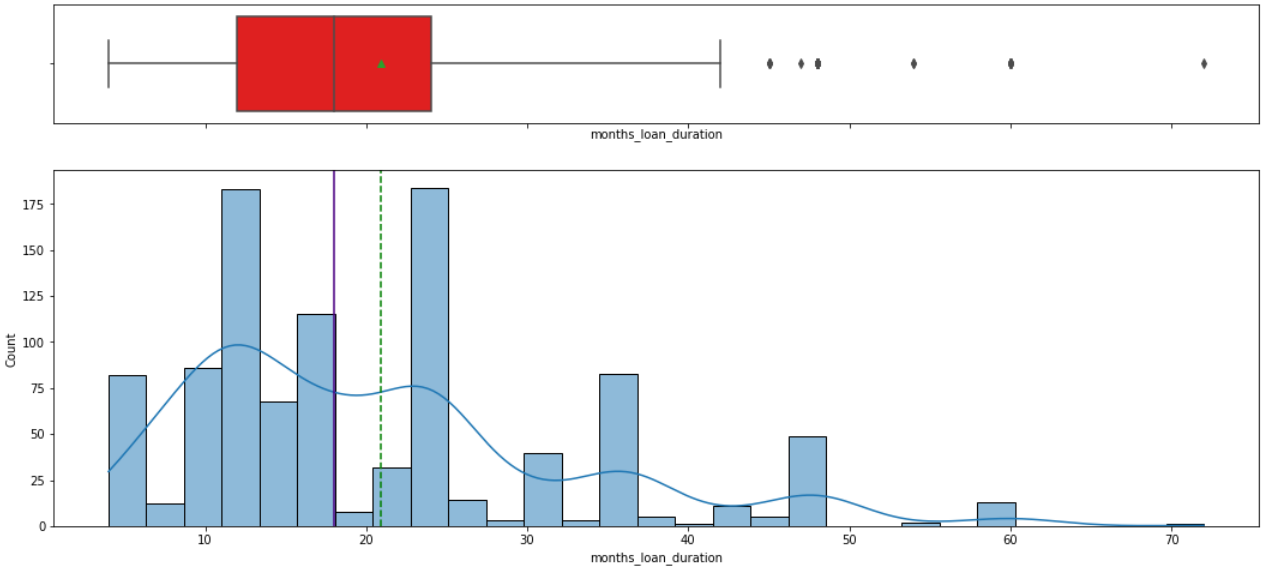
In [49]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64
 5   savings_balance       1000 non-null   object
```

```
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64
 8   years_at_residence    1000 non-null   int64
 9   age                   1000 non-null   int64
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

Analysis of Months of Loan Duration Variable

In [50]:
```
histogram_boxplot(data, "months_loan_duration")
```



In [51]:
```
data["months_loan_duration"].describe()
```

Out[51]:
```
count   1000.00
mean      20.90
std       12.06
min        4.00
25%       12.00
50%       18.00
75%       24.00
max       72.00
Name: months_loan_duration, dtype: float64
```

In [52]:
```
data["months_loan_duration"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

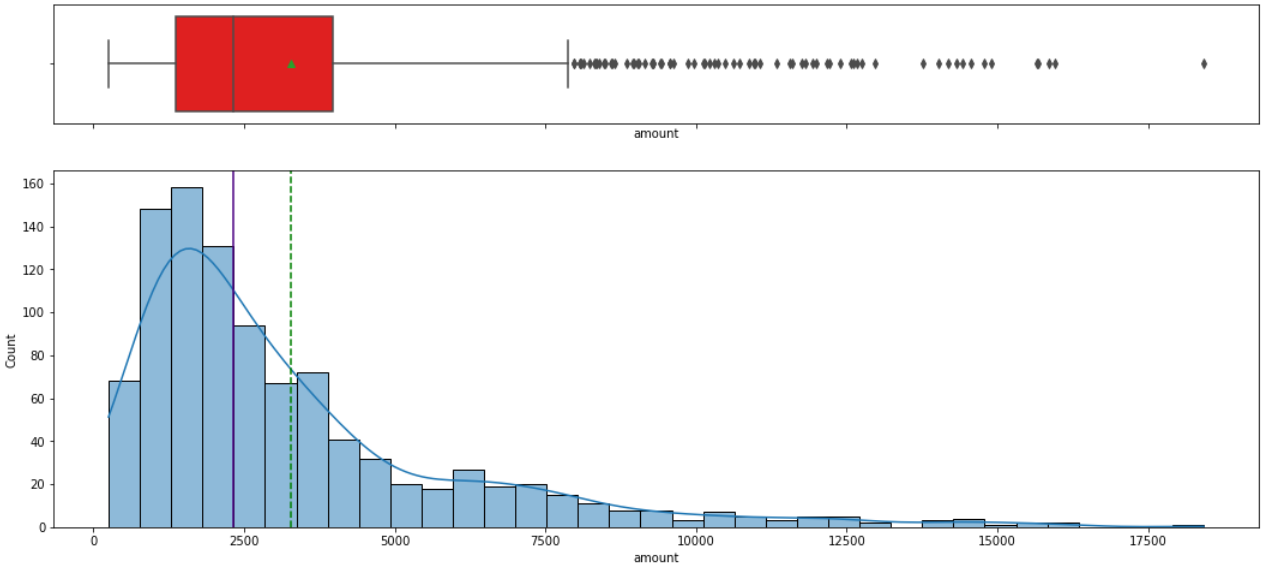
Out[52]:
```
0.25   12.00
0.50   18.00
0.60   24.00
0.75   24.00
```

```
0.90   36.00
0.95   48.00
0.99   60.00
Name: months_loan_duration, dtype: float64
```

In [ ]:

Analysis of Amount Variable

In [53]:
```
histogram_boxplot(data, "amount")
```



In [54]:
```
data["amount"].describe()
```

Out[54]:
```
count    1000.00
mean     3271.26
std      2822.74
min       250.00
25%      1365.50
50%      2319.50
75%      3972.25
max     18424.00
Name: amount, dtype: float64
```

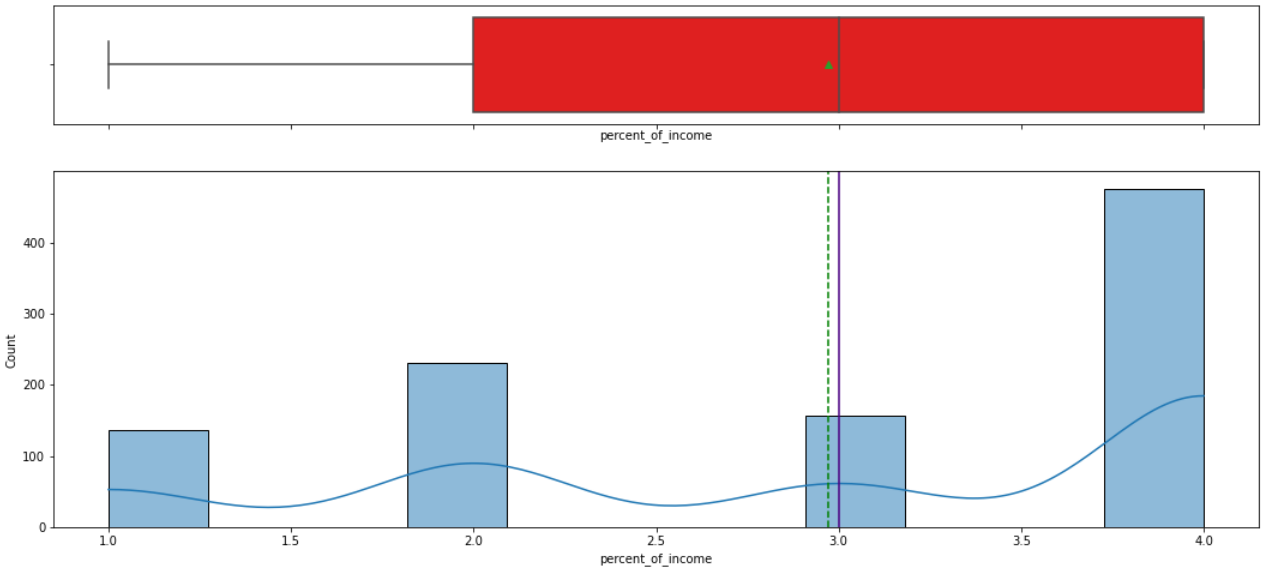
In [55]:
```
data["amount"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[55]:
```
0.25    1365.50
0.50    2319.50
0.60    2852.40
0.75    3972.25
0.90    7179.40
0.95    9162.70
0.99   14180.39
Name: amount, dtype: float64
```

In [ ]:

Anlaysis of Percent of Income Variable

In [56]:
```
histogram_boxplot(data, "percent_of_income")
```



In [57]:
```
data["percent_of_income"].describe()
```

Out[57]:
```
count   1000.00
mean       2.97
std        1.12
min        1.00
25%        2.00
50%        3.00
75%        4.00
max        4.00
Name: percent_of_income, dtype: float64
```

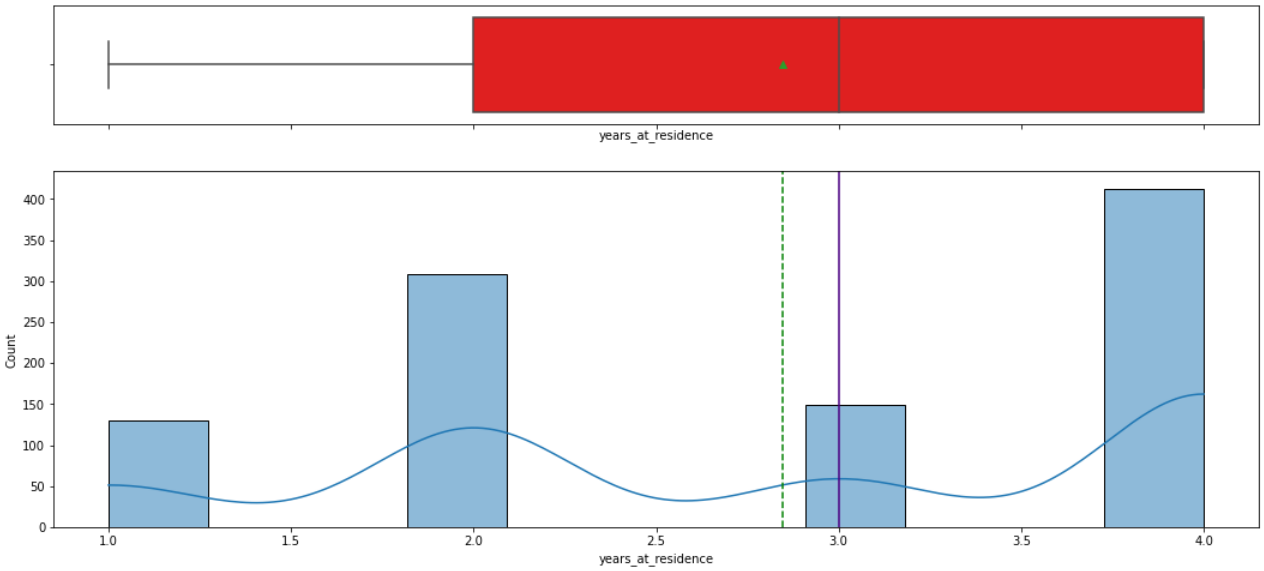
In [58]:
```
data["percent_of_income"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[58]:
```
0.25   2.00
0.50   3.00
0.60   4.00
0.75   4.00
0.90   4.00
0.95   4.00
0.99   4.00
Name: percent_of_income, dtype: float64
```

In [ ]:

Analysis of Years at Residence Variable

In [59]:
```
histogram_boxplot(data, "years_at_residence")
```



In [60]:
```
data[data["years_at_residence"] > 5]
```

Out[60]:

checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	employme

In [61]:
```
data["years_at_residence"].describe()
```

Out[61]:
```
count   1000.00
mean       2.85
std        1.10
min        1.00
25%        2.00
50%        3.00
75%        4.00
max        4.00
Name: years_at_residence, dtype: float64
```

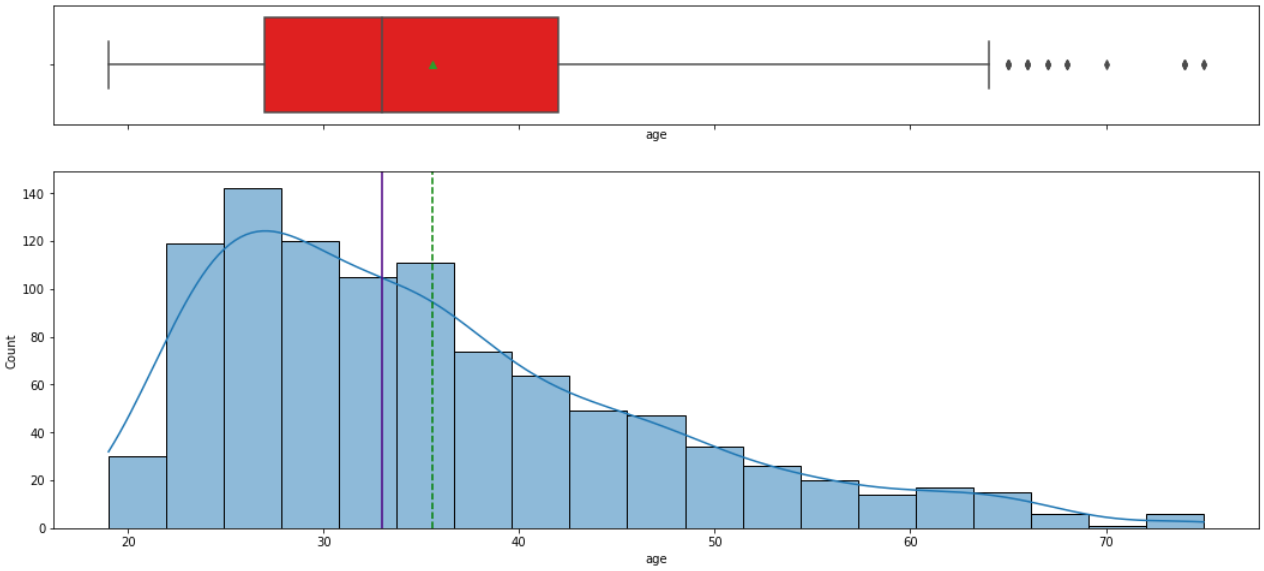
In [62]:
```
data["years_at_residence"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[62]:
```
0.25   2.00
0.50   3.00
0.60   4.00
0.75   4.00
0.90   4.00
0.95   4.00
0.99   4.00
Name: years_at_residence, dtype: float64
```

In [ ]:

Analysis of Age Variable

In [63]:
```
histogram_boxplot(data, "age")
```



In [64]:
```
data["age"].describe()
```

Out[64]:
```
count   1000.00
mean      35.55
std       11.38
min       19.00
25%       27.00
50%       33.00
75%       42.00
max       75.00
Name: age, dtype: float64
```

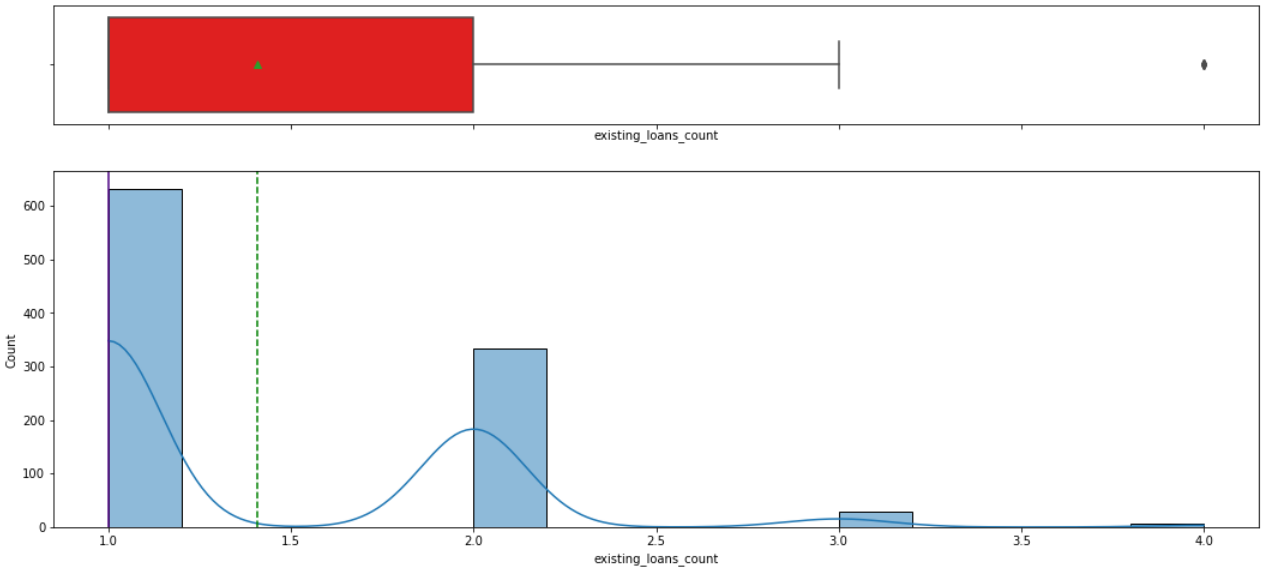
In [65]:
```
data["age"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[65]:
```
0.25   27.00
0.50   33.00
0.60   36.00
0.75   42.00
0.90   52.00
0.95   60.00
0.99   67.01
Name: age, dtype: float64
```

In [ ]:

Analysis of Existing Loans Count Variable

In [66]:
```
histogram_boxplot(data, "existing_loans_count")
```



In [67]:
```
data["existing_loans_count"].describe()
```

Out[67]:
```
count   1000.00
mean       1.41
std        0.58
min        1.00
25%        1.00
50%        1.00
75%        2.00
max        4.00
Name: existing_loans_count, dtype: float64
```

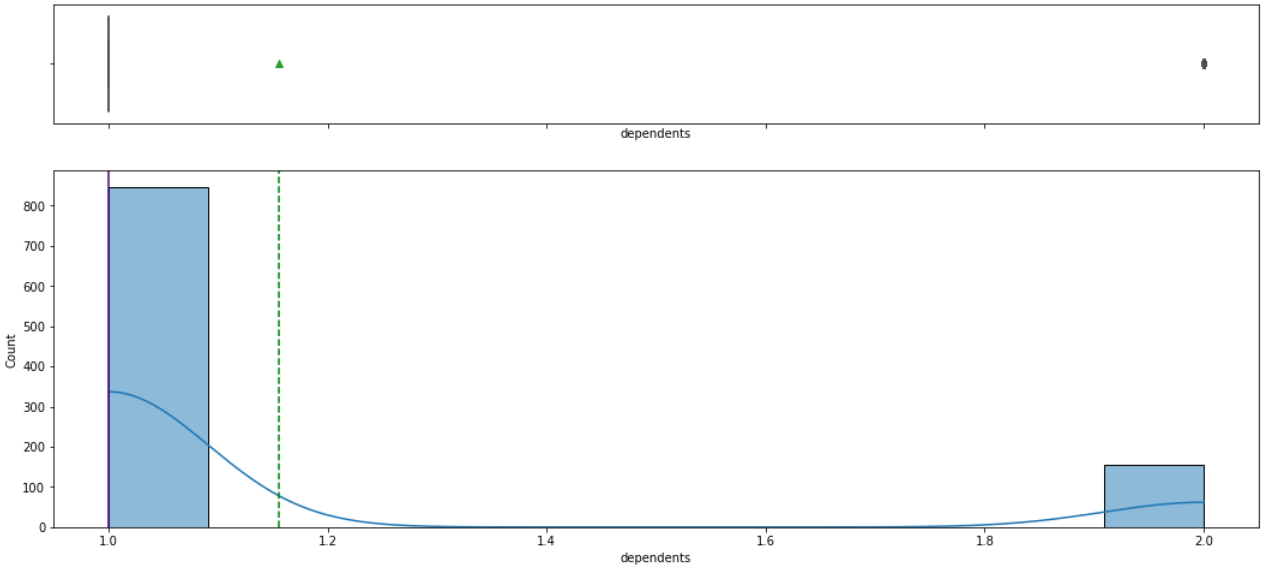
In [68]:
```
data["existing_loans_count"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[68]:
```
0.25   1.00
0.50   1.00
0.60   1.00
0.75   2.00
0.90   2.00
0.95   2.00
0.99   3.00
Name: existing_loans_count, dtype: float64
```

In [ ]:
```

```

Analysis of Dependents Variable

In [69]:
```
histogram_boxplot(data, "dependents")
```



In [70]:
```
data["dependents"].describe()
```

Out[70]:
```
count   1000.00
mean       1.16
std        0.36
min        1.00
25%        1.00
50%        1.00
75%        1.00
max        2.00
Name: dependents, dtype: float64
```

In [71]:
```
data["dependents"].quantile([0.25, 0.5, 0.6, 0.75, 0.9, 0.95, 0.99]).T
```

Out[71]:
```
0.25   1.00
0.50   1.00
0.60   1.00
0.75   1.00
0.90   2.00
0.95   2.00
0.99   2.00
Name: dependents, dtype: float64
```

Observation of Categorical Variables

In [72]:
```
# function to create labeled barplots


def labeled_barplot(data, feature, perc=False, n=None):
    """
```

```
    Barplot with percentage at the top

    data: dataframe
    feature: dataframe column
    perc: whether to display percentages instead of count (default is False)
    n: displays the top n category levels (default is None, i.e., display all levels)
    """

    total = len(data[feature])  # length of the column
    count = data[feature].nunique()
    if n is None:
        plt.figure(figsize=(count + 1, 5))
    else:
        plt.figure(figsize=(n + 4, 12))

    plt.xticks(rotation=90, fontsize=15)
    ax = sns.countplot(
        data=data,
        x=feature,
        palette="Paired",
        order=data[feature].value_counts().index[:n],
    )

    for p in ax.patches:
        if perc == True:
            label = "{:.1f}%".format(
                100 * p.get_height() / total
            )  # percentage of each class of the category
        else:
            label = p.get_height()  # count of each level of the category

        x = p.get_x() + p.get_width() / 2  # width of the plot
        y = p.get_height()  # height of the plot

        ax.annotate(
            label,
            (x, y),
            ha="center",
            va="center",
            size=12,
            xytext=(0, 5),
            textcoords="offset points",
        )  # annotate the percentage

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

In [73]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
```

```
 4   amount                1000 non-null   int64
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64
 8   years_at_residence    1000 non-null   int64
 9   age                   1000 non-null   int64
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64
 13  job                   1000 non-null   object
 14  dependents            1000 non-null   int64
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

In [ ]:

Observations of Loan Default

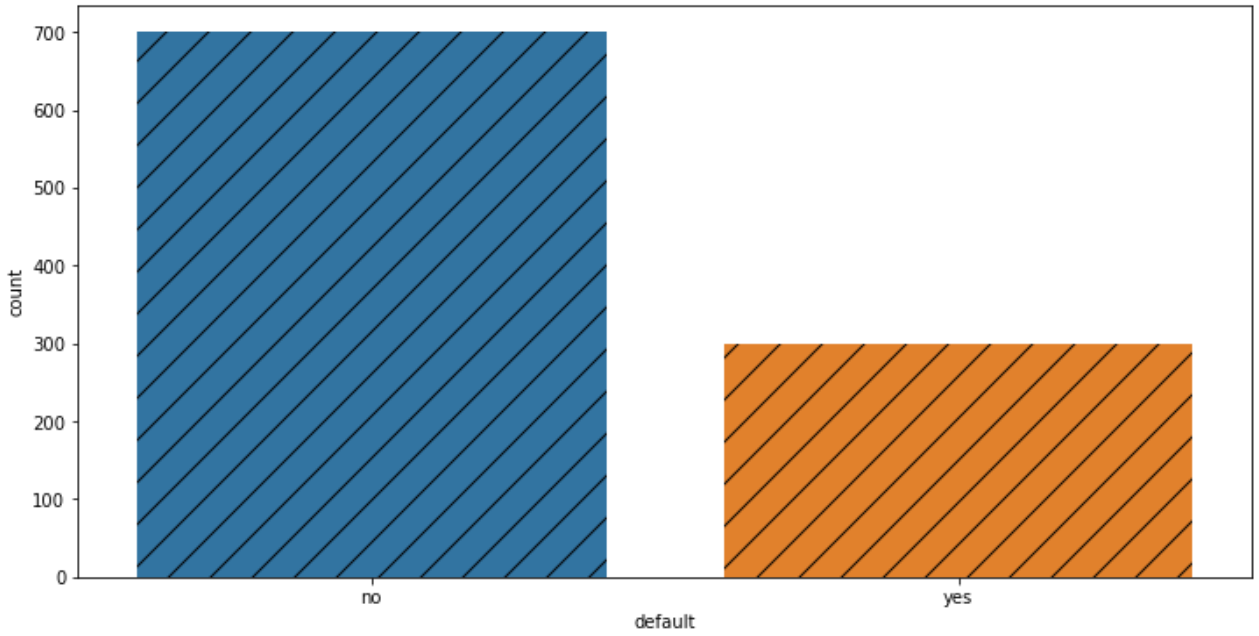
In [74]:
```
data["default"].value_counts(1)
```

Out[74]:
```
no    0.70
yes   0.30
Name: default, dtype: float64
```

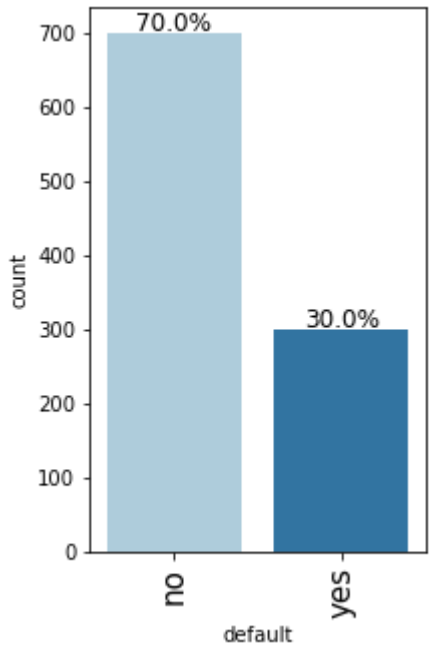
In [75]:
```
data["default"].value_counts()
```

Out[75]:
```
no     700
yes    300
Name: default, dtype: int64
```

In [76]:
```
# Extracting the Default
plt.subplots(figsize = (12,6))
sns.countplot(data = data, x = 'default', hatch="/");
```



In [77]:
```
labeled_barplot(data, "default", perc=True)
```



In [ ]:
```

```

Observations of Checking Balance

In [78]:
```
data["checking_balance"].value_counts(1)
```

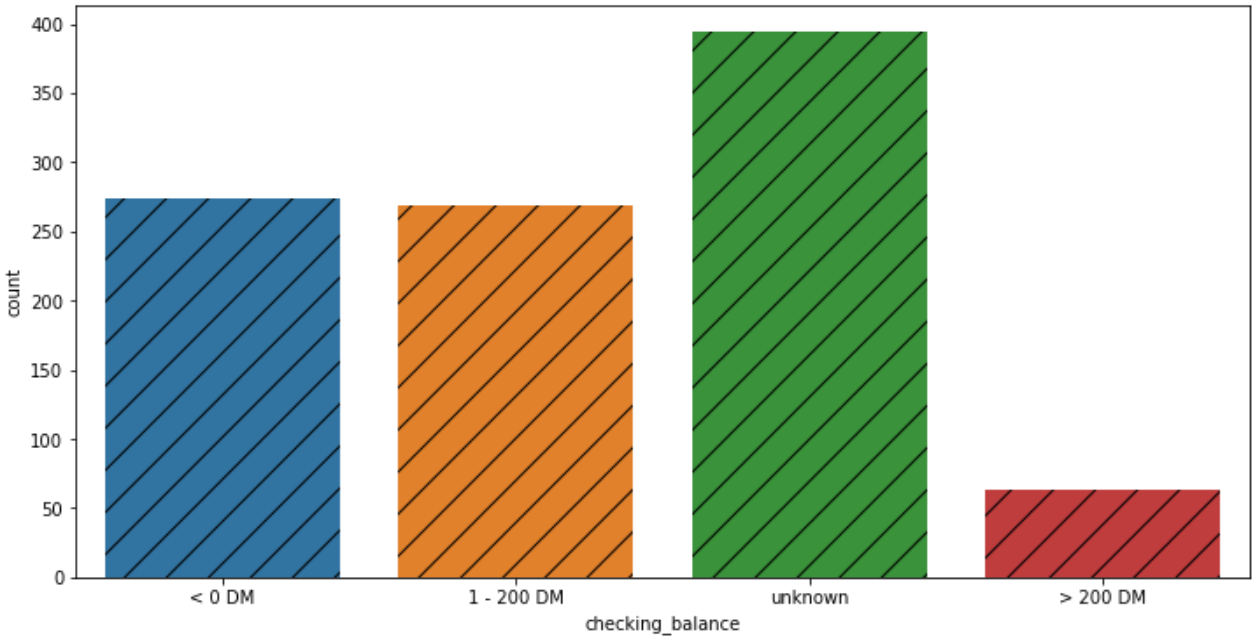
Out[78]:
```
unknown       0.39
< 0 DM        0.27
1 - 200 DM    0.27
```

```
> 200 DM     0.06
Name: checking_balance, dtype: float64
```

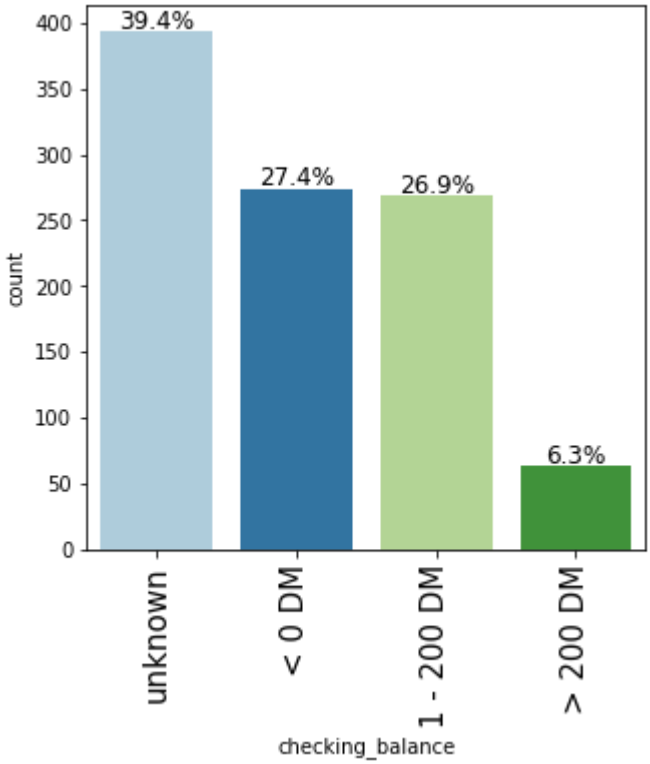
In [79]:
```
data["credit_history"].value_counts()
```

Out[79]:
```
good         530
critical     293
poor          88
very good     49
perfect       40
Name: credit_history, dtype: int64
```

In [80]:
```
# Extracting the Checking Balance
plt.subplots(figsize = (12,6))
sns.countplot(data = data, x = 'checking_balance', hatch="/");
```



In [81]:
```
labeled_barplot(data, "checking_balance", perc=True)
```



In [ ]:

Observations of credit_history

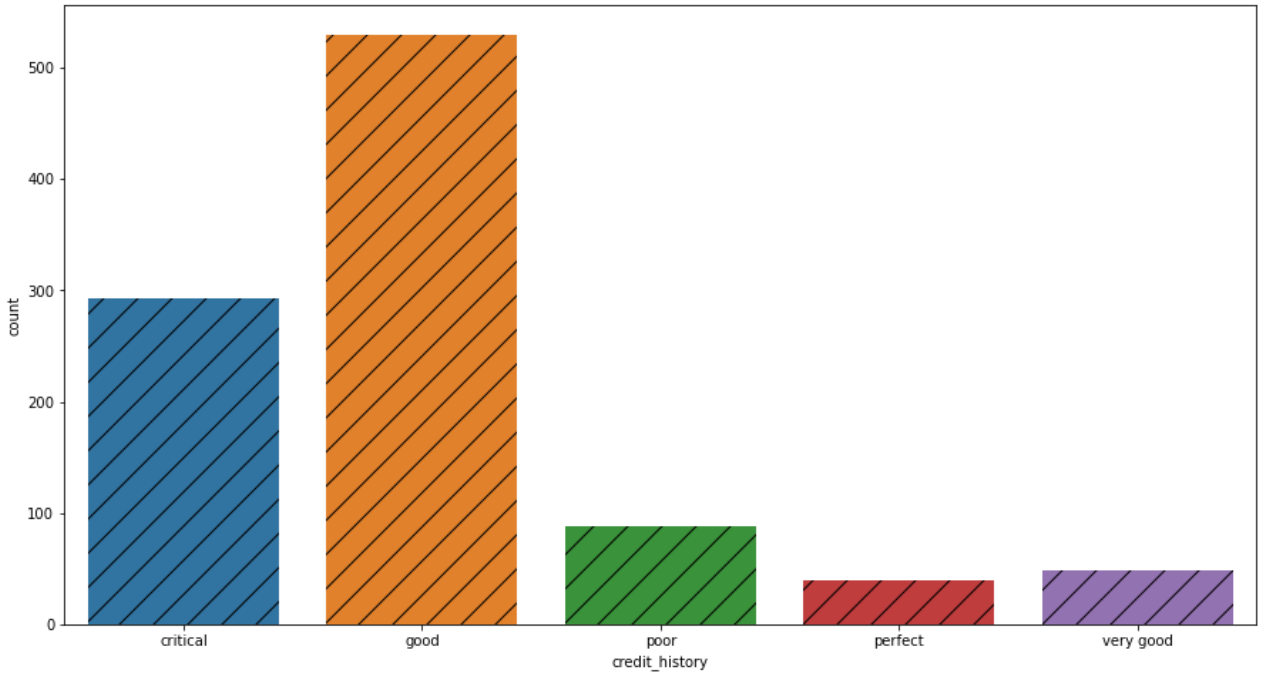
In [82]:
```
data["credit_history"].value_counts(1)
```

Out[82]:
```
good         0.53
critical     0.29
poor         0.09
very good    0.05
perfect      0.04
Name: credit_history, dtype: float64
```

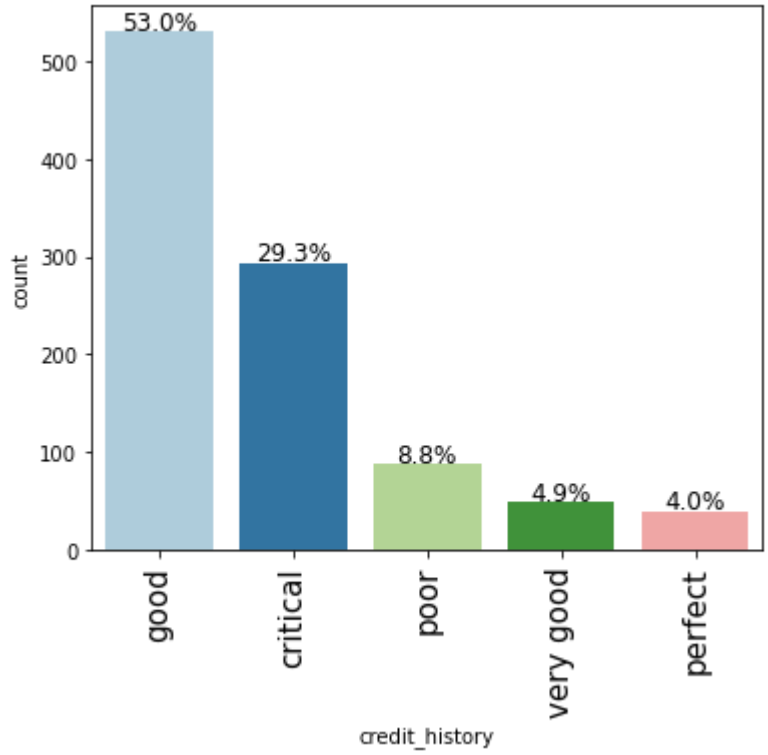
In [83]:
```
data["credit_history"].value_counts()
```

Out[83]:
```
good         530
critical     293
poor          88
very good     49
perfect       40
Name: credit_history, dtype: int64
```

In [84]:
```
# Extracting the Credit History
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'credit_history', hatch="/");
```



In [85]:
```
labeled_barplot(data, "credit_history", perc=True)
```



In [ ]:
```

```

Observations of Purpose of Loan

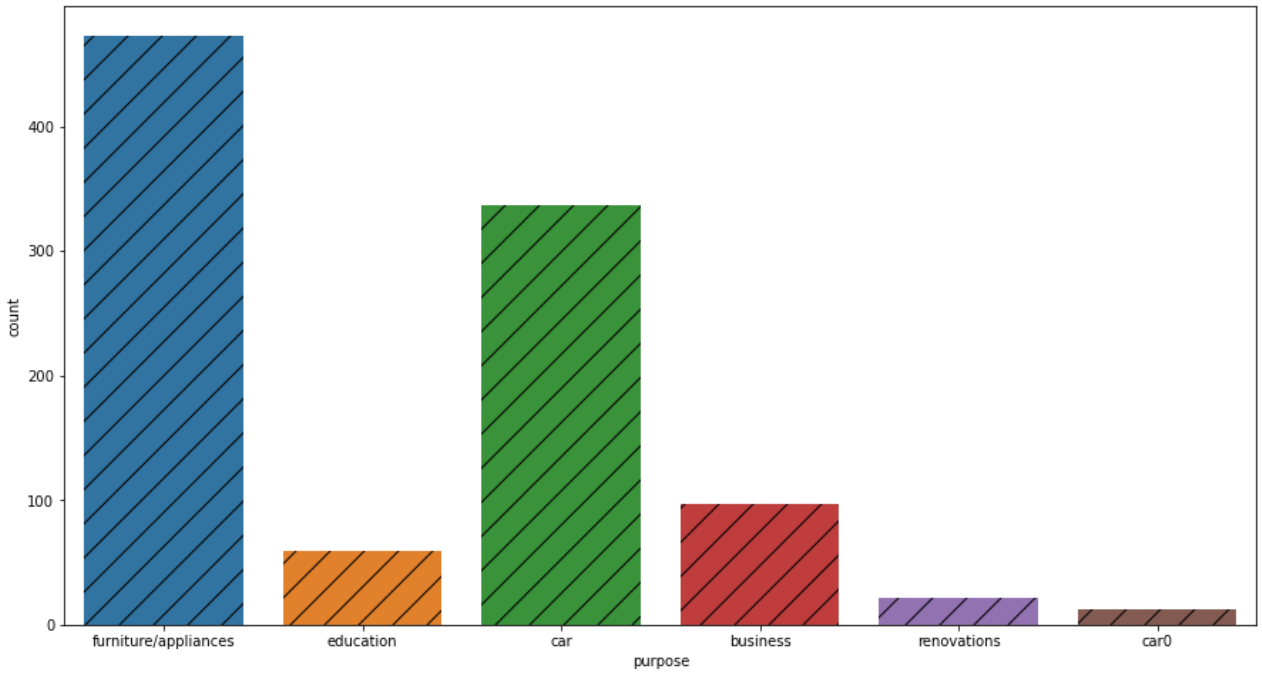
In [86]:
```
data["purpose"].value_counts(1)
```

Out[86]:
```
furniture/appliances   0.47
car                    0.34
business               0.10
education              0.06
renovations            0.02
car0                   0.01
Name: purpose, dtype: float64
```

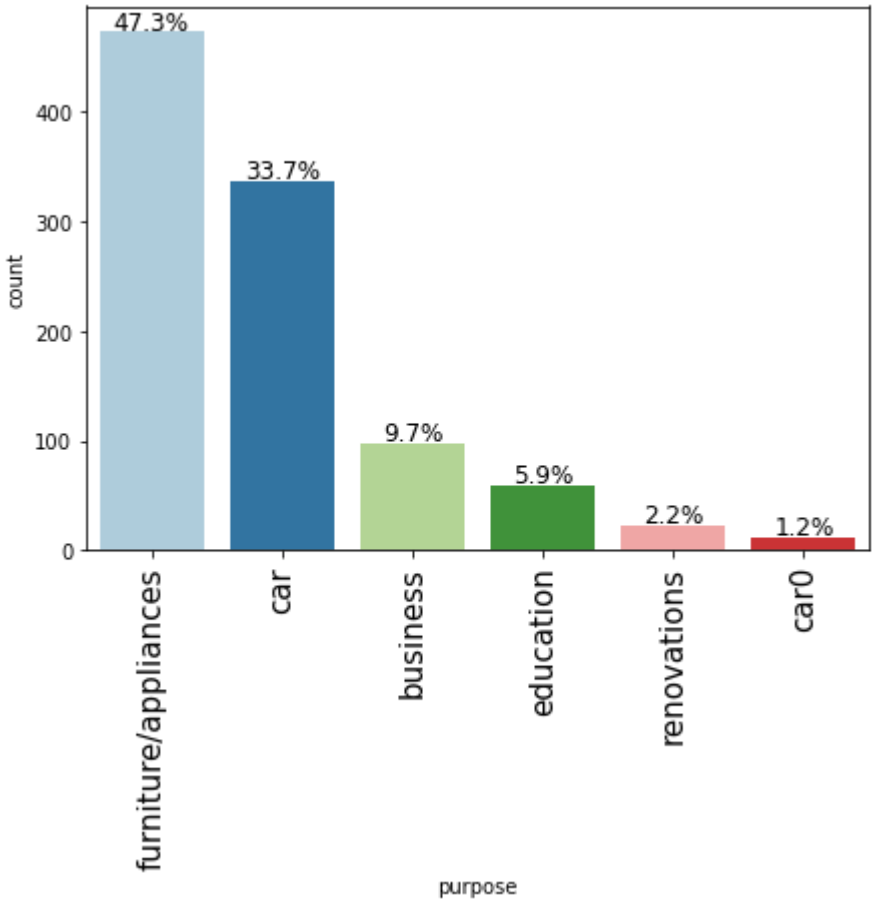
In [87]:
```
data["purpose"].value_counts()
```

Out[87]:
```
furniture/appliances    473
car                     337
business                 97
education                59
renovations              22
car0                     12
Name: purpose, dtype: int64
```

In [88]:
```
# Extracting the Purpose
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'purpose', hatch="/");
```



In [89]:
```
labeled_barplot(data, "purpose", perc=True)
```



In [ ]:

Observations of Savings Balance

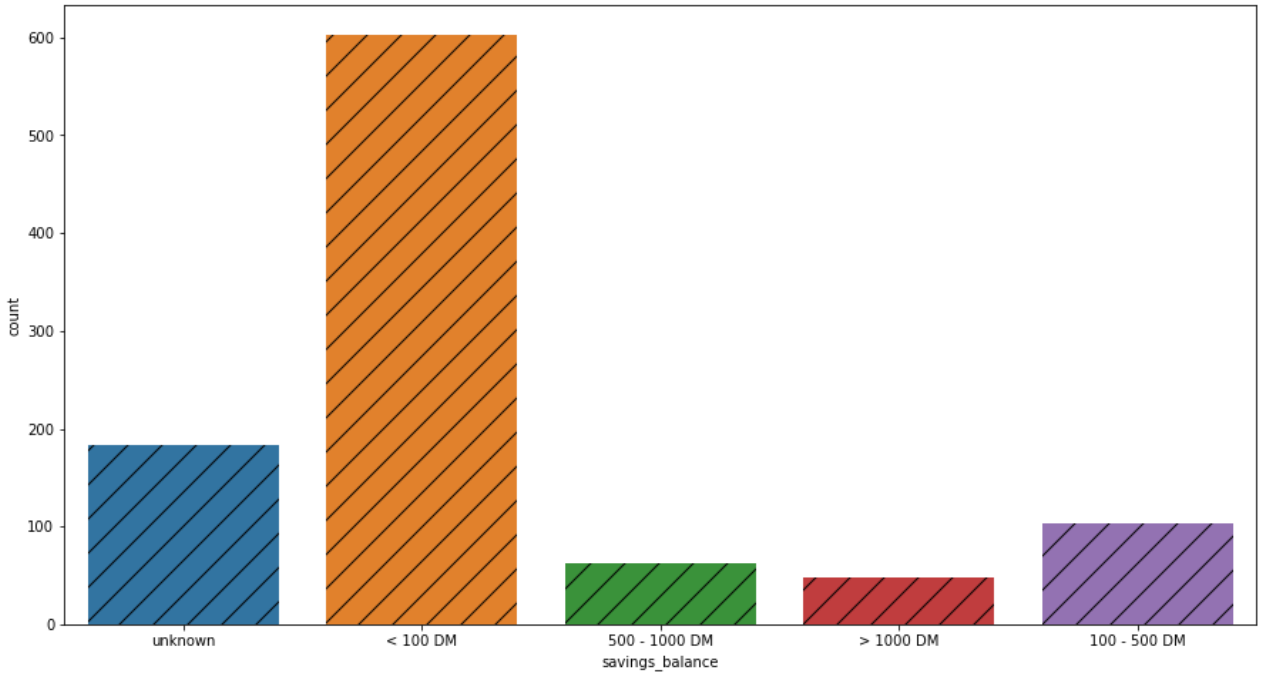
In [90]:
```
data["savings_balance"].value_counts(1)
```

Out[90]:
```
< 100 DM         0.60
unknown          0.18
100 - 500 DM     0.10
500 - 1000 DM    0.06
> 1000 DM        0.05
Name: savings_balance, dtype: float64
```

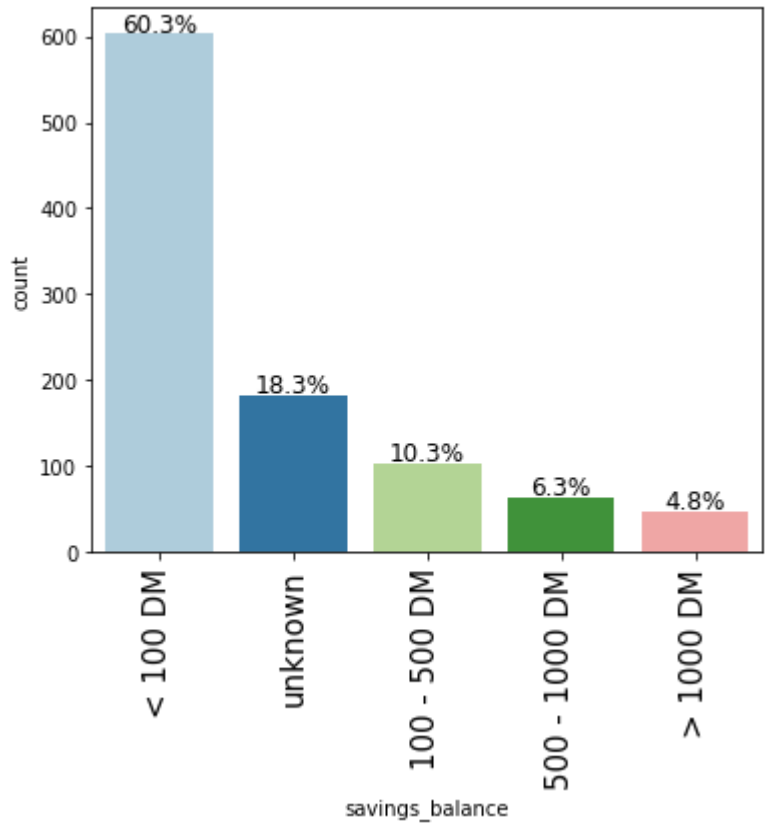
In [91]:
```
data["savings_balance"].value_counts()
```

Out[91]:
```
< 100 DM         603
unknown          183
100 - 500 DM     103
500 - 1000 DM     63
> 1000 DM         48
Name: savings_balance, dtype: int64
```

In [92]:
```
# Extracting the Savings Balance
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'savings_balance', hatch="/");
```



In [93]:
```
labeled_barplot(data, "savings_balance", perc=True)
```



In [ ]:
```

```

Observations of Employment Duration

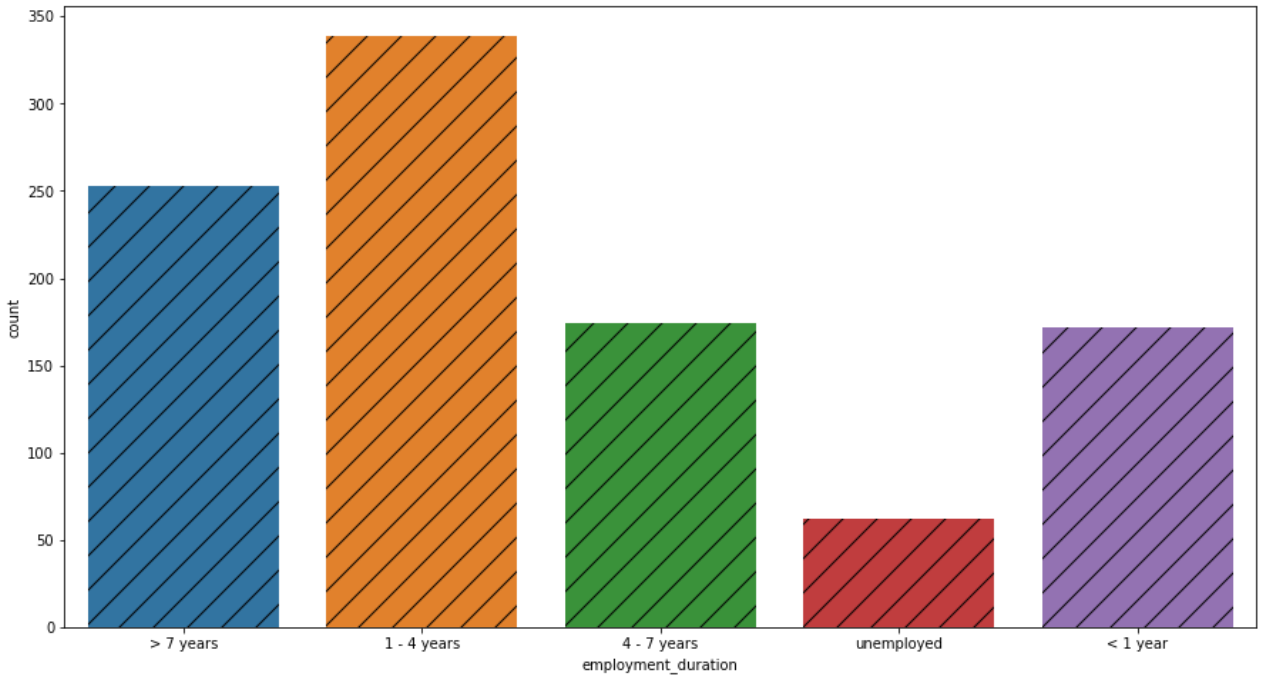
In [94]:
```
data["employment_duration"].value_counts(1)
```

Out[94]:
```
1 - 4 years   0.34
> 7 years     0.25
4 - 7 years   0.17
< 1 year      0.17
unemployed    0.06
Name: employment_duration, dtype: float64
```

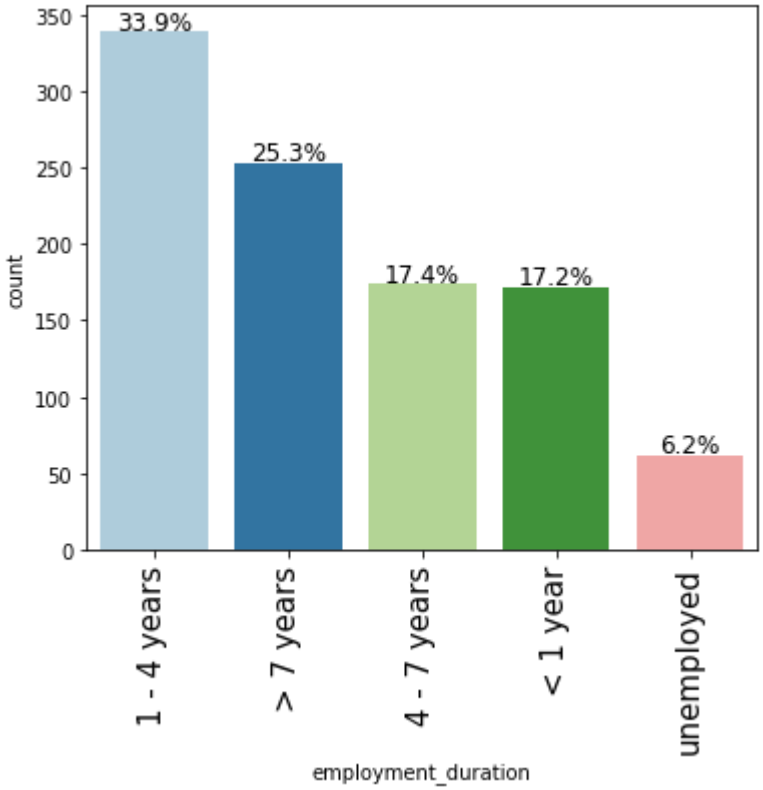
In [95]:
```
data["employment_duration"].value_counts()
```

Out[95]:
```
1 - 4 years    339
> 7 years      253
4 - 7 years    174
< 1 year       172
unemployed      62
Name: employment_duration, dtype: int64
```

In [96]:
```
# Extracting the Employment Duration
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'employment_duration', hatch="/");
```



In [97]:
```
labeled_barplot(data, "employment_duration", perc=True)
```



In [ ]:

Observations of Other Credit

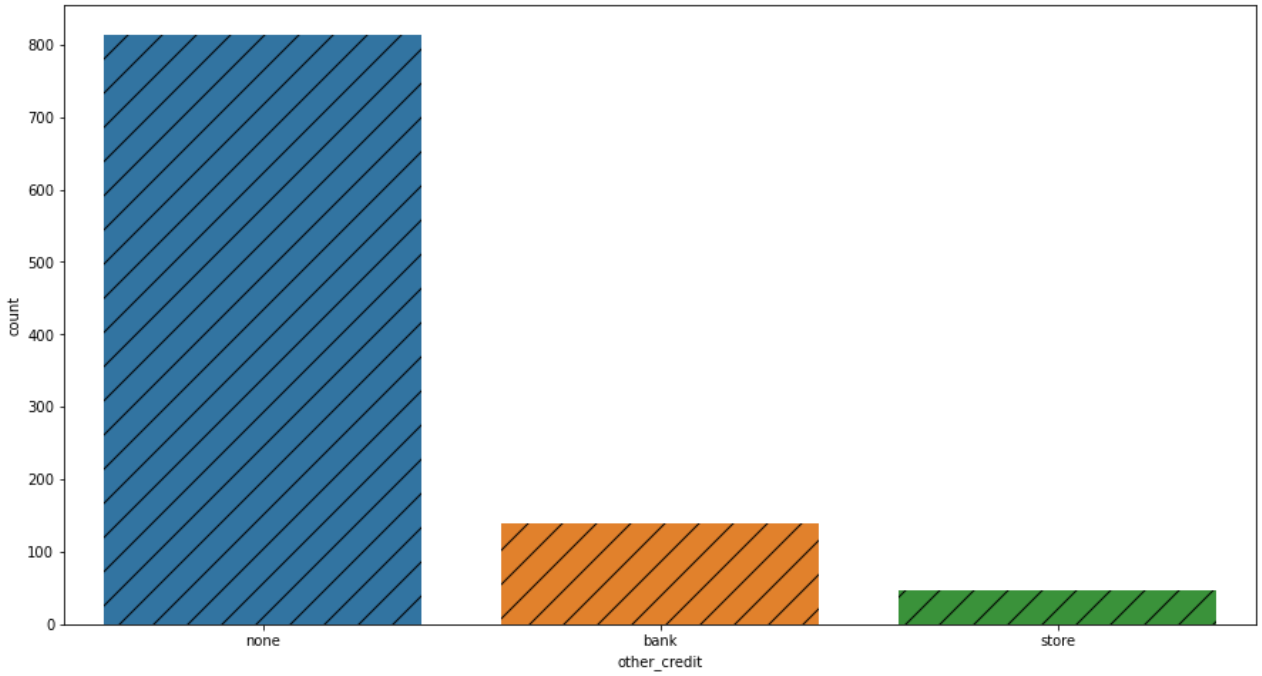
In [98]:
```
data["other_credit"].value_counts(1)
```

Out[98]:
```
none     0.81
bank     0.14
store    0.05
Name: other_credit, dtype: float64
```

In [99]:
```
data["other_credit"].value_counts()
```

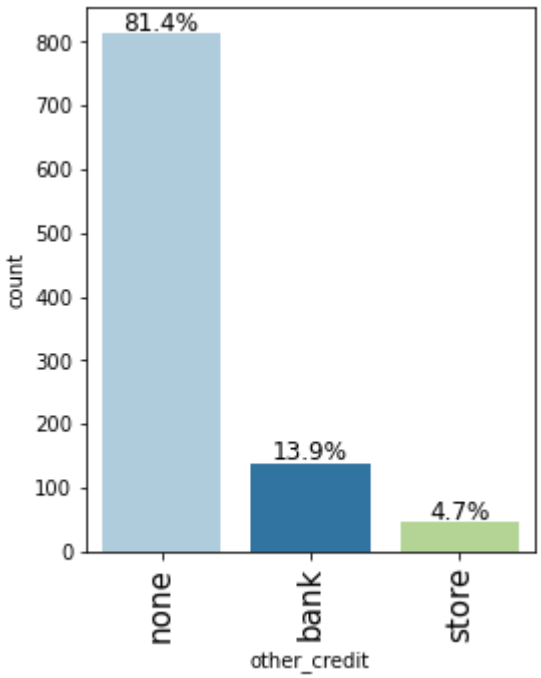
Out[99]:
```
none     814
bank     139
store     47
Name: other_credit, dtype: int64
```

In [100…
```
# Extracting the Other Credits
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'other_credit', hatch="/");
```



In [101…

```
labeled_barplot(data, "other_credit", perc=True)
```



In [ ]:

Observations of Housing

In [102…

```
data["housing"].value_counts(1)
```

Out[102…

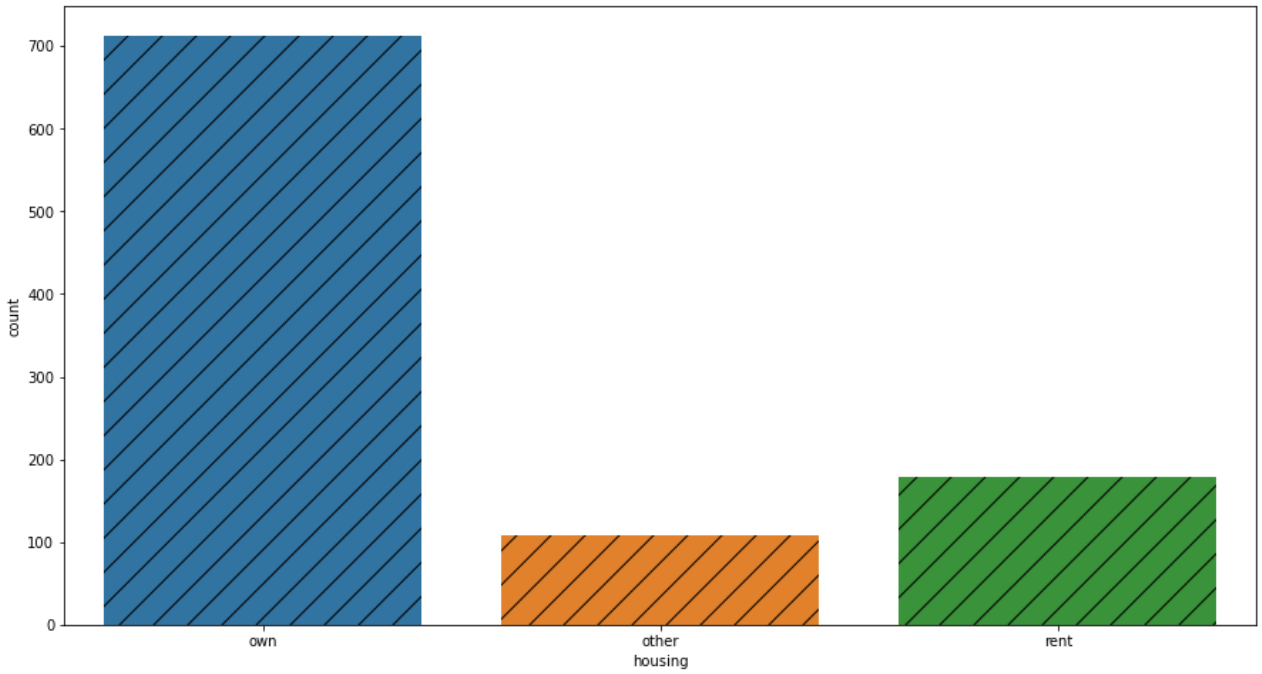
```
own     0.71
rent    0.18
```

```
other   0.11
Name: housing, dtype: float64
```

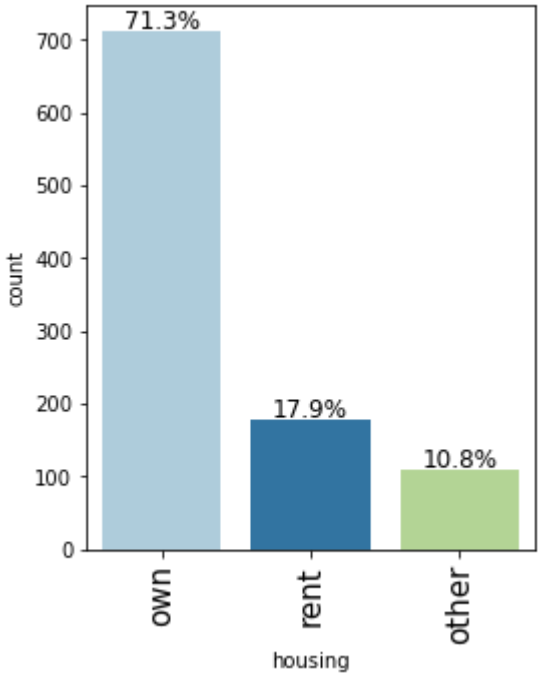
In [103…
```
data["housing"].value_counts()
```

Out[103…
```
own      713
rent     179
other    108
Name: housing, dtype: int64
```

In [104…
```
# Extracting the Housing
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'housing', hatch="/");
```



In [105…
```
labeled_barplot(data, "housing", perc=True)
```



In [ ]:

Observations of job

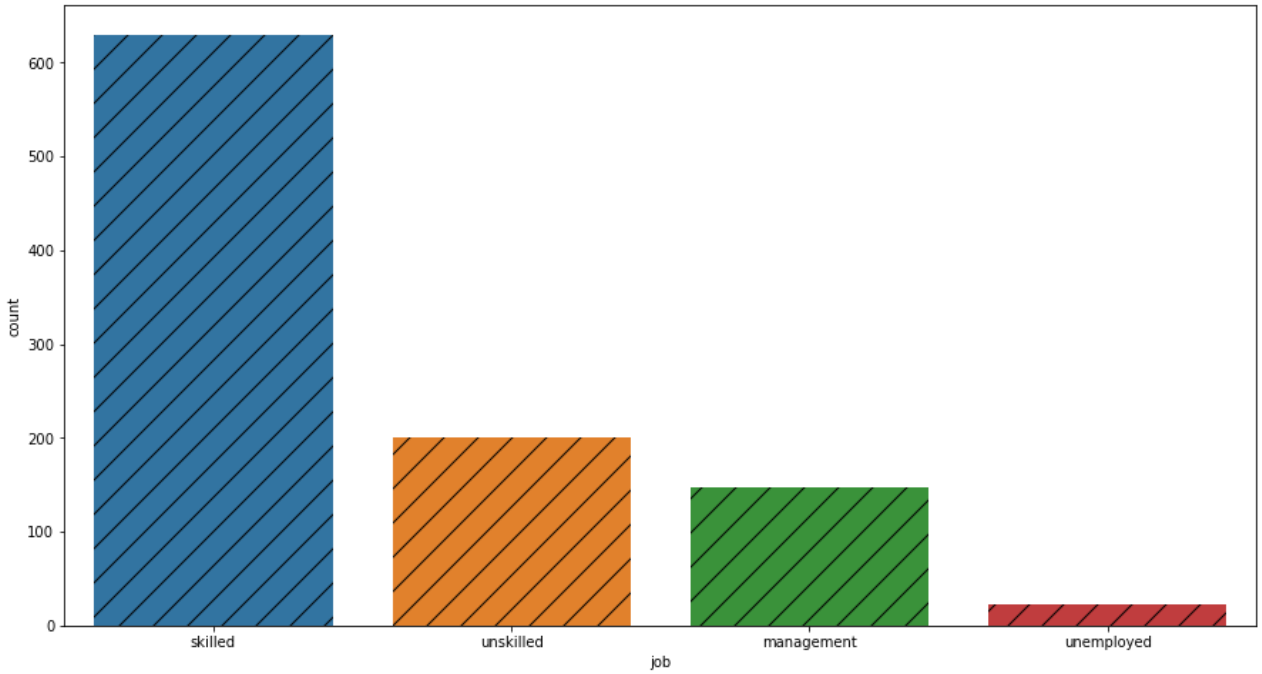
In [106…
```
data["job"].value_counts(1)
```

Out[106…
```
skilled      0.63
unskilled    0.20
management   0.15
unemployed   0.02
Name: job, dtype: float64
```

In [107…
```
data["job"].value_counts()
```

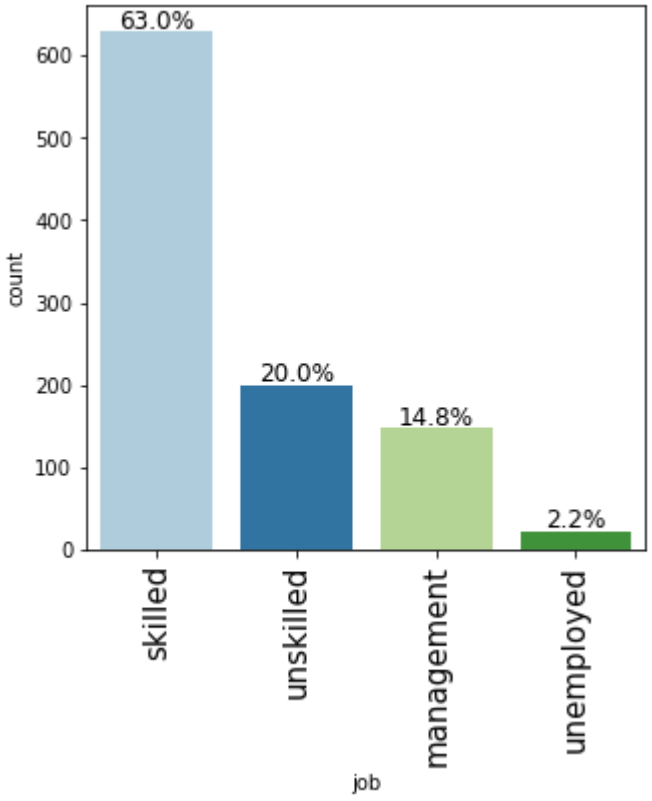
Out[107…
```
skilled       630
unskilled     200
management    148
unemployed     22
Name: job, dtype: int64
```

In [108…
```
# Extracting the Job
plt.subplots(figsize = (15,8))
sns.countplot(data = data, x = 'job', hatch="/");
```



In [109…

```
labeled_barplot(data, "job", perc=True)
```



In [ ]:

Observations of Phone

In [110…

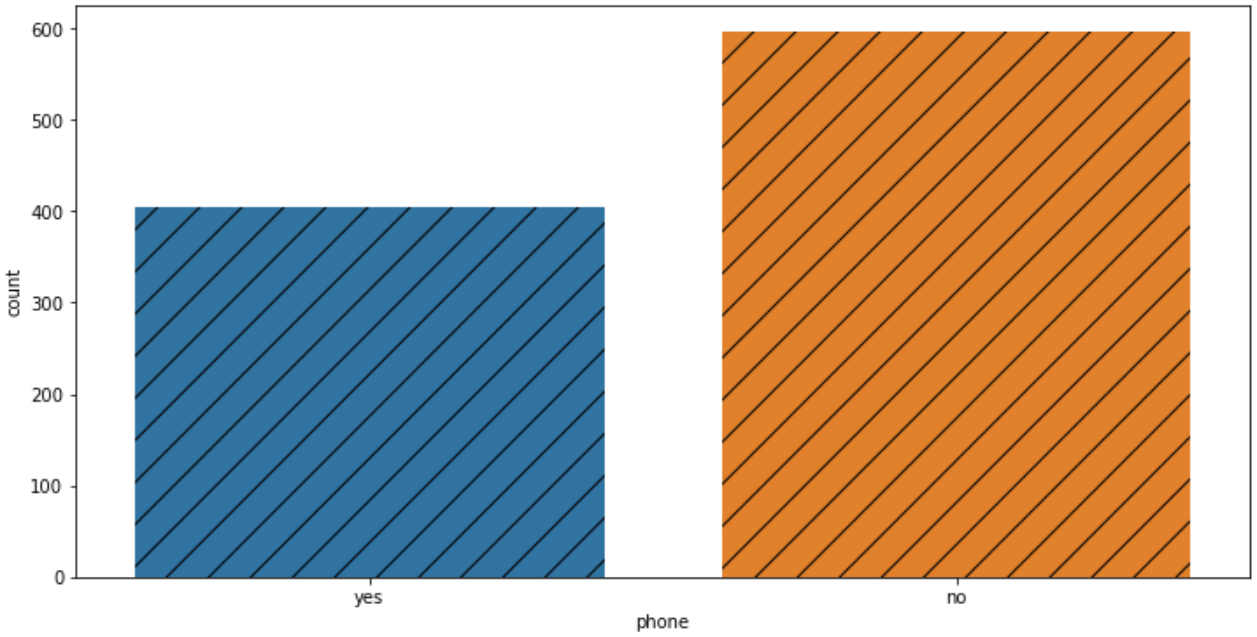
```
data["phone"].value_counts(1)
```

Out[110…
```
no    0.60
yes   0.40
Name: phone, dtype: float64
```

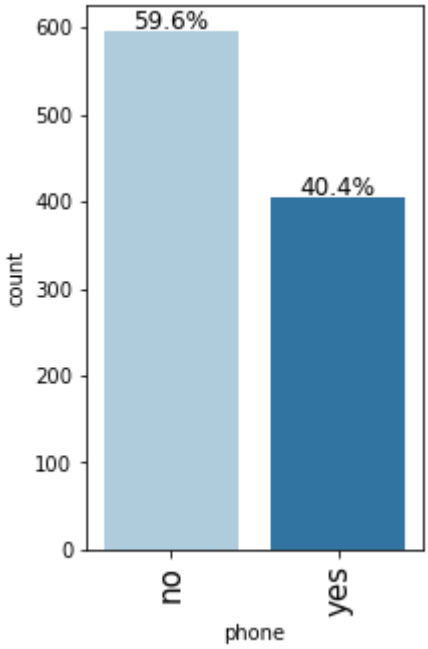
In [111…
```
data["phone"].value_counts()
```

Out[111…
```
no     596
yes    404
Name: phone, dtype: int64
```

In [112…
```
# Extracting the Phone
plt.subplots(figsize = (12,6))
sns.countplot(data = data, x = 'phone', hatch="/");
```



In [113…
```
labeled_barplot(data, "phone", perc=True)
```



In [ ]:

Part III: EDA - Multivariate Data Analysis

Boxplot Comparison Analysis

In [114…

```
def stacked_barplot(data, predictor, target):
    """
    Print the category counts and plot a stacked bar chart

    data: dataframe
    predictor: independent variable
    target: target variable
    """
    count = data[predictor].nunique()
    sorter = data[target].value_counts().index[-1]
    tab1 = pd.crosstab(data[predictor], data[target], margins=True).sort_values(
        by=sorter, ascending=False
    )
    print(tab1)
    print("-" * 120)
    tab = pd.crosstab(data[predictor], data[target], normalize="index").sort_values(
        by=sorter, ascending=False
    )
    tab.plot(kind="bar", stacked=True, figsize=(count + 3, 3))
    plt.legend(
        loc="lower left", frameon=False,
    )
    plt.legend(loc="upper left", bbox_to_anchor=(1, 1))
    plt.show()
```

In [115…

```
### function to plot distributions wrt target


def distribution_plot_wrt_target(data, predictor, target):

    fig, axs = plt.subplots(2, 2, figsize=(25, 9))

    target_uniq = data[target].unique()

    axs[0, 0].set_title("Distribution of target for target=" + str(target_uniq[0]))
    sns.histplot(
        data=data[data[target] == target_uniq[0]],
        x=predictor,
        kde=True,
        ax=axs[0, 0],
        color="teal",
        stat="density",
    )

    axs[0, 1].set_title("Distribution of target for target=" + str(target_uniq[1]))
    sns.histplot(
        data=data[data[target] == target_uniq[1]],
        x=predictor,
        kde=True,
        ax=axs[0, 1],
        color="orange",
        stat="density",
    )

    axs[1, 0].set_title("Boxplot w.r.t target")
    sns.boxplot(data=data, x=target, y=predictor, ax=axs[1, 0], palette="gist_rainbow")

    axs[1, 1].set_title("Boxplot (without outliers) w.r.t target")
    sns.boxplot(
        data=data,
        x=target,
        y=predictor,
        ax=axs[1, 1],
        showfliers=False,
        palette="gist_rainbow",
    )

    plt.tight_layout()
    plt.show()
```

Comparison of the Numerical Columns

In [ ]:

In [116…

```
# Extracting the columns of the entire datasets
data.columns
```

Out[116…

```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
```

```
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

In [117…

```
# Extracting the numerical col of the datasets
col_var = [
    "checking_balance",
    "months_loan_duration",
    "credit_history",
    "purpose",
    "amount",
    "savings_balance",
    "employment_duration",
    "percent_of_income",
    "years_at_residence",
    "age",
    "other_credit",
    "housing",
    "existing_loans_count",
    "job",
    "dependents",
    "phone",
    "default",
]
```

In [118…

```
data[col_var].columns
```

Out[118…

```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

In [119…

```
data.info()
```

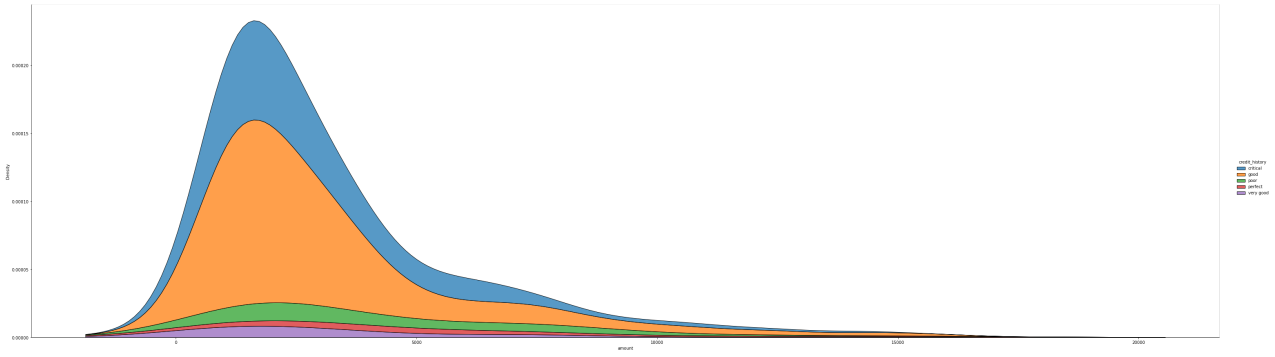
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 17 columns):
 #   Column                Non-Null Count  Dtype 
---  ------                --------------  ----- 
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64 
 2   credit_history        1000 non-null   object
 3   purpose               1000 non-null   object
 4   amount                1000 non-null   int64 
 5   savings_balance       1000 non-null   object
 6   employment_duration   1000 non-null   object
 7   percent_of_income     1000 non-null   int64 
 8   years_at_residence    1000 non-null   int64 
 9   age                   1000 non-null   int64 
 10  other_credit          1000 non-null   object
 11  housing               1000 non-null   object
 12  existing_loans_count  1000 non-null   int64 
 13  job                   1000 non-null   object
```

```
 14  dependents            1000 non-null   int64
 15  phone                 1000 non-null   object
 16  default               1000 non-null   object
dtypes: int64(7), object(10)
memory usage: 132.9+ KB
```

Loan Amount in Relation to Credit History

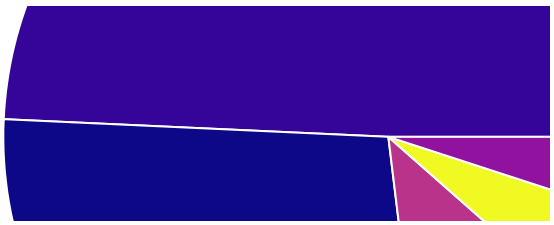
In [120…

```
# Ploting a displot of Loan Amount in Relation to Credit History
sns.displot(
    data=data,
    x="amount",
    hue="credit_history",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



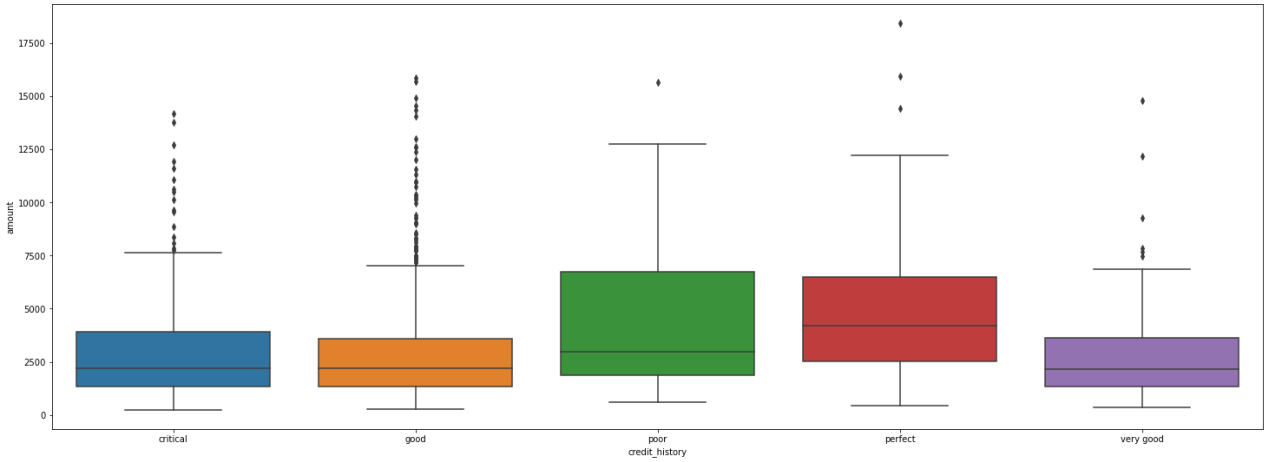
In [121…

```
# Creating a sunburst chart for Loan Amount in Relation to Credit History
import plotly.express as px

fig = px.sunburst(
    data,
    path=["credit_history"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```



In [122…

```
# Boxplot of Loan Amount in Relation to Credit History
plt.figure(figsize=(25, 9))
sns.boxplot(data=data, x="credit_history", y="amount")
```



In [123…

```
# Creating summary statitics pivot table for Loan Amount in Relation to Credit History
amount_credit = data.pivot_table(
    index=["credit_history"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_credit)
```

```
                amount
                   max    mean  median    min     std         var
credit_history
critical       14179.00 3088.04 2181.00 250.00 2502.83  6264150.84
good           15857.00 3040.96 2217.50 276.00 2670.57  7131921.22
perfect        18424.00 5305.68 4193.00 426.00 4268.98 18224193.25
```

```
poor           15653.00 4302.60 2985.50 585.00 3183.53 10134855.00
very good      14782.00 3344.88 2149.00 339.00 3122.87  9752346.11
```

In [124…

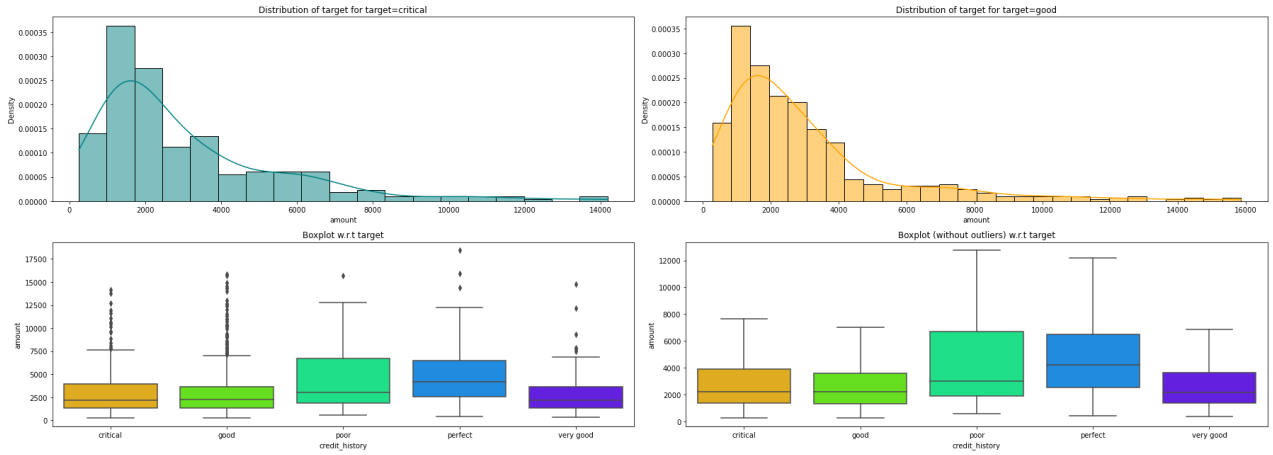
```
# comparing Loan Amount in Relation to Credit History
data.groupby(["credit_history"])["amount"].describe()
```

Out[124…

credit_history	count	mean	std	min	25%	50%	75%	max
critical	293.00	3088.04	2502.83	250.00	1343.00	2181.00	3905.00	14179.00
good	530.00	3040.96	2670.57	276.00	1330.25	2217.50	3598.00	15857.00
perfect	40.00	5305.68	4268.98	426.00	2529.25	4193.00	6496.50	18424.00
poor	88.00	4302.60	3183.53	585.00	1883.00	2985.50	6719.75	15653.00
very good	49.00	3344.88	3122.87	339.00	1358.00	2149.00	3632.00	14782.00

In [125…

```
# Loan Amount in Relation to Credit History
distribution_plot_wrt_target(data, "amount", "credit_history")
```



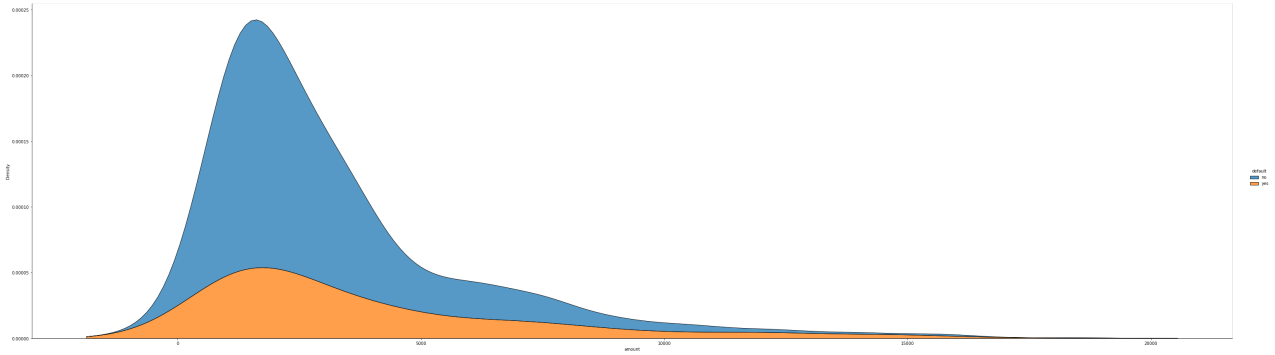
Loan Amount in Relation to Default

In [126…

```
# Ploting a displot of Loan Amount in Relation to Default
sns.displot(
    data=data,
    x="amount",
    hue="default",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```

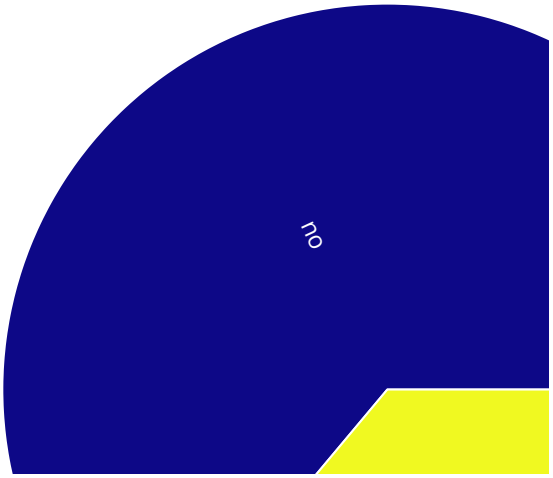
Out[126…

```
<seaborn.axisgrid.FacetGrid at 0x17082fa8a00>
```

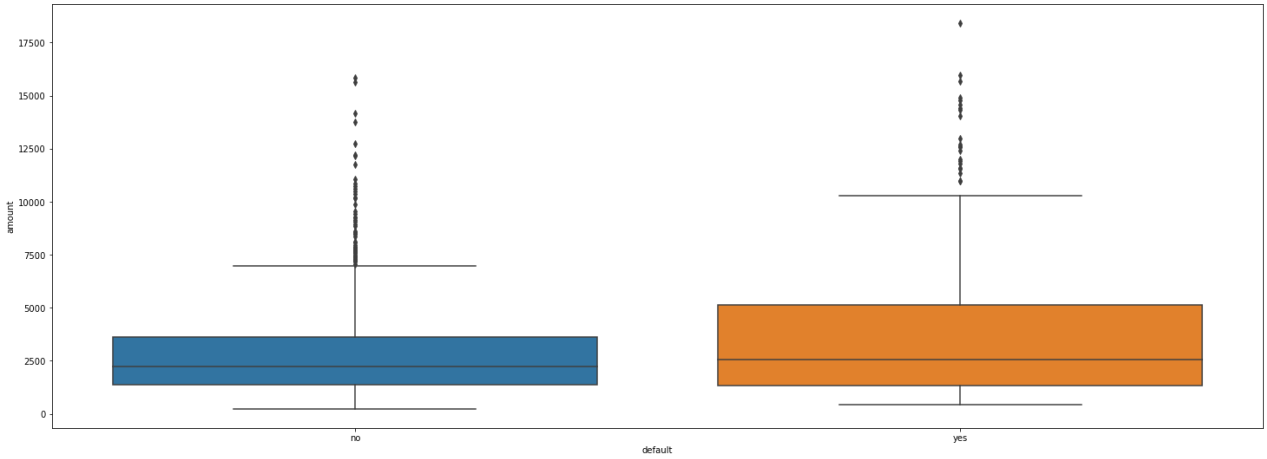


In [127…

```
# Creating a sunburst chart of Loan Amount in Relation to Default
import plotly.express as px

fig = px.sunburst(
    data,
    path=["default"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```



In [128…
```
# Boxplot of Loan Amount in Relation to Default
plt.figure(figsize=(25, 9))
sns.boxplot(data=data, x="default", y="amount")
```



In [129…
```
# Creating summary statitics pivot table for Loan Amount in Relation to Default
amount_default = data.pivot_table(
    index=["default"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_default)
```

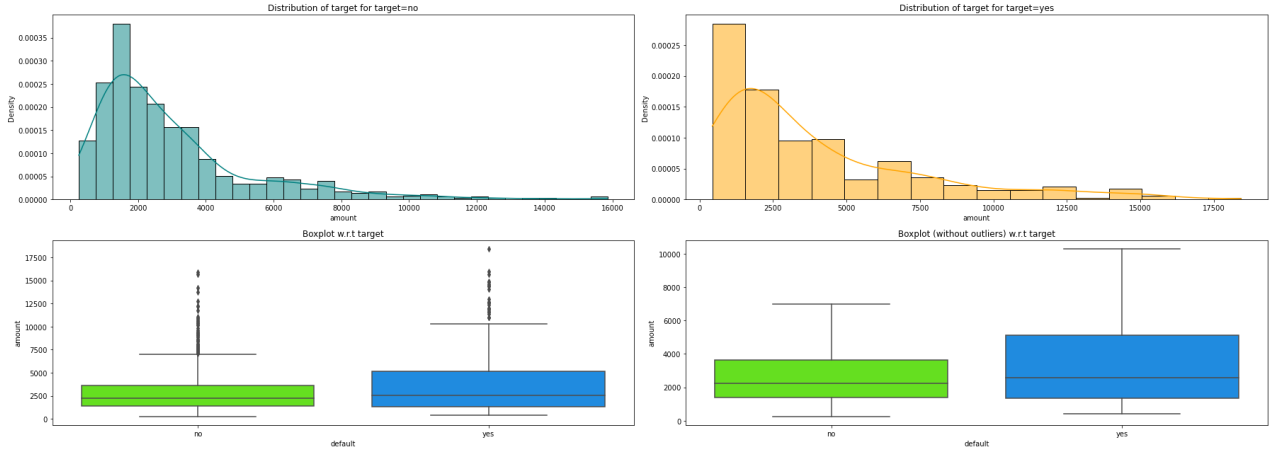
```
           amount
              max    mean  median    min     std         var
default
no       15857.00 2985.46 2244.00 250.00 2401.47  5767069.10
yes      18424.00 3938.13 2574.50 433.00 3535.82 12502015.68
```

In [130…
```
# comparing Loan Amount in Relation to Default
data.groupby(["default"])["amount"].describe()
```

Out[130…

default	count	mean	std	min	25%	50%	75%	max
no	700.00	2985.46	2401.47	250.00	1375.50	2244.00	3634.75	15857.00
yes	300.00	3938.13	3535.82	433.00	1352.50	2574.50	5141.50	18424.00

In [131…
```
# Loan Amount in Relation to Default
distribution_plot_wrt_target(data, "amount", "default")
```



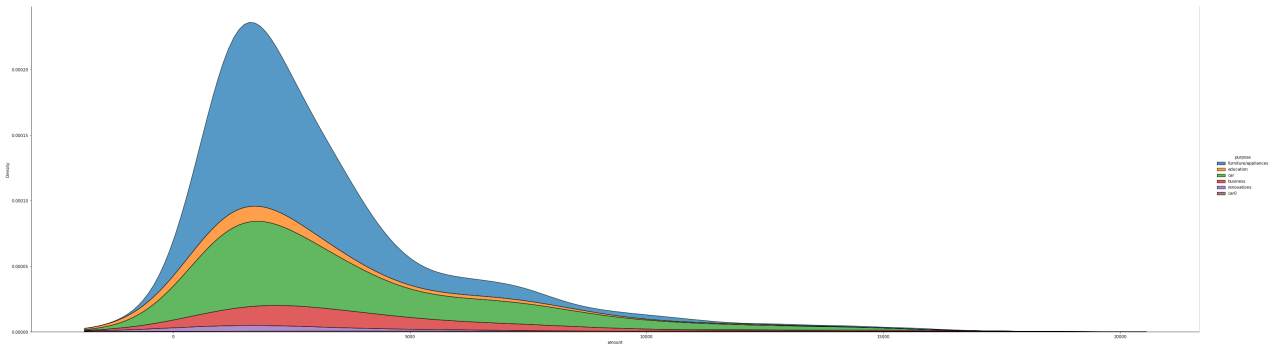
Laon Amount in Relation to Purpose of the Loan

In [132…

```
# Ploting a displot of Laon Amount in Relation to Purpose of the Loan
sns.displot(
    data=data,
    x="amount",
    hue="purpose",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```

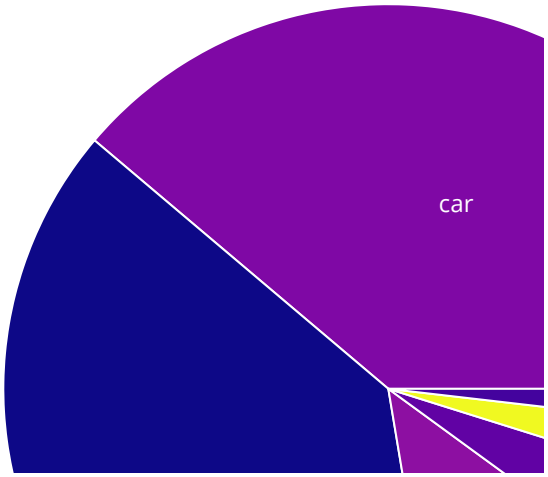
Out[132…

```
<seaborn.axisgrid.FacetGrid at 0x170841891c0>
```



In [133…

```
# Creating a sunburst chart for Laon Amount in Relation to Purpose of the Loan
import plotly.express as px

fig = px.sunburst(
    data,
    path=["purpose"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

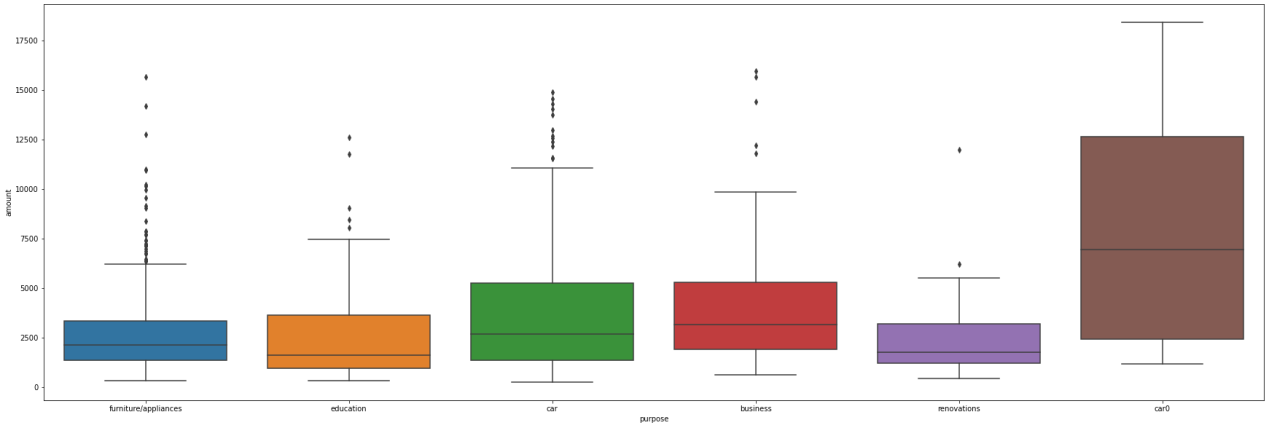


In [134…

```
# Boxplot of Laon Amount in Relation to Purpose of the Loan
plt.figure(figsize=(30, 10))
sns.boxplot(data=data, x="purpose", y="amount")
```

Out[134…

```
<AxesSubplot:xlabel='purpose', ylabel='amount'>
```



In [135…

```
# Creating summary statitics pivot taable for Laon Amount in Relation to Purpose of the
amount_purpose = data.pivot_table(
    index=["purpose"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
```

```
)
print(amount_purpose)
```

```
                      amount
                         max     mean  median     min     std          var
purpose
business             15945.00 4158.04 3161.00  609.00 3231.48 10442491.79
car                  14896.00 3768.19 2679.00  250.00 3123.59  9756842.53
car0                 18424.00 8209.33 6948.00 1164.00 6112.70 37365139.88
education            12612.00 2879.20 1597.00  339.00 2883.92  8316973.03
furniture/appliances 15653.00 2684.24 2116.00  338.00 2063.63  4258584.05
renovations          11998.00 2728.09 1749.00  454.00 2627.49  6903705.61
```

In [136…

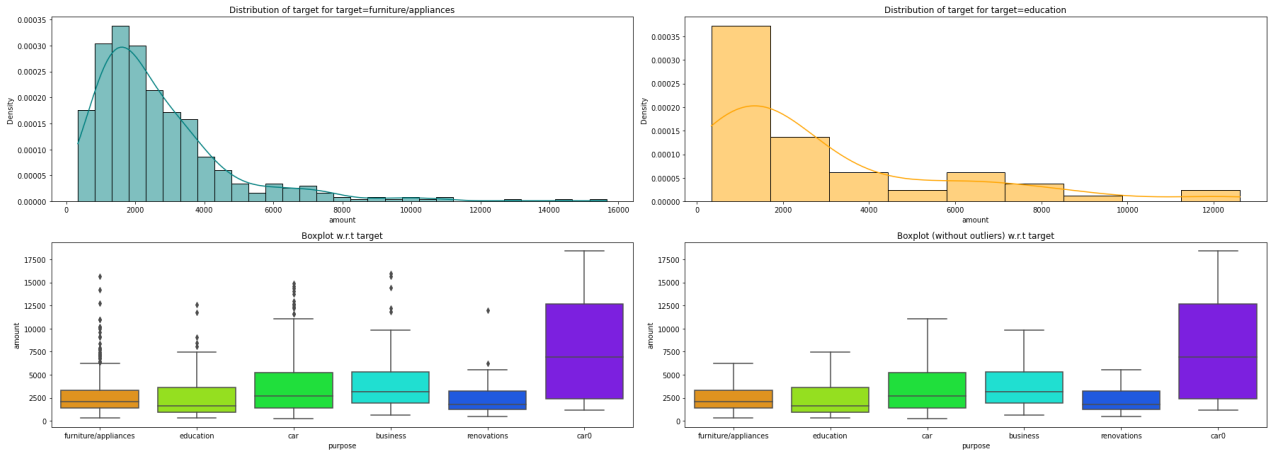
```
# comparing Laon Amount in Relation to Purpose of the Loan
data.groupby(["purpose"])["amount"].describe()
```

Out[136…

purpose	count	mean	std	min	25%	50%	75%	max
business	97.00	4158.04	3231.48	609.00	1908.00	3161.00	5293.00	15945.00
car	337.00	3768.19	3123.59	250.00	1364.00	2679.00	5248.00	14896.00
car0	12.00	8209.33	6112.70	1164.00	2410.50	6948.00	12649.00	18424.00
education	59.00	2879.20	2883.92	339.00	936.50	1597.00	3638.00	12612.00
furniture/appliances	473.00	2684.24	2063.63	338.00	1360.00	2116.00	3345.00	15653.00
renovations	22.00	2728.09	2627.49	454.00	1214.50	1749.00	3203.25	11998.00

In [137…

```
# Laon Amount in Relation to Purpose of the Loan
distribution_plot_wrt_target(data, "amount", "purpose")
```

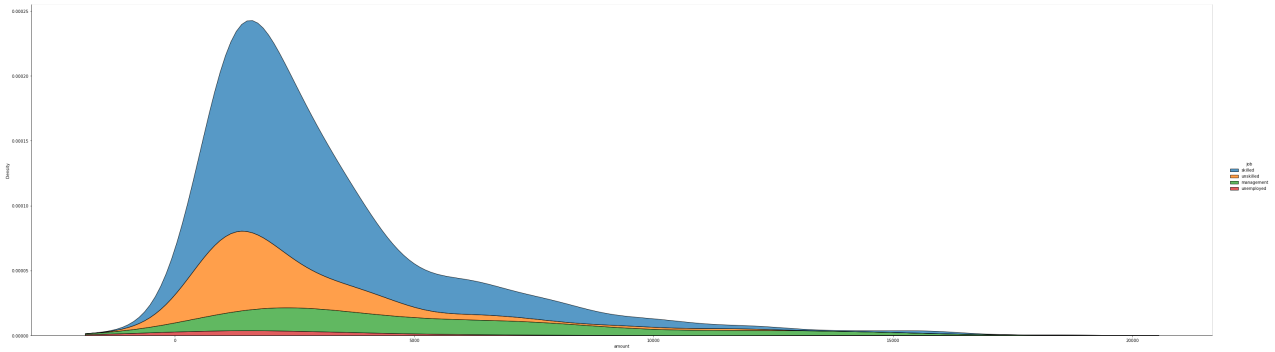


Loan Amount in Relation to Job

In [138…

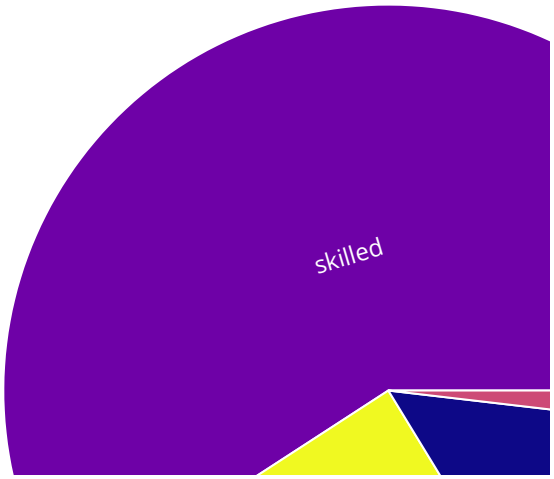
```
# Ploting a displot of  Loan Amount in Relation to Job
sns.displot(
```

```
    data=data,
    x="amount",
    hue="job",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



In [139…

```
# Creating a sunburst chart for  Loan Amount in Relation to Job
import plotly.express as px

fig = px.sunburst(
    data,
    path=["job"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

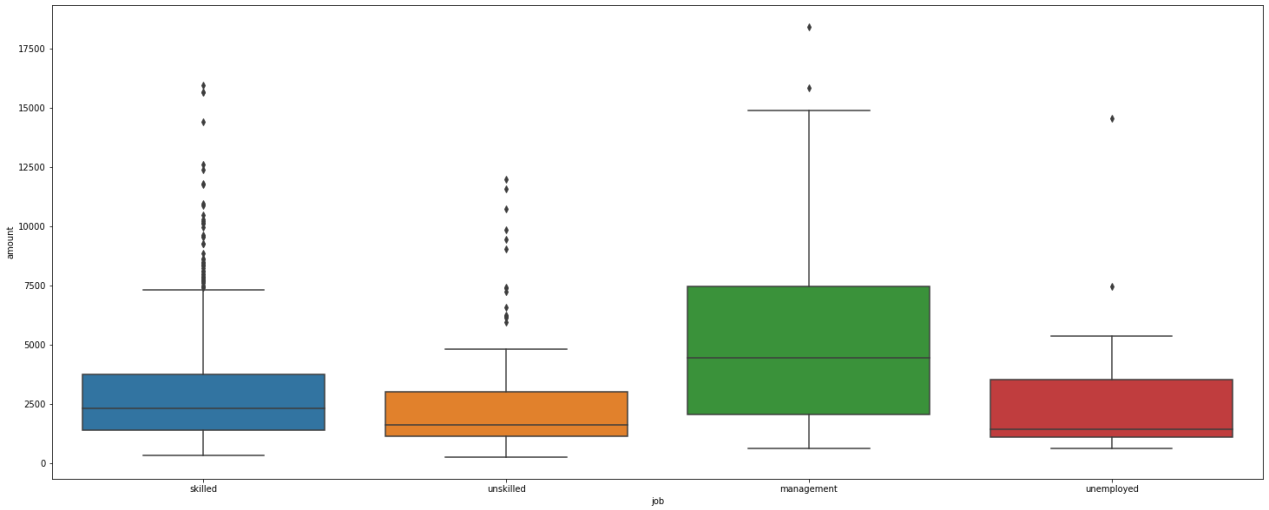


In [140…

```
#  Boxplot of  Loan Amount in Relation to Job
plt.figure(figsize=(25, 10))
sns.boxplot(data=data, x="job", y="amount")
```

Out[140…

```
<AxesSubplot:xlabel='job', ylabel='amount'>
```



In [141…

```
# Creating summary statitics pivot taable for Loan Amount in Relation to Job
amount_job = data.pivot_table(
    index=["job"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_job)
```

```
             amount
                max    mean  median    min     std         var
job
management 18424.00 5435.49 4459.00 629.00 3914.66 15324592.20
skilled    15945.00 3070.97 2324.00 338.00 2444.54  5975796.44
unemployed 14555.00 2745.14 1416.00 609.00 3168.67 10040465.17
unskilled  11998.00 2358.52 1622.00 250.00 2062.26  4252933.95
```

In [142…

```
# comparing Loan Amount in Relation to Job
data.groupby(["job"])["amount"].describe()
```

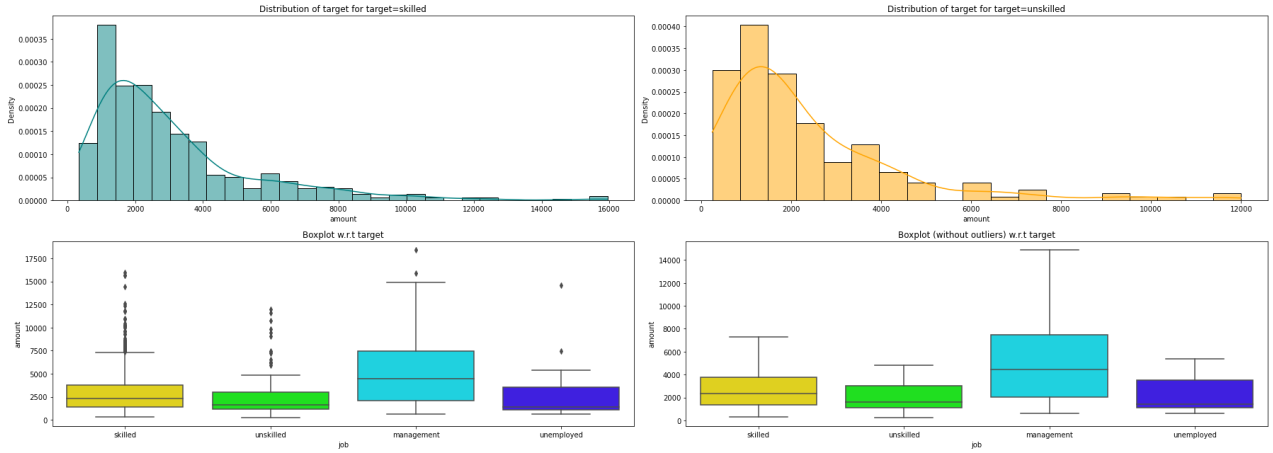
Out[142…

job	count	mean	std	min	25%	50%	75%	max
management	148.00	5435.49	3914.66	629.00	2063.75	4459.00	7478.25	18424.00

job	count	mean	std	min	25%	50%	75%	max
skilled	630.00	3070.97	2444.54	338.00	1386.00	2324.00	3755.00	15945.00
unemployed	22.00	2745.14	3168.67	609.00	1117.25	1416.00	3541.00	14555.00
unskilled	200.00	2358.52	2062.26	250.00	1134.25	1622.00	3028.00	11998.00

In [143…

```
# Loan Amount in Relation to Job
distribution_plot_wrt_target(data, "amount", "job")
```



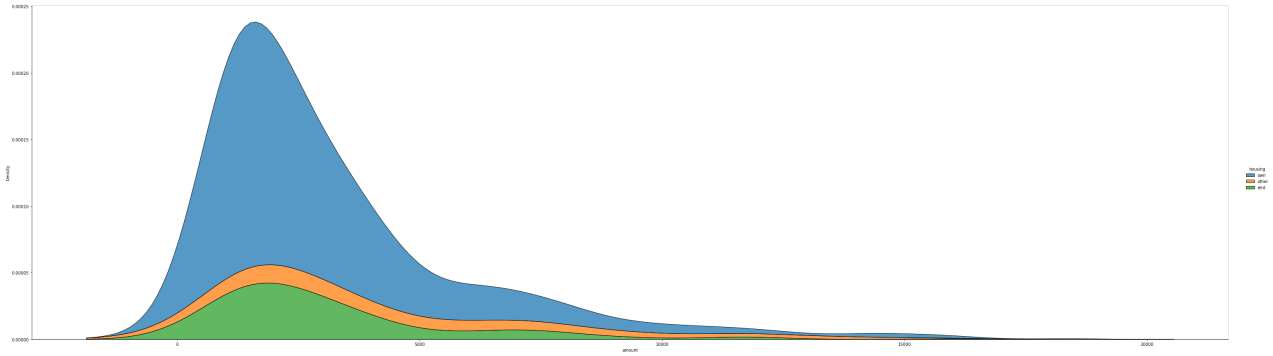
Loan Amount in Relation to Housing

In [144…

```
# Ploting a displot of Loan Amount in Relation to Housing
sns.displot(
    data=data,
    x="amount",
    hue="housing",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```

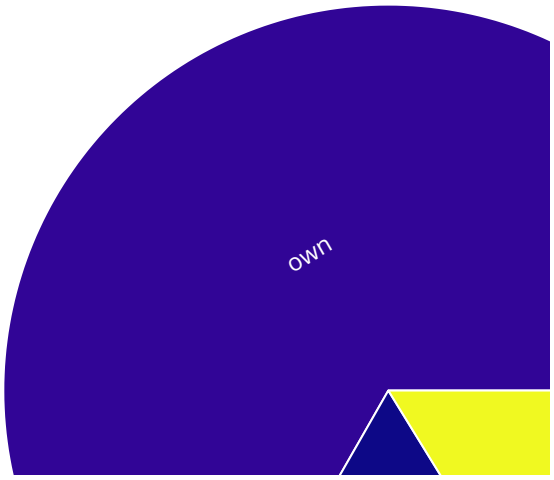
Out[144…

```
<seaborn.axisgrid.FacetGrid at 0x17082da8b50>
```



In [145…

```
# Creating a sunburst chart for Loan Amount in Relation to Housing
import plotly.express as px

fig = px.sunburst(
    data,
    path=["housing"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

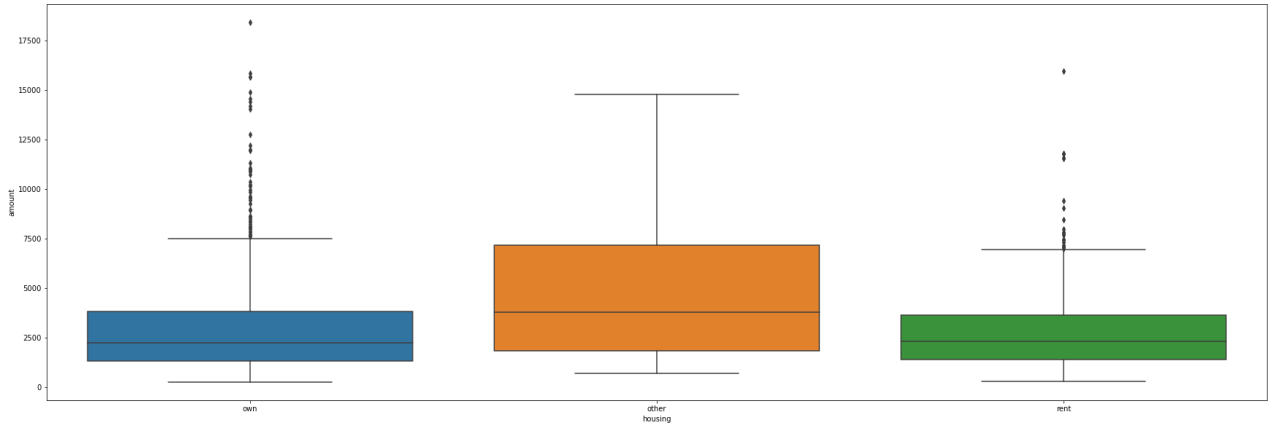


In [146…

```
# Boxplot of Loan Amount in Relation to Housing
plt.figure(figsize=(30, 10))
sns.boxplot(data=data, x="housing", y="amount")
```

Out[146…

```
<AxesSubplot:xlabel='housing', ylabel='amount'>
```



In [147…

```
# Creating summary statitics pivot taable for Loan Amount in Relation to Housing
amount_housing = data.pivot_table(
    index=["housing"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_housing)
```

```
          amount
             max    mean  median    min     std         var
housing
other   14782.00 4906.21 3800.50 700.00 3667.00 13446917.01
own     18424.00 3060.94 2238.00 250.00 2659.58  7073348.80
rent    15945.00 3122.55 2301.00 276.00 2553.52  6520459.64
```

In [148…

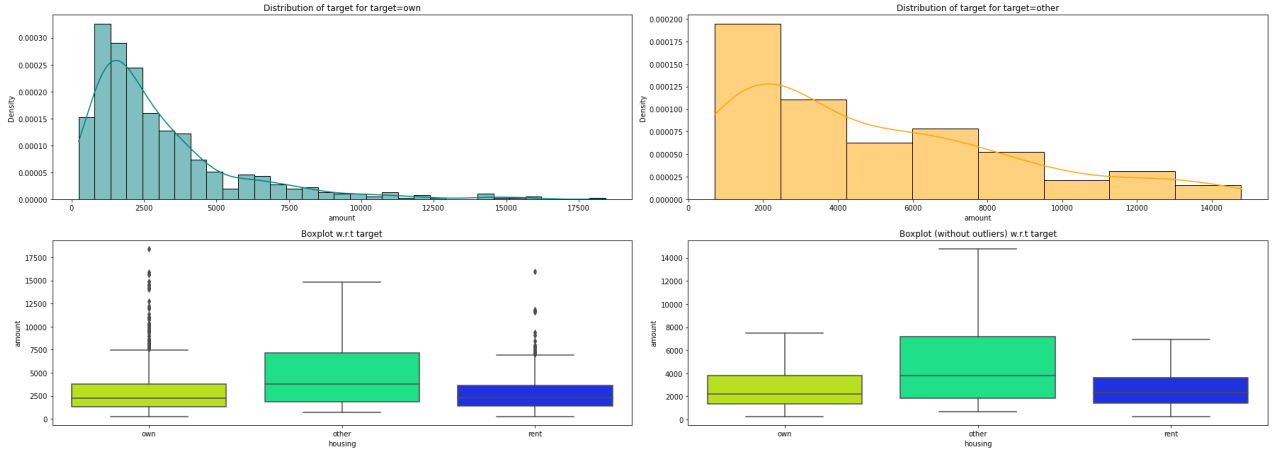
```
# comparing Loan Amount in Relation to Housing
data.groupby(["housing"])["amount"].describe()
```

Out[148…

housing	count	mean	std	min	25%	50%	75%	max
other	108.00	4906.21	3667.00	700.00	1843.00	3800.50	7166.25	14782.00
own	713.00	3060.94	2659.58	250.00	1323.00	2238.00	3804.00	18424.00
rent	179.00	3122.55	2553.52	276.00	1402.50	2301.00	3627.00	15945.00

In [149…

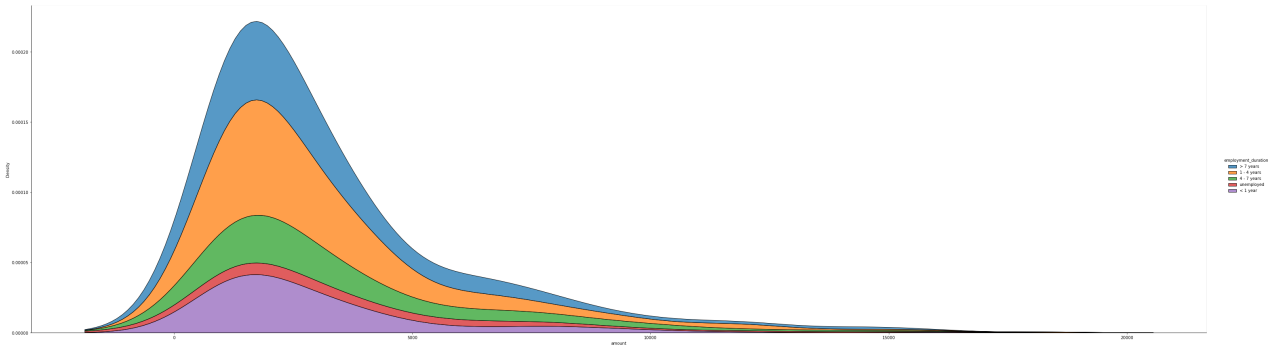
```
# Loan Amount in Relation to Housing
distribution_plot_wrt_target(data, "amount", "housing")
```



Loan Amount in Relation to Employment Duration

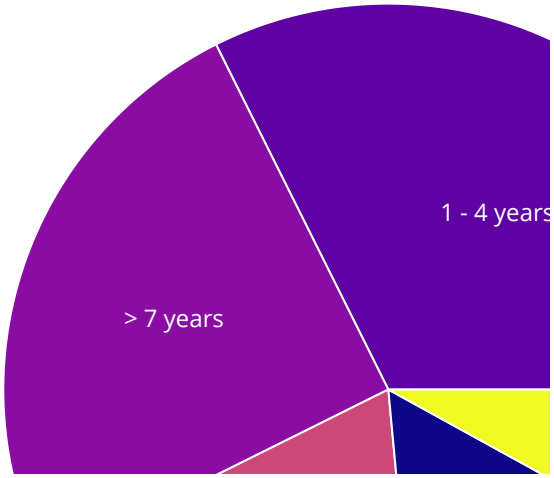
In [150…

```
# Ploting a displot of Loan Amount in Relation to Employment Duration
sns.displot(
    data=data,
    x="amount",
    hue="employment_duration",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



In [151…

```
# Creating a sunburst chart for Loan Amount in Relation to Employment Duration
import plotly.express as px

fig = px.sunburst(
    data,
    path=["employment_duration"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

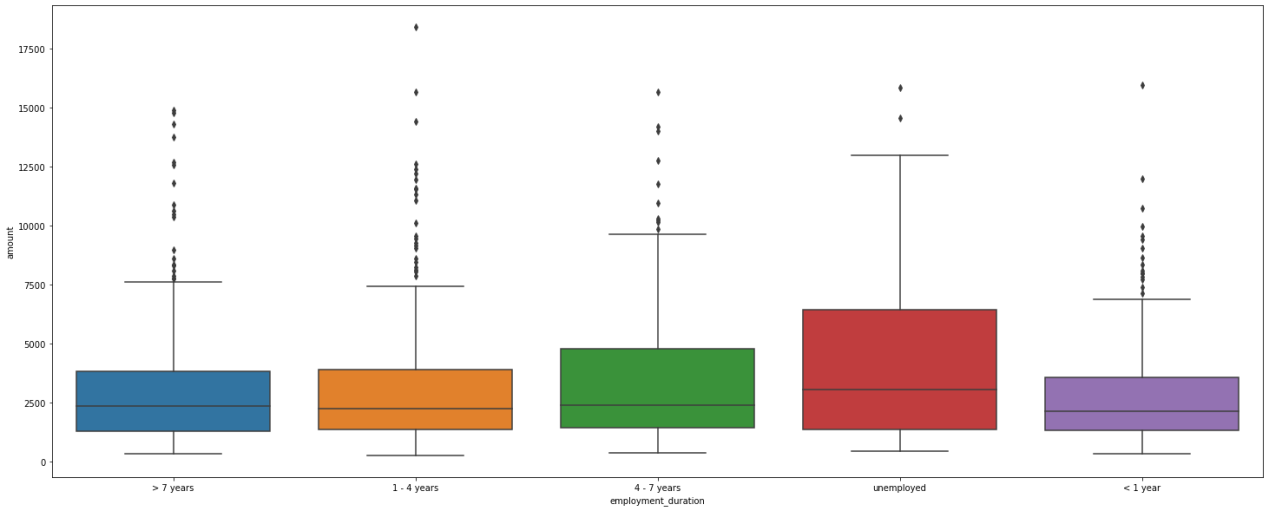


In [152…

```
# Boxplot of Loan Amount in Relation to Employment Duration
plt.figure(figsize=(25, 10))
sns.boxplot(data=data, x="employment_duration", y="amount")
```

Out[152…

```
<AxesSubplot:xlabel='employment_duration', ylabel='amount'>
```



In [153…

```
# Creating summary statitics pivot taable for  Loan Amount in Relation to Employment Du
amount_emp = data.pivot_table(
    index=["employment_duration"],
    values=["amount"],
```

```
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_emp)
```

```
                      amount
                         max     mean  median     min      std          var
employment_duration
1 - 4 years          18424.00 3125.29 2235.00 250.00 2687.82  7224383.63
4 - 7 years          15653.00 3601.70 2386.00 385.00 3025.70  9154845.90
< 1 year             15945.00 2952.45 2139.00 343.00 2498.49  6242442.46
> 7 years            14896.00 3224.62 2337.00 338.00 2809.15  7891327.92
unemployed           15857.00 4216.76 3044.00 433.00 3572.42 12762185.50
```

In [154…

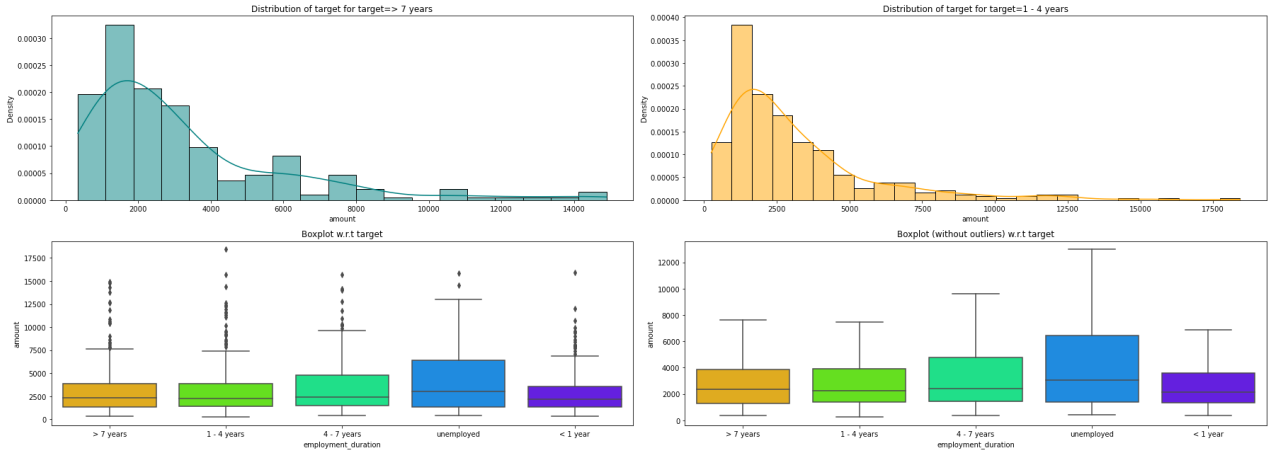
```
# comparing Loan Amount in Relation to Employment Duration
data.groupby(["employment_duration"])["amount"].describe()
```

Out[154…

employment_duration	count	mean	std	min	25%	50%	75%	max
1 - 4 years	339.00	3125.29	2687.82	250.00	1377.50	2235.00	3884.00	18424.00
4 - 7 years	174.00	3601.70	3025.70	385.00	1450.50	2386.00	4783.75	15653.00
< 1 year	172.00	2952.45	2498.49	343.00	1335.50	2139.00	3572.25	15945.00
> 7 years	253.00	3224.62	2809.15	338.00	1300.00	2337.00	3832.00	14896.00
unemployed	62.00	4216.76	3572.42	433.00	1357.50	3044.00	6429.50	15857.00

In [155…

```
# Loan Amount in Relation to Employment Duration
distribution_plot_wrt_target(data, "amount", "employment_duration")
```

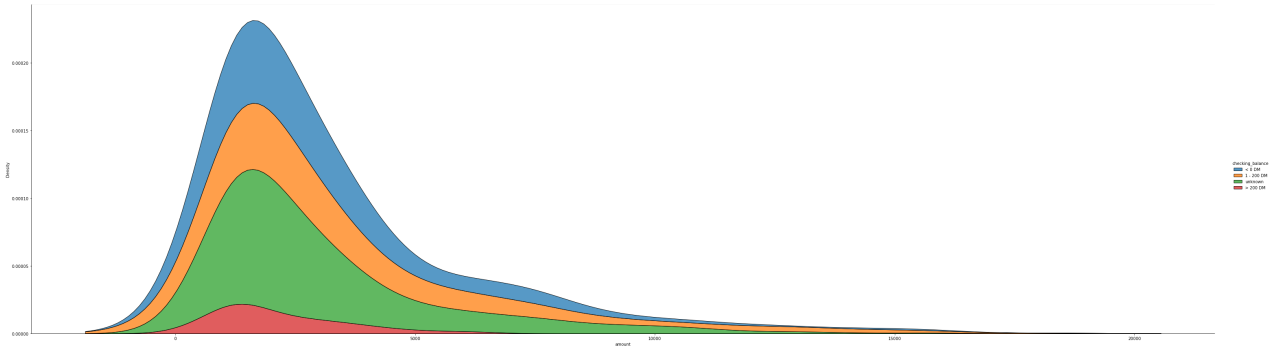


In [ ]:

Loan Amount in Relation to Checking Balance

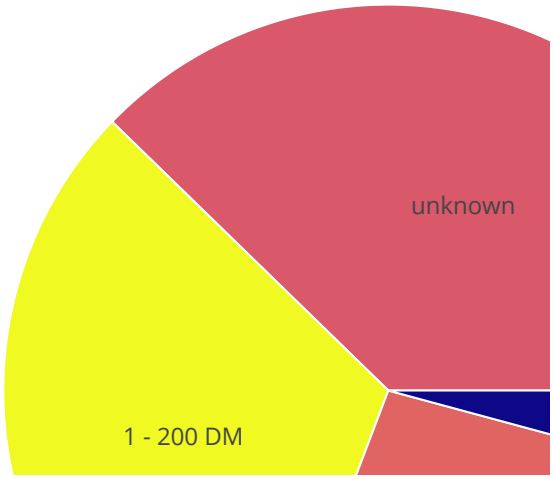
In [156…

```
# Ploting a displot of Loan Amount in Relation to Checking Balance
sns.displot(
    data=data,
    x="amount",
    hue="checking_balance",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



In [157…

```
# Creating a sunburst chart for Loan Amount in Relation to Checking Balance
import plotly.express as px

fig = px.sunburst(
    data,
    path=["checking_balance"],
    values="amount",
    color="amount",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

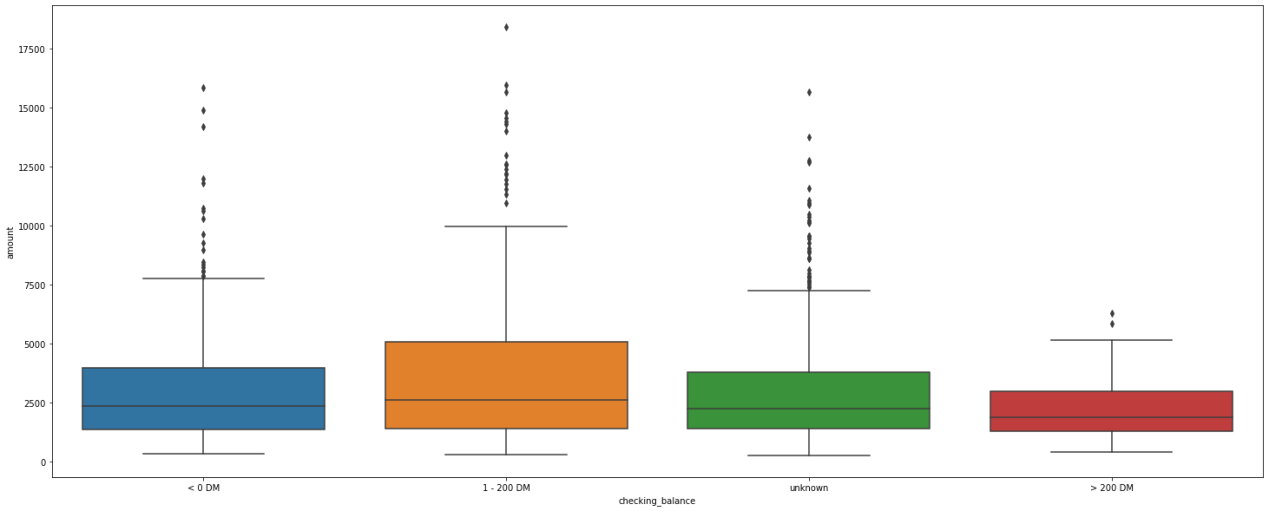


In [158…

```
# Boxplot of Loan Amount in Relation to Checking Balance
plt.figure(figsize=(25, 10))
sns.boxplot(data=data, x="checking_balance", y="amount")
```

Out[158…

```
<AxesSubplot:xlabel='checking_balance', ylabel='amount'>
```



In [159…

```
# Creating summary statitics pivot taable for Loan Amount in Relation to Checking Balan
amount_check = data.pivot_table(
    index=["checking_balance"],
    values=["amount"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(amount_check)
```

```
                  amount
                     max    mean  median    min     std         var
checking_balance
1 - 200 DM       18424.00 3827.56 2622.00 276.00 3465.20 12007639.40
< 0 DM           15857.00 3175.22 2353.50 338.00 2636.38  6950520.91
> 200 DM          6289.00 2177.65 1881.00 392.00 1343.19  1804156.20
unknown          15653.00 3133.10 2248.00 250.00 2554.16  6523755.47
```

In [160…

```
# comparing Loan Amount in Relation to Checking Balance
data.groupby(["checking_balance"])["amount"].describe()
```

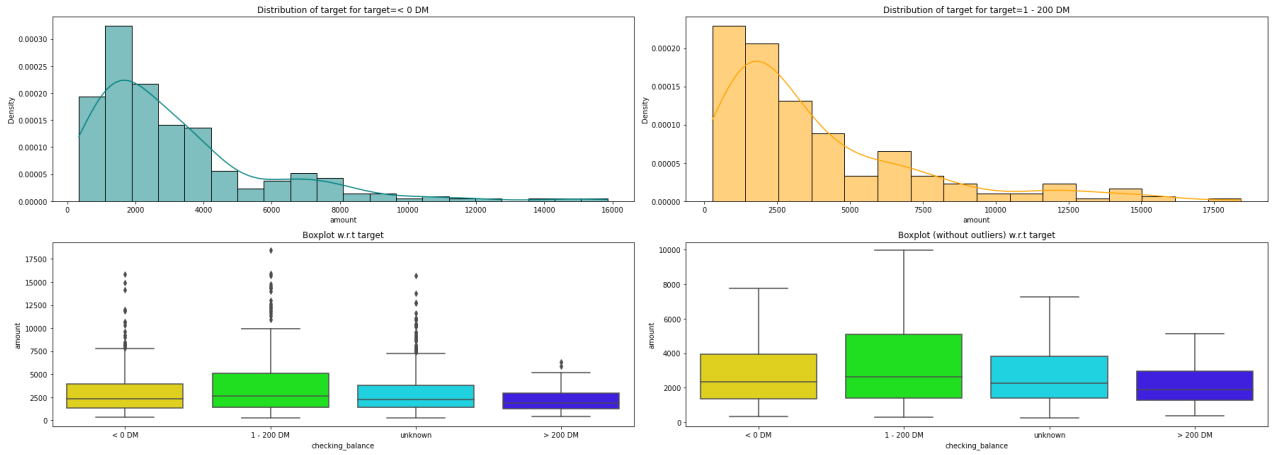
Out[160…

checking_balance	count	mean	std	min	25%	50%	75%	max

checking_balance	count	mean	std	min	25%	50%	75%	max
1 - 200 DM	269.00	3827.56	3465.20	276.00	1391.00	2622.00	5084.00	18424.00
< 0 DM	274.00	3175.22	2636.38	338.00	1353.50	2353.50	3954.00	15857.00
> 200 DM	63.00	2177.65	1343.19	392.00	1275.00	1881.00	2969.50	6289.00
unknown	394.00	3133.10	2554.16	250.00	1414.25	2248.00	3804.00	15653.00

In [161…

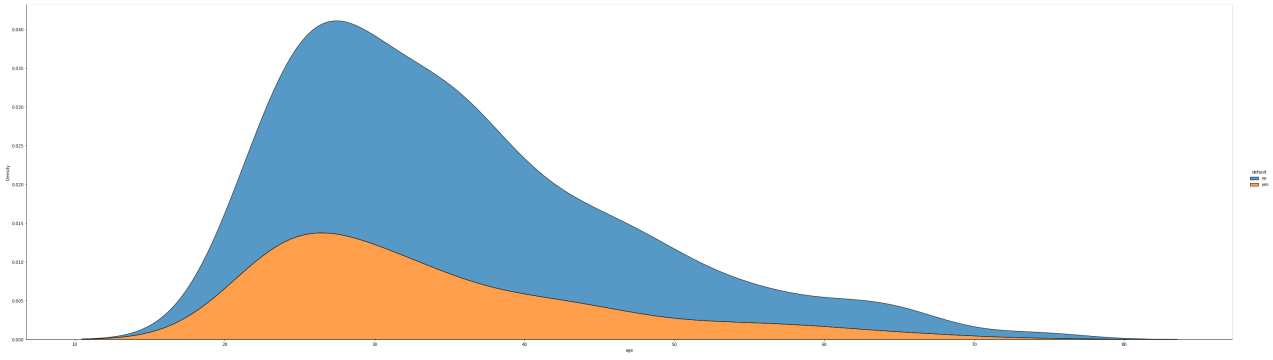
```
# Loan Amount in Relation to Checking Balance
distribution_plot_wrt_target(data, "amount", "checking_balance")
```



Age in Relation to Default

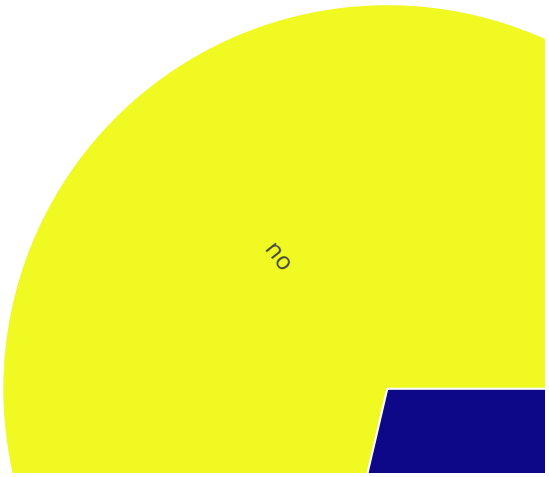
In [162…

```
# Ploting a displot of Age vs Default
sns.displot(
    data=data,
    x="age",
    hue="default",
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



In [163…

```
# Creating a sunburst chart for Age vs Default
import plotly.express as px

fig = px.sunburst(
    data,
    path=["default"],
    values="age",
    color="age",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

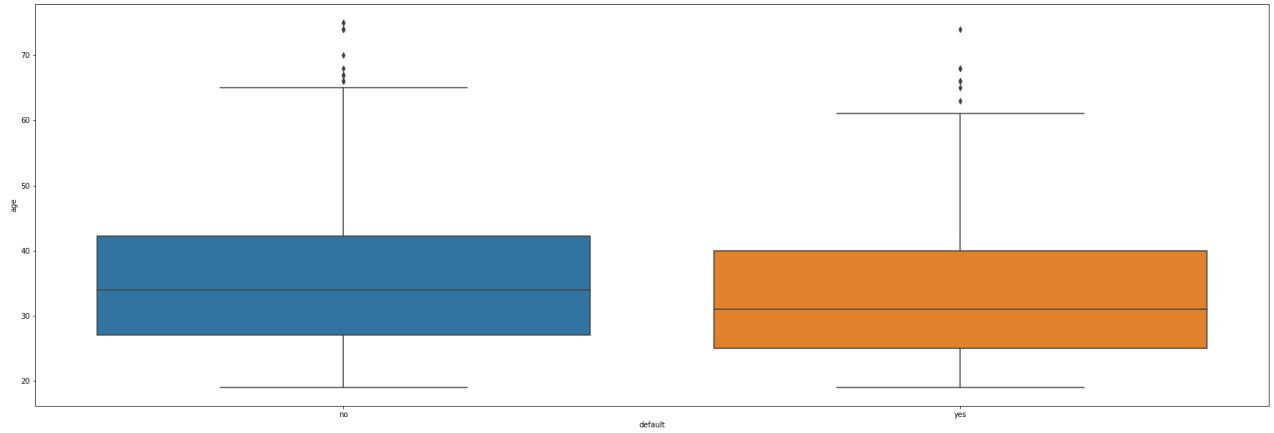


In [164…

```
# Boxplot of Age vs Default
plt.figure(figsize=(30, 10))
sns.boxplot(data=data, x="default", y="age")
```

Out[164…

```
<AxesSubplot:xlabel='default', ylabel='age'>
```



In [165…

```
# Creating summary statitics pivot table for Age vs Default
Age_default = data.pivot_table(
    index=["default"],
    values=["age"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(Age_default)
```

```
          age
          max  mean median   min   std    var
default
no      75.00 36.22  34.00 19.00 11.38 129.53
yes     74.00 33.96  31.00 19.00 11.22 125.94
```

In [166…

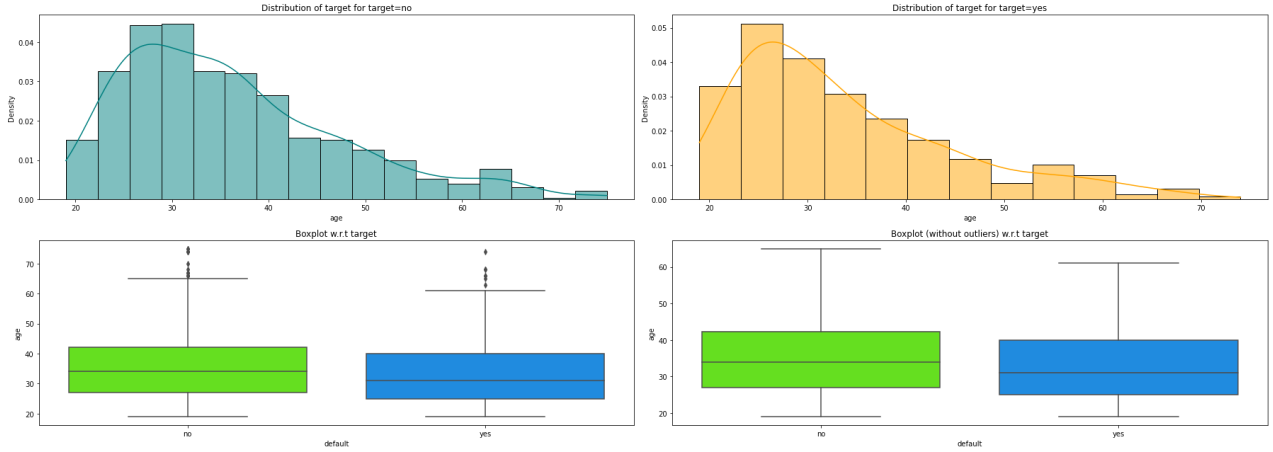
```
# comparing Age vs Default
data.groupby(["default"])["age"].describe()
```

Out[166…

default	count	mean	std	min	25%	50%	75%	max
no	700.00	36.22	11.38	19.00	27.00	34.00	42.25	75.00
yes	300.00	33.96	11.22	19.00	25.00	31.00	40.00	74.00

In [167…

```
distribution_plot_wrt_target(
    data, "age", "default"
)  ## Complete the code to find distribution of Age vs Default
```

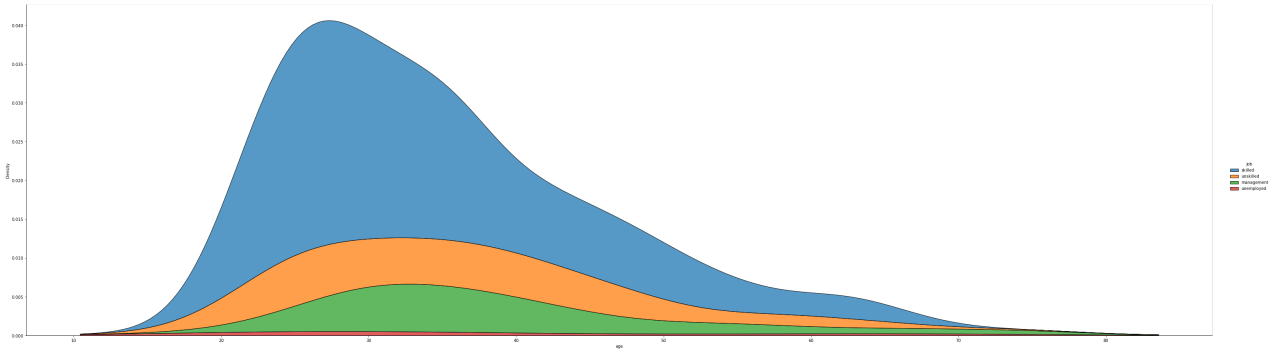


In [ ]:

Age in Relation to Job

In [168…

```
# Ploting a displot of Age vs Job
sns.displot(
    data=data, x="age", hue="job", multiple="stack", kind="kde", height=12, aspect=3.5,
)
```



In [169…

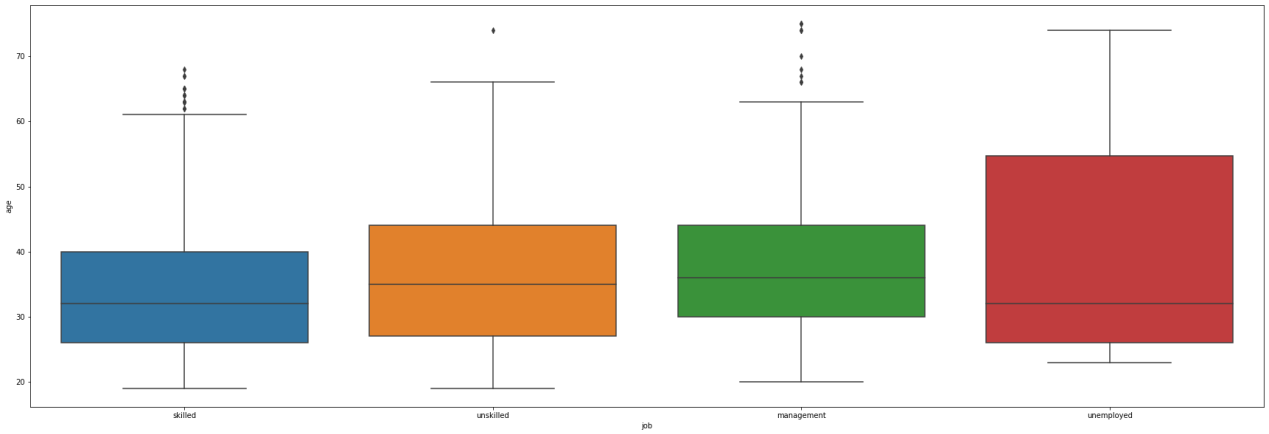
```
# Creating a sunburst chart for Age vs Job
import plotly.express as px

fig = px.sunburst(
    data,
    path=["job"],
    values="age",
    color="age",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

In [170…

```
# Boxplot of Age vs Job
plt.figure(figsize=(30, 10))
sns.boxplot(data=data, x="job", y="age")
```

Out[170…

```
<AxesSubplot:xlabel='job', ylabel='age'>
```



In [171…

```
# Creating summary statitics pivot table for Age vs Job
Age_job = data.pivot_table(
    index=["job"],
    values=["age"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(Age_job)
```

```
              age
              max  mean median   min    std    var
job
```

```
management 75.00 39.03  36.00 20.00 11.95 142.82
skilled    68.00 34.25  32.00 19.00 10.71 114.71
unemployed 74.00 40.09  32.00 23.00 17.78 315.99
unskilled  74.00 36.54  35.00 19.00 11.43 130.62
```

In [172…

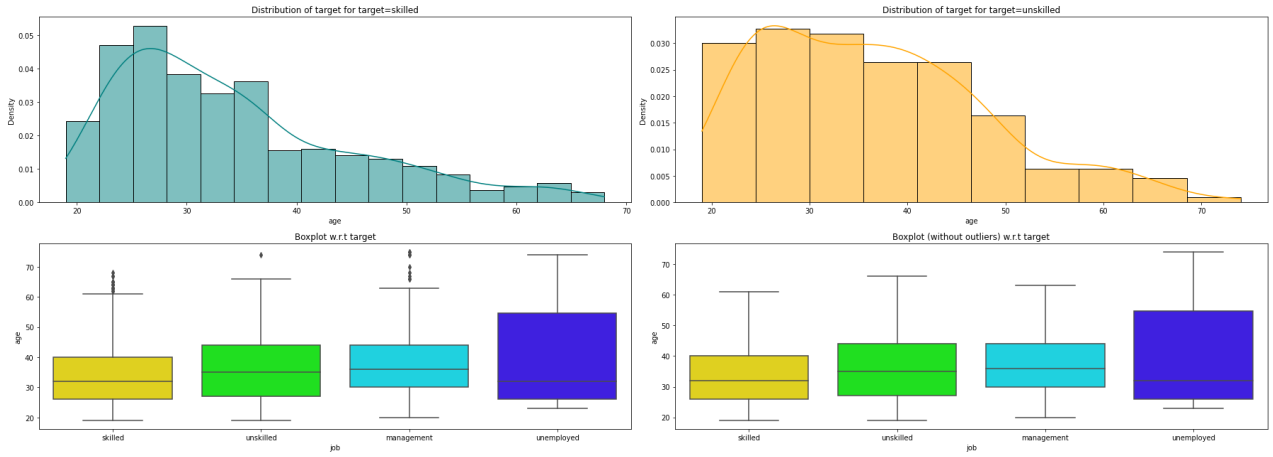
```
# comparing Age vs Job
data.groupby(["job"])["age"].describe()
```

Out[172…

job	count	mean	std	min	25%	50%	75%	max
management	148.00	39.03	11.95	20.00	30.00	36.00	44.00	75.00
skilled	630.00	34.25	10.71	19.00	26.00	32.00	40.00	68.00
unemployed	22.00	40.09	17.78	23.00	26.00	32.00	54.75	74.00
unskilled	200.00	36.54	11.43	19.00	27.00	35.00	44.00	74.00

In [173…

```
distribution_plot_wrt_target(
    data, "age", "job"
)  ## Complete the code to find distribution of Age vs Job
```



General Observation

- hh

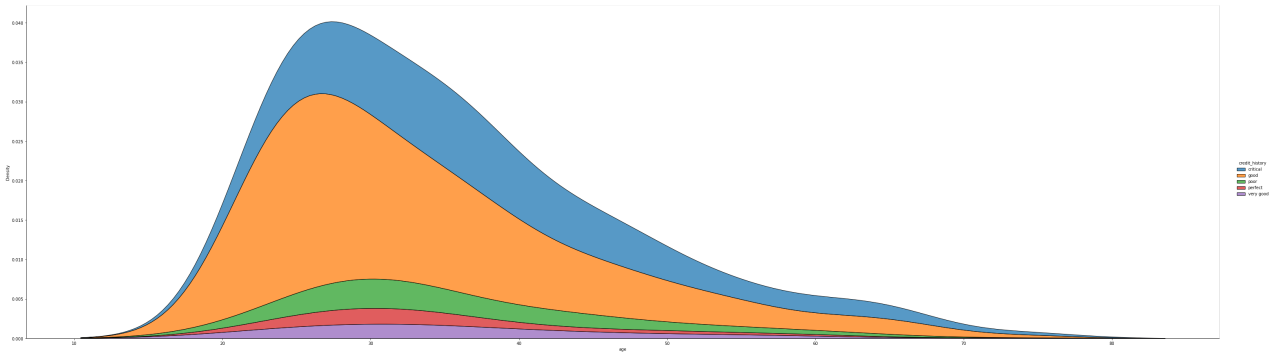
In [ ]:

Age in Relation to Credit History

In [174…

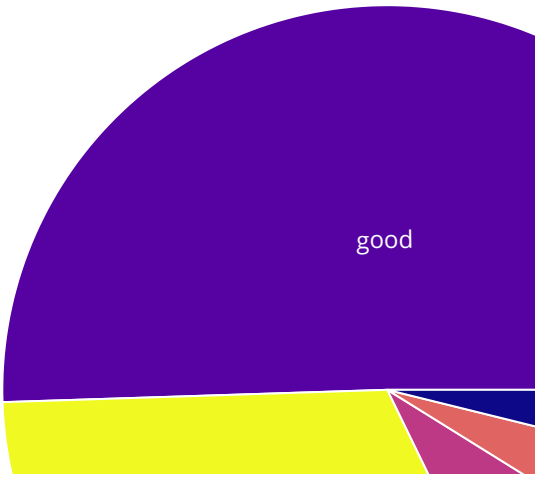
```
# Ploting a displot of Age vs Credit History
sns.displot(
    data=data,
    x="age",
    hue="credit_history",
```

```
    multiple="stack",
    kind="kde",
    height=12,
    aspect=3.5,
)
```



In [175…

```
# Creating a sunburst chart for Age vs Credit History
import plotly.express as px

fig = px.sunburst(
    data,
    path=["credit_history"],
    values="age",
    color="age",
    color_discrete_map={"(?)": "red", "Lunch": "gold", "Dinner": "green"},
)
fig.show()
```

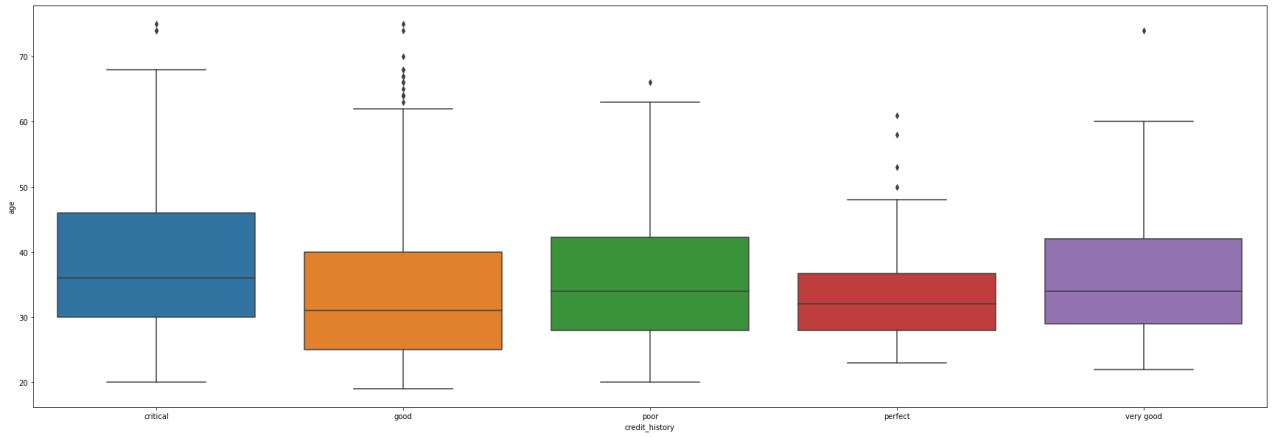


In [176…

```
# Boxplot of Age vs Credit History
plt.figure(figsize=(30, 10))
sns.boxplot(data=data, x="credit_history", y="age")
```

Out[176…

```
<AxesSubplot:xlabel='credit_history', ylabel='age'>
```



In [177…

```
# Creating summary statitics pivot table for Age vs Credit History
Age_credit = data.pivot_table(
    index=["credit_history"],
    values=["age"],
    aggfunc={"max", "median", "mean", "std", "var", "min"},
)
print(Age_credit)
```

```
                age
                max   mean median   min   std    var
credit_history
critical      75.00 38.44  36.00 20.00 11.60 134.47
good          75.00 33.88  31.00 19.00 11.22 125.99
perfect       61.00 34.30  32.00 23.00  9.41  88.47
poor          66.00 36.14  34.00 20.00 10.43 108.79
very good     74.00 36.27  34.00 22.00 11.54 133.28
```

In [178…

```
# comparing Age vs Credit History
data.groupby(["credit_history"])["age"].describe()
```

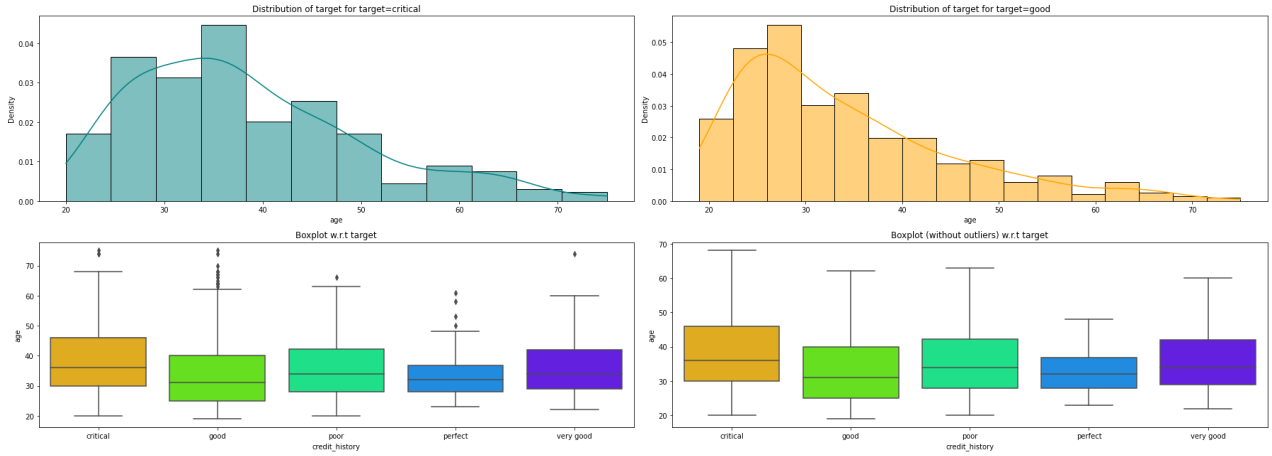
Out[178…

credit_history	count	mean	std	min	25%	50%	75%	max
critical	293.00	38.44	11.60	20.00	30.00	36.00	46.00	75.00
good	530.00	33.88	11.22	19.00	25.00	31.00	40.00	75.00
perfect	40.00	34.30	9.41	23.00	28.00	32.00	36.75	61.00
poor	88.00	36.14	10.43	20.00	28.00	34.00	42.25	66.00

	count	mean	std	min	25%	50%	75%	max
credit_history								
very good	49.00	36.27	11.54	22.00	29.00	34.00	42.00	74.00

In [179…

```
distribution_plot_wrt_target(
    data, "age", "credit_history"
)  ## Complete the code to find distribution of Age vs Credit History
```



General Observation

- hh

In [ ]:

General Observation

- hh

In [ ]:

General Observation

- hh

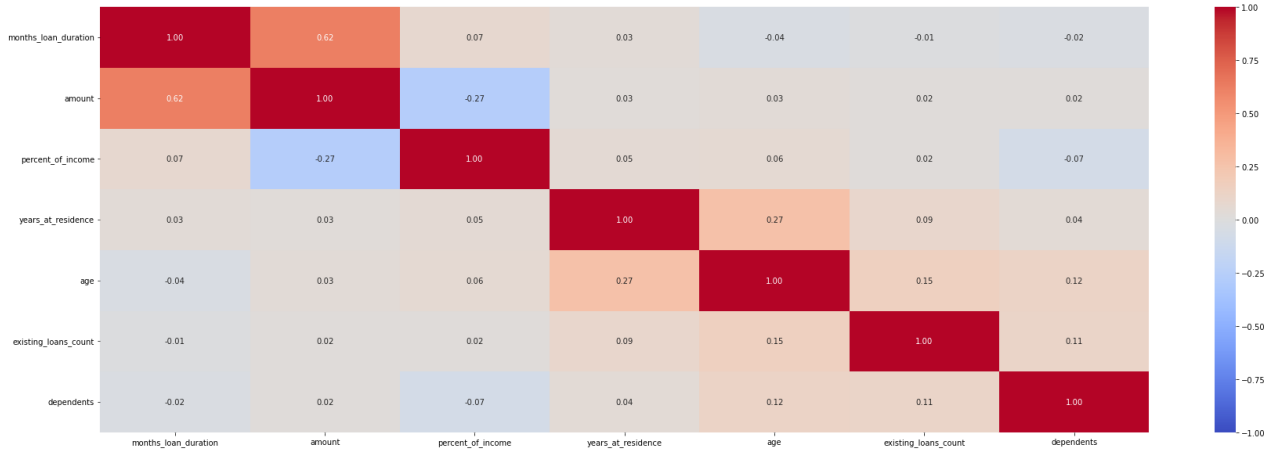
In [ ]:

In [ ]:

Correlation and Pairplot Analysis

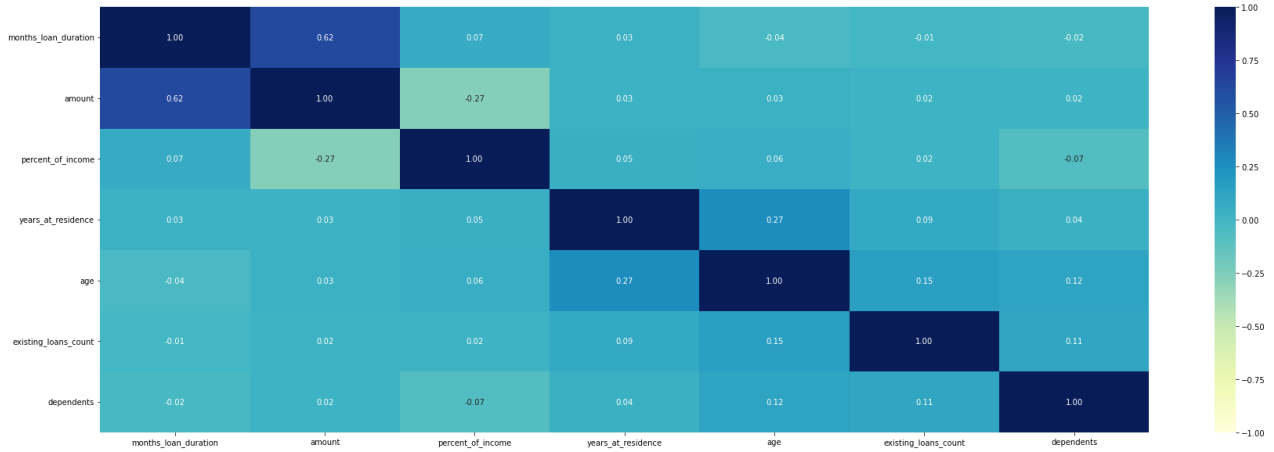
In [180…

```
# Displaying the correlation between numerical variables of the dataset
plt.figure(figsize=(30, 10))
sns.heatmap(data.corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="coolwarm")
plt.show()
```



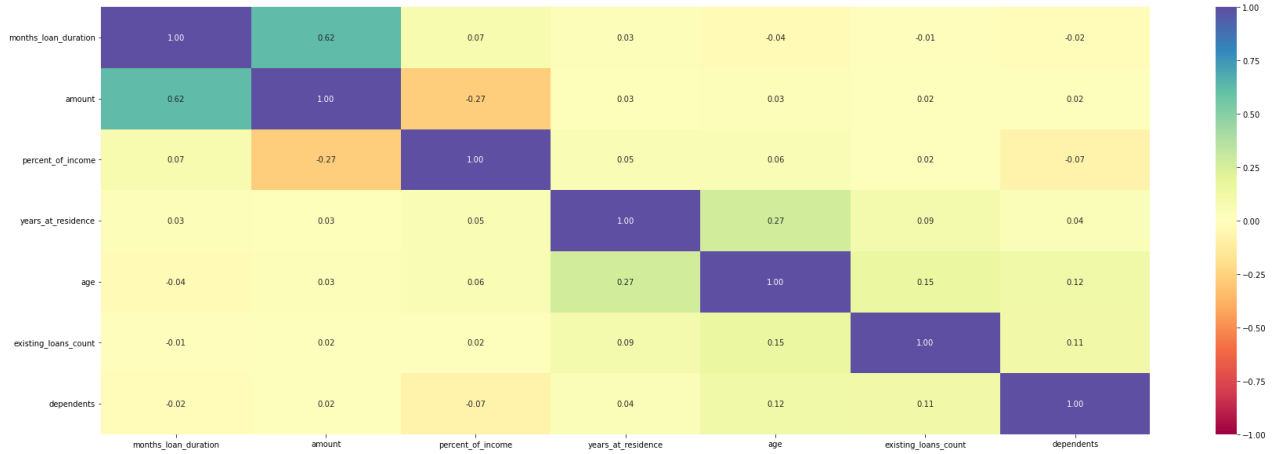
In [181…

```
# creates heatmap showing correlation of numeric columns in data
plt.figure(figsize=(30, 10))
sns.heatmap(data.corr(), vmin=-1, vmax=1, cmap="YlGnBu", annot=True, fmt=".2f")
```



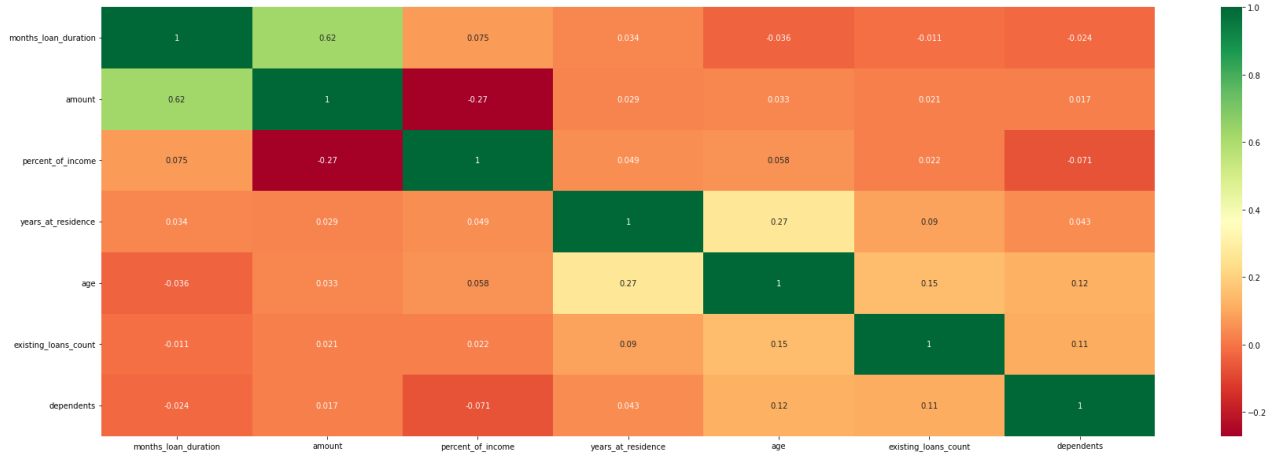
In [182…

```
cols_list = data.select_dtypes(include=np.number).columns.tolist()

plt.figure(figsize=(30, 10))
sns.heatmap(
    data[cols_list].corr(), annot=True, vmin=-1, vmax=1, fmt=".2f", cmap="Spectral"
)  ## Complete the code to find the correlation between the variables
plt.show()
```



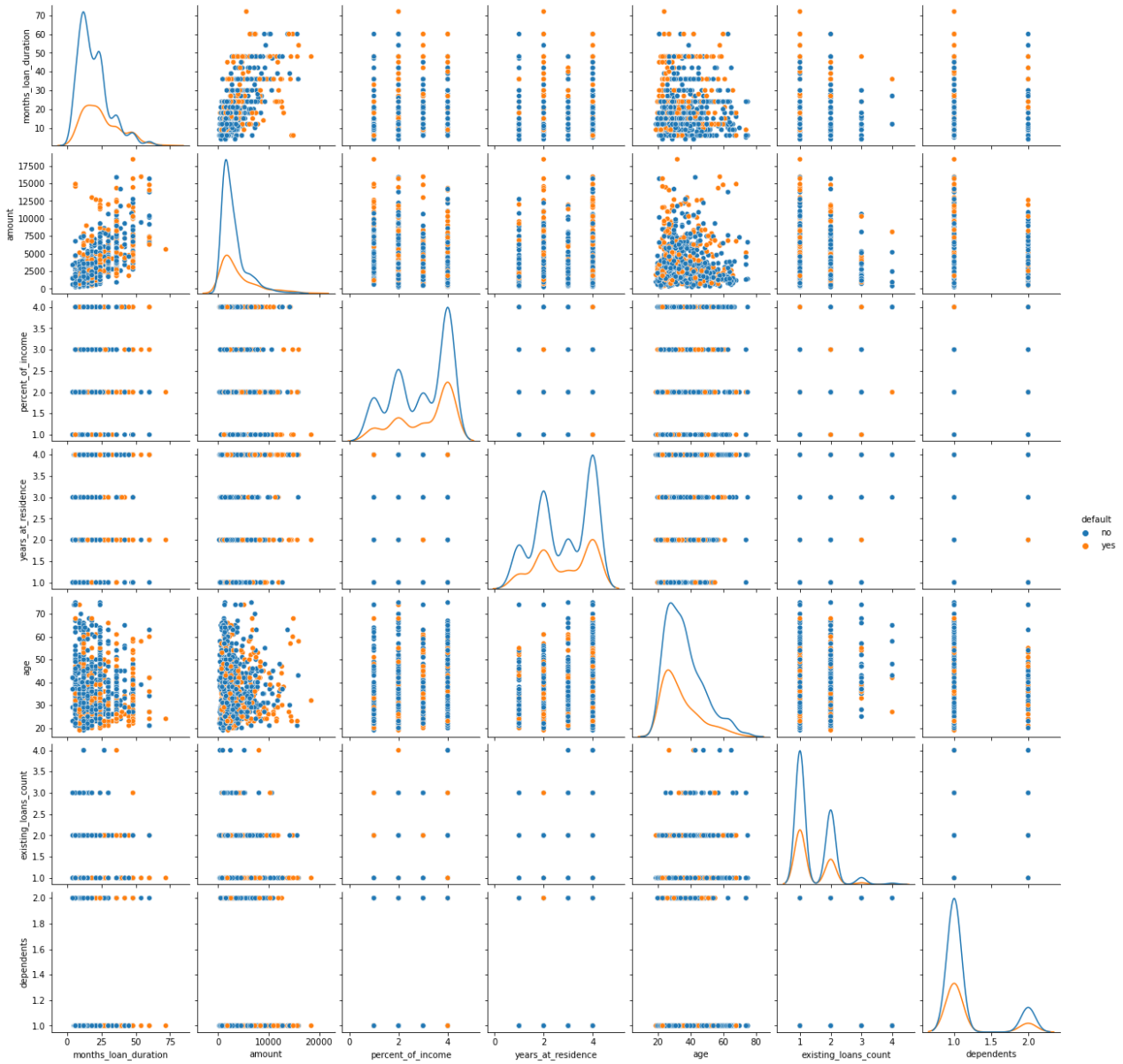
In [183…

```
# using heatmap
correlation = data.corr()  # creating a 2-D Matrix with correlation plots
plt.figure(figsize=(30, 10))
sns.heatmap(correlation, annot=True, cmap="RdYlGn")
```



In [184…

```
import seaborn as sns

sns.pairplot(data, hue="default", diag_kind="kde", diag_kws=dict(fill=False))
plt.show()
```



In [ ]:

Part VI: Data Preparation for Model Building

Notes

- ***Feature engineering includes all the steps required to prepare the data i.e outlier treatment, data cleaning, and scaling. Hyperparameter tuning is not a part of feature engineering.***

Let's check the count of each unique category in each of the categorical variables.

In [185…

```
# Extracting the missing values in the dataset
```

```
data.isnull().sum()
```

Out[185…

```
checking_balance        0
months_loan_duration    0
credit_history          0
purpose                 0
amount                  0
savings_balance         0
employment_duration     0
percent_of_income       0
years_at_residence      0
age                     0
other_credit            0
housing                 0
existing_loans_count    0
job                     0
dependents              0
phone                   0
default                 0
dtype: int64
```

In [186…

```
# Checking the total number of missing values in the dataset
data.isnull().sum().sum()
```

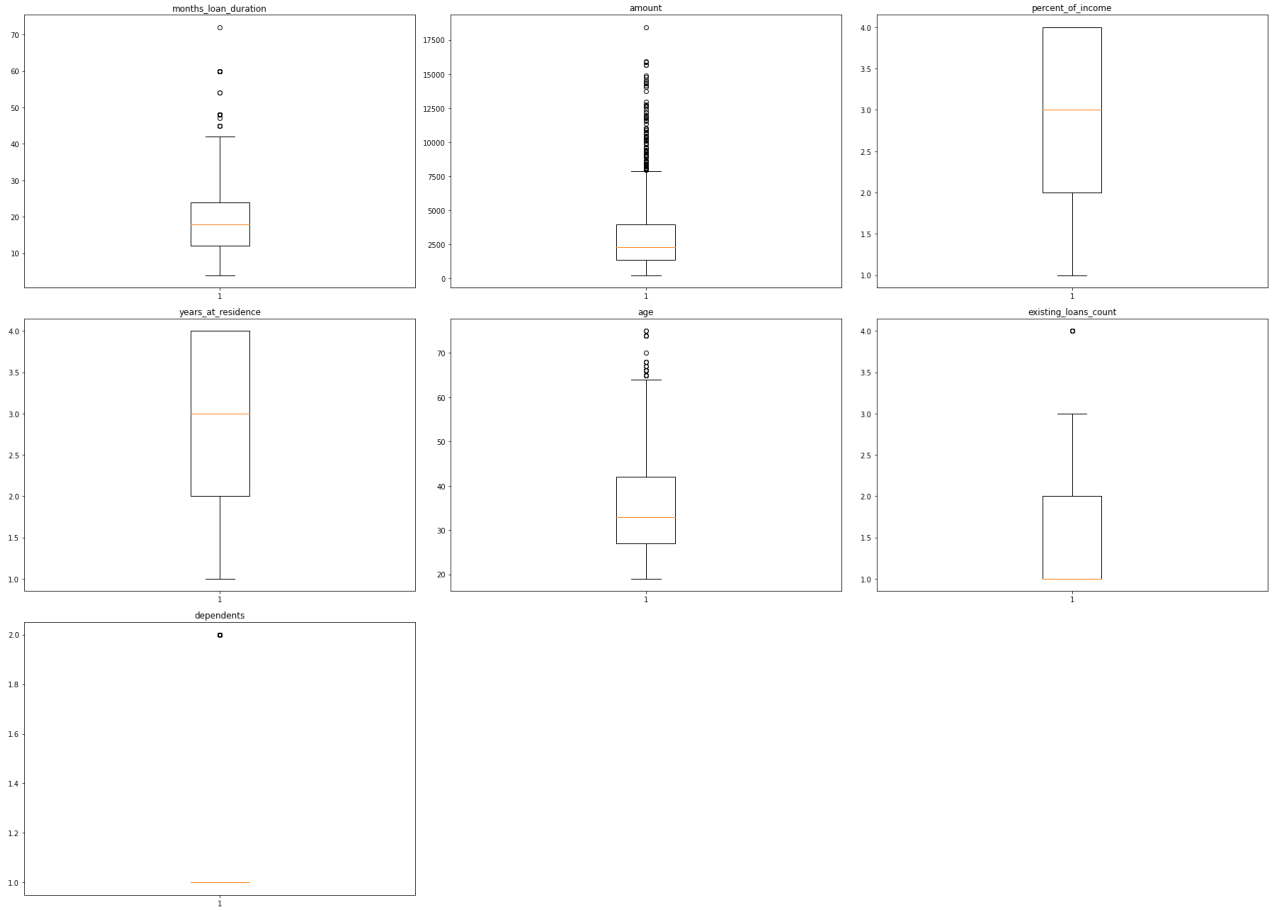
Out[186…

```
0
```

Outlier Detection and Treatment

In [187…

```
# Outlier Check
# outlier detection using boxplot
import numpy as np

numeric_columns = data.select_dtypes(include=np.number).columns.tolist()
plt.figure(figsize=(25, 18))
for i, parameter in enumerate(numeric_columns):
    plt.subplot(3, 3, i + 1)
    plt.boxplot(data[parameter], whis=1.5)
    plt.tight_layout()
    plt.title(parameter)

plt.show()
```



- There are quite a few outliers in the data.
- However, we will not treat them as they are proper values.

Observations

- jj

Data Preparation for modeling

- We want to predict which visa will be certified.
- Before we proceed to build a model, we'll have to encode categorical features.
- We'll split the data into train and test to be able to evaluate the model that we build on the train data.

In [188…

```
data.columns
```

Out[188…

```
Index(['checking_balance', 'months_loan_duration', 'credit_history', 'purpose',
       'amount', 'savings_balance', 'employment_duration', 'percent_of_income',
       'years_at_residence', 'age', 'other_credit', 'housing',
       'existing_loans_count', 'job', 'dependents', 'phone', 'default'],
      dtype='object')
```

In [189…

```
data.head()
```

Out[189...

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance
0	< 0 DM	6	critical	furniture/appliances	1169	unknown
1	1 - 200 DM	48	good	furniture/appliances	5951	< 100 DM
2	unknown	12	critical	education	2096	< 100 DM
3	< 0 DM	42	good	furniture/appliances	7882	< 100 DM
4	< 0 DM	24	poor	car	4870	< 100 DM

In [190...

```
data["default"].value_counts()
```

Out[190...

```
no     700
yes    300
Name: default, dtype: int64
```

In [191...

```
data["default"].value_counts(normalize=True)
```

Out[191...

```
no    0.70
yes   0.30
Name: default, dtype: float64
```

Lets convert the columns with an 'object' datatype into categorical variables

In [192...

```
creditData = data.copy()
```

In [193...

```
replaceStruct = {
                "checking_balance":     {"< 0 DM": 1, "1 - 200 DM": 2 ,"> 200 DM": 3 ,"
                "credit_history": {"critical": 1, "poor":2 , "good": 3, "very good": 4,
                 "savings_balance": {"< 100 DM": 1, "100 - 500 DM":2 , "500 - 1000 DM":
                 "employment_duration":     {"unemployed": 1, "< 1 year": 2 ,"1 - 4 yea
                "phone":     {"no": 1, "yes": 2 },
                #"job":     {"unemployed": 1, "unskilled": 2, "skilled": 3, "management
                "default":     {"no": 0, "yes": 1 }
                    }
oneHotCols=["purpose","housing","other_credit","job"]
```

In [194...

```
creditData=creditData.replace(replaceStruct)
creditData=pd.get_dummies(creditData, columns=oneHotCols)
creditData.head(10)
```

Out[194...

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_durati
0	1	6	1	1169	-1	
1	2	48	3	5951	1	

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_durati
2	-1	12	1	2096	1	
3	1	42	3	7882	1	
4	1	24	2	4870	1	
5	-1	36	3	9055	-1	
6	-1	24	3	2835	3	
7	2	36	3	6948	1	
8	-1	12	3	3059	4	
9	2	30	1	5234	1	

In [195…

```
creditData.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 29 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   checking_balance               1000 non-null   int64
 1   months_loan_duration           1000 non-null   int64
 2   credit_history                 1000 non-null   int64
 3   amount                         1000 non-null   int64
 4   savings_balance                1000 non-null   int64
 5   employment_duration            1000 non-null   int64
 6   percent_of_income              1000 non-null   int64
 7   years_at_residence             1000 non-null   int64
 8   age                            1000 non-null   int64
 9   existing_loans_count           1000 non-null   int64
 10  dependents                     1000 non-null   int64
 11  phone                          1000 non-null   int64
 12  default                        1000 non-null   int64
 13  purpose_business               1000 non-null   uint8
 14  purpose_car                    1000 non-null   uint8
 15  purpose_car0                   1000 non-null   uint8
 16  purpose_education              1000 non-null   uint8
 17  purpose_furniture/appliances   1000 non-null   uint8
 18  purpose_renovations            1000 non-null   uint8
 19  housing_other                  1000 non-null   uint8
 20  housing_own                    1000 non-null   uint8
 21  housing_rent                   1000 non-null   uint8
 22  other_credit_bank              1000 non-null   uint8
 23  other_credit_none              1000 non-null   uint8
 24  other_credit_store             1000 non-null   uint8
 25  job_management                 1000 non-null   uint8
 26  job_skilled                    1000 non-null   uint8
 27  job_unemployed                 1000 non-null   uint8
 28  job_unskilled                  1000 non-null   uint8
dtypes: int64(13), uint8(16)
memory usage: 117.3 KB
```

In [196…
```
df = creditData.copy()
```

Split the data into train and test sets

- When data (classification) exhibit a significant imbalance in the distribution of the target classes, it is good to use stratified sampling to ensure that relative class frequencies are approximately preserved in train and test sets.
- This is done by setting the `stratify` parameter to target variable in the train_test_split function.

In [197…
```
#data["case_status"] = data["case_status"].apply(lambda x: 1 if x == "Certified" else 0

X = df.drop('default', axis = 1) ## Complete the code to drop case status from the data
Y = df["default"]



X = pd.get_dummies(X,  columns=X.select_dtypes(include=["object", "category"]).columns.
    drop_first=True,
)
X.head()  ## Complete the code to create dummies
```

Out[197…

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_durati
0	1	6	1	1169	-1	
1	2	48	3	5951	1	
2	-1	12	1	2096	1	
3	1	42	3	7882	1	
4	1	24	2	4870	1	

In [198…
```
X
```

Out[198…

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_dur
0	1	6	1	1169	-1	
1	2	48	3	5951	1	
2	-1	12	1	2096	1	
3	1	42	3	7882	1	
4	1	24	2	4870	1	
…	…	…	…	…	…	

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_dur
995	-1	12	3	1736	1	
996	1	30	3	3857	1	
997	-1	12	3	804	1	
998	1	45	3	1845	1	
999	2	45	1	4576	2	

1000 rows × 28 columns

In [199…

```
# Splitting data into training and test set:
## Complete the code to split the data into train and test in the ratio 70:30

X_train, X_test, Y_train, Y_test = train_test_split(
    X, Y, test_size=0.3, random_state=1, stratify=Y
)
print(X_train.shape, X_test.shape)
```

```
(700, 28) (300, 28)
```

In [200…

```
Y.value_counts()
```

Out[200…

```
0    700
1    300
Name: default, dtype: int64
```

In [201…

```
Y_test.value_counts(1)
```

Out[201…

```
0   0.70
1   0.30
Name: default, dtype: float64
```

In [202…

```
print("*" * 60)
print("Shape of Training set : ", X_train.shape)

print("*" * 60)
print("Shape of test set : ", X_test.shape)

print("*" * 60)
print("Percentage of classes in training set:")
print("*" * 60)
print(Y_train.value_counts(normalize=True))
print("*" * 60)
print("Percentage of classes in test set:")

print(Y_test.value_counts(normalize=True))
print("*" * 60)
```

```
************************************************************
```

```
Shape of Training set :  (700, 28)
**********************************************************
Shape of test set :  (300, 28)
**********************************************************
Percentage of classes in training set:
**********************************************************
0   0.70
1   0.30
Name: default, dtype: float64
**********************************************************
Percentage of classes in test set:
0   0.70
1   0.30
Name: default, dtype: float64
**********************************************************
```

In [203…

```
print(X_train.shape, X_test.shape)
```

```
(700, 28) (300, 28)
```

Model evaluation criterion

First, let's create functions to calculate different metrics and confusion matrix so that we don't have to use the same code repeatedly for each model.

- The model_performance_classification_sklearn function will be used to check the model performance of models.
- The confusion_matrix_sklearn function will be used to plot the confusion matrix.

In [204…

```
# defining a function to compute different metrics to check performance of a classifica


def model_performance_classification_sklearn(model, predictors, target):
    """
    Function to compute different metrics to check classification model performance

    model: classifier
    predictors: independent variables
    target: dependent variable
    """

    # predicting using the independent variables
    pred = model.predict(predictors)

    acc = accuracy_score(target, pred)  # to compute Accuracy
    recall = recall_score(target, pred)  # to compute Recall
    precision = precision_score(target, pred)  # to compute Precision
    f1 = f1_score(target, pred)  # to compute F1-score

    # creating a dataframe of metrics
    df_perf = pd.DataFrame(
        {"Accuracy": acc, "Recall": recall, "Precision": precision, "F1": f1,},
```

```
        index=[0],
    )

    return df_perf
```

In [205…

```
def confusion_matrix_sklearn(model, predictors, target):
    """
    To plot the confusion_matrix with percentages

    model: classifier
    predictors: independent variables
    target: dependent variable
    """
    Y_pred = model.predict(predictors)
    cm = confusion_matrix(target, Y_pred)
    labels = np.asarray(
        [
            ["{0:0.0f}".format(item) + "\n{0:.2%}".format(item / cm.flatten().sum())]
            for item in cm.flatten()
        ]
    ).reshape(2, 2)

    plt.figure(figsize=(15, 6))
    sns.heatmap(cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [206…

```
## Function to create confusion matrix
def make_confusion_matrix(model, y_actual, labels=[1, 0]):
    """
    model : classifier to predict values of X
    y_actual : ground truth

    """
    y_predict = model.predict(X_test)
    cm = metrics.confusion_matrix(y_actual, y_predict, labels=[0, 1])
    df_cm = pd.DataFrame(
        cm,
        index=[i for i in ["Actual - No", "Actual - Yes"]],
        columns=[i for i in ["Predicted - No", "Predicted - Yes"]],
    )
    group_counts = ["{0:0.0f}".format(value) for value in cm.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cm.flatten() / np.sum(cm)
    labels = [f"{v1}\n{v2}" for v1, v2 in zip(group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)
    plt.figure(figsize=(15, 7))
    sns.heatmap(df_cm, annot=labels, fmt="")
    plt.ylabel("True label")
    plt.xlabel("Predicted label")
```

In [207…

```
##  Function to calculate different metric scores of the model - Accuracy, Recall and P
```

```python
def get_metrics_score(model, flag=True):
    """
    model : classifier to predict values of X

    """
    # defining an empty list to store train and test results
    score_list = []

    # Predicting on train and tests
    pred_train = model.predict(X_train)
    pred_test = model.predict(X_test)

    # Accuracy of the model
    train_acc = model.score(X_train, Y_train)
    test_acc = model.score(X_test, Y_test)

    # Recall of the model
    train_recall = metrics.recall_score(Y_train, pred_train)
    test_recall = metrics.recall_score(Y_test, pred_test)

    # Precision of the model
    train_precision = metrics.precision_score(Y_train, pred_train)
    test_precision = metrics.precision_score(Y_test, pred_test)

    score_list.extend(
        (
            train_acc,
            test_acc,
            train_recall,
            test_recall,
            train_precision,
            test_precision,
        )
    )

    # If the flag is set to True then only the following print statements will be dispa
    if flag == True:
        print("Accuracy on training set : ", model.score(X_train, Y_train))
        print("Accuracy on test set : ", model.score(X_test, Y_test))
        print("Recall on training set : ", metrics.recall_score(Y_train, pred_train))
        print("Recall on test set : ", metrics.recall_score(Y_test, pred_test))
        print(
            "Precision on training set : ", metrics.precision_score(Y_train, pred_train
        )
        print("Precision on test set : ", metrics.precision_score(Y_test, pred_test))

    return score_list  # returning the list with train and test scores
```

Part VII:Building Models for Decision Trees, Bagging Boosting

Building the model

- We are going to build 2 ensemble models here - Bagging Classifier and Random Forest Classifier.
- First, let's build these models with default parameters and then use hyperparameter tuning to optimize the model performance.
- We will calculate all three metrics - Accuracy, Precision and Recall but the metric of interest here is recall.
- `Recall` - It gives the ratio of True positives to Actual positives, so high Recall implies low false negatives, i.e. low chances of predicting a defaulter as non defaulter

Decision Tree Model

- ***The decision tree model would be considered overfit if the metric of interest is 'accuracy' since the difference between the training data accuracy and testing data accuracy is very large which indicates the fact that the model is not able to generalize to new data points and is overfitting the training dataset.***

- ***When classification problems exhibit a significant imbalance in the distribution of the target classes, it is good to use stratified sampling to ensure that relative class frequencies are approximately preserved in train and test sets. This is done using the stratify parameter in the train_test_split function.***

In [208…

```
model = DecisionTreeClassifier(
    criterion="gini", random_state=1
)  ## Complete the code to define decision tree classifier with random state = 1
model.fit(
    X_train, Y_train
)  ## Complete the code to fit decision tree classifier on the train data
```

Out[208…

```
DecisionTreeClassifier(random_state=1)
```

In [209…

```
X.shape
```

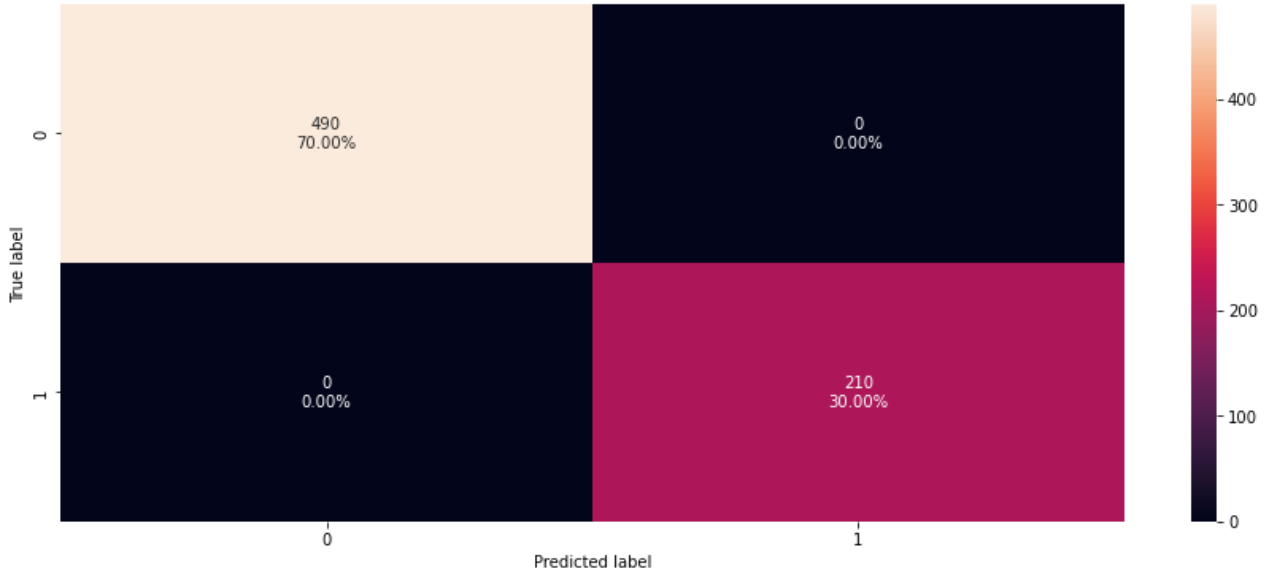
Out[209…

```
(1000, 28)
```

Checking model performance on training set

In [210…

```
# confusion_matrix_sklearn(model, X_train, Y_train) ## Complete the code to create conf
confusion_matrix_sklearn(model, X_train, Y_train)
```



In [211…

```
decision_tree_perf_train = model_performance_classification_sklearn(
    model, X_train, Y_train
)  ## Complete the code to check performance on train data
decision_tree_perf_train
```

Out[211…

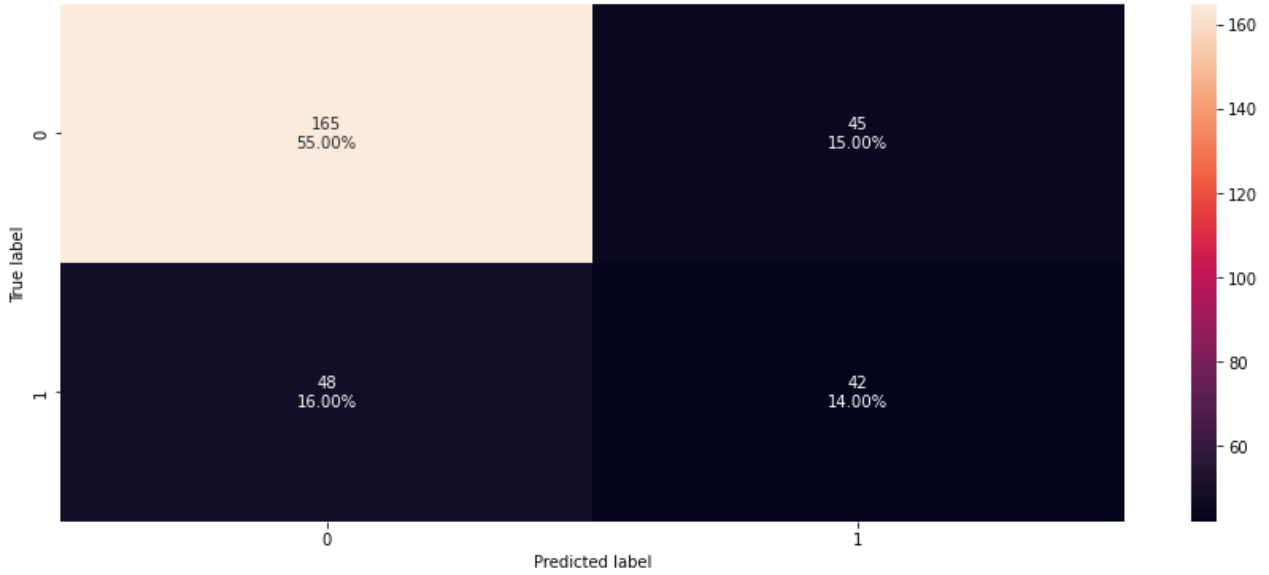
	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Observation

The decision tree model is highly overfitting the train dataset.

In [212…

```
confusion_matrix_sklearn(
    model, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [213…
```
decision_tree_perf_test = model_performance_classification_sklearn(
    model, X_test, Y_test
)  ## Complete the code to check performance for test data
decision_tree_perf_test
```

Out[213…

	Accuracy	Recall	Precision	F1
0	0.69	0.47	0.48	0.47

- The decision tree is overfitting the training data.

Method2: Weighted Decision Tree Model

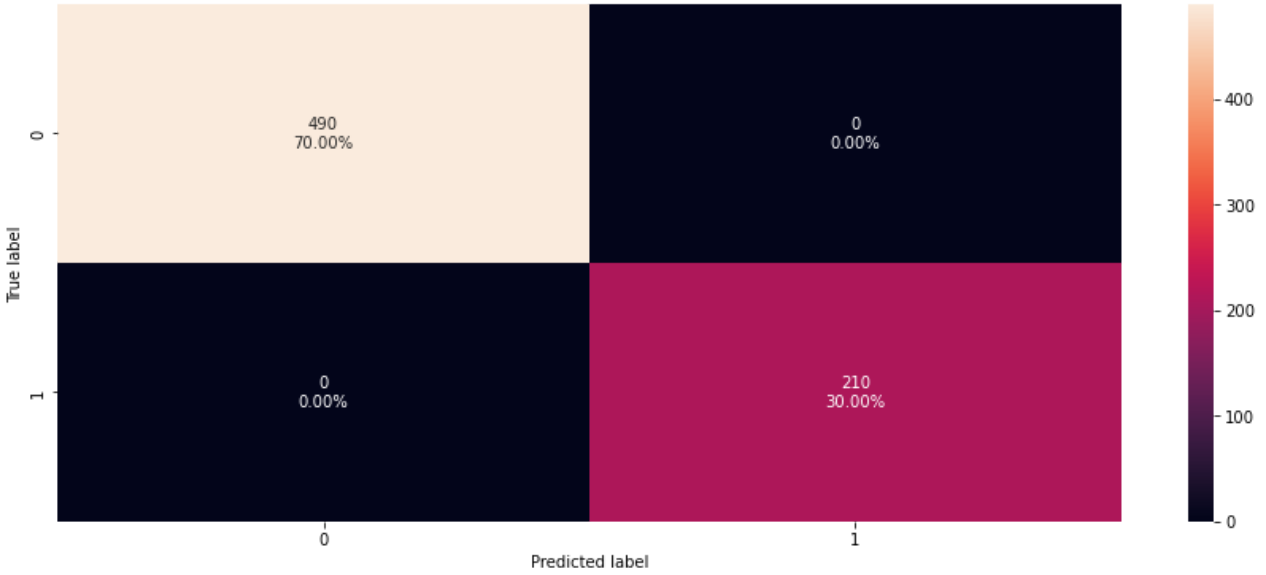
- We will build our model using the DecisionTreeClassifier function. Using default 'gini' criteria to split.
- If the frequency of class A is 10% and the frequency of class B is 90%, then class B will become the dominant class and the decision tree will become biased toward the dominant classes.
- In this case, we can pass a dictionary {0:0.17,1:0.83} to the model to specify the weight of each class and the decision tree will give more weightage to class 1.
- class_weight is a hyperparameter for the decision tree classifier.

In [214…
```
dtree = DecisionTreeClassifier(
    criterion="gini", class_weight={0: 0.17, 1: 0.83}, random_state=1
)
```

In [215…
```
dtree.fit(X_train, Y_train)
```

Out[215…
```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [216…
```
confusion_matrix_sklearn(dtree, X_train, Y_train)
```



In [217…

```
# Training Performance Mearsures
dtree_model_train_perf = model_performance_classification_sklearn(
    dtree, X_train, Y_train
)
print("Training performance \n")
dtree_model_train_perf
```

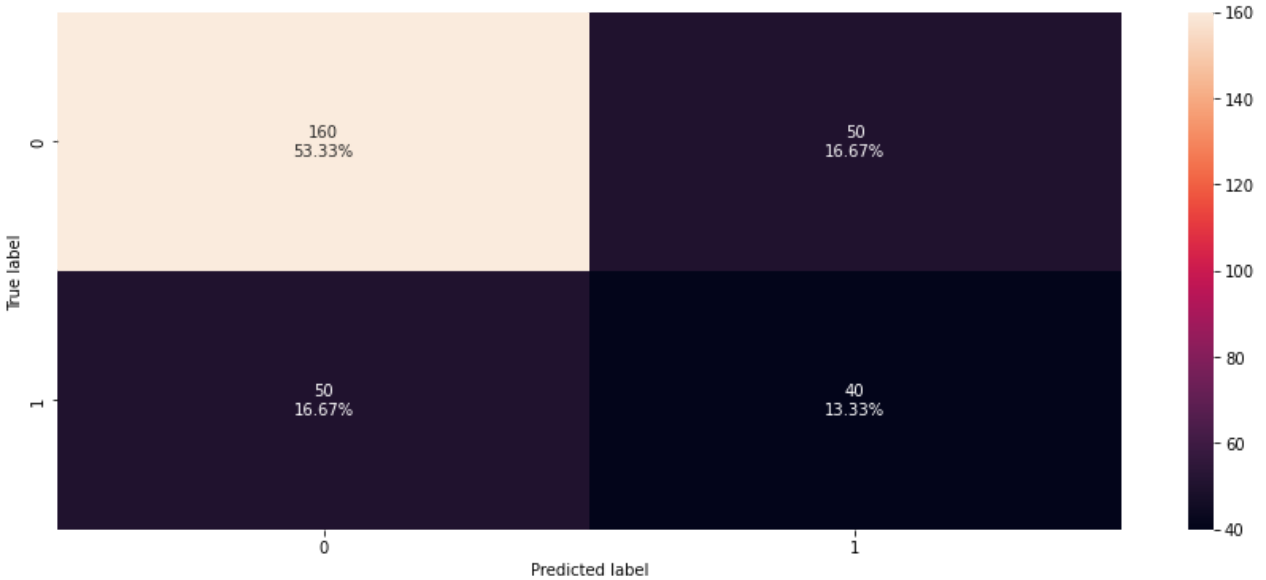
```
Training performance
```

Out[217…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

In [218…

```
confusion_matrix_sklearn(dtree, X_test, Y_test)
```



In [219…

```
# Test Performance Measures
```

```
dtree_model_test_perf = model_performance_classification_sklearn(dtree, X_test, Y_test)
print("Testing performance \n")
dtree_model_test_perf
```

```
Testing performance
```

Out[219…

	Accuracy	Recall	Precision	F1
0	0.67	0.44	0.44	0.44

- Decision tree is working well on the training data but is not able to generalize well on the test data concerning the recall.

Hyperparameter Tuning - Decision Tree

In [220…

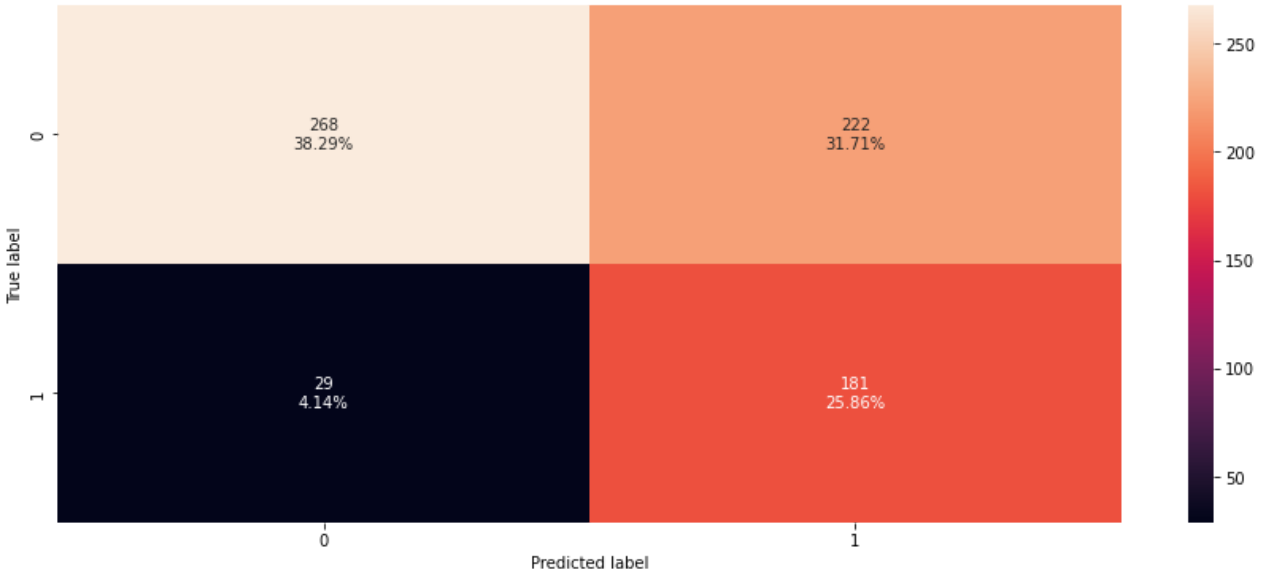
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(class_weight="balanced", random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(10, 30, 5),
    "min_samples_leaf": [3, 5, 7],
    "max_leaf_nodes": [2, 3, 5],
    "min_impurity_decrease": [0.0001, 0.001],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    dtree_estimator, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with n_jobs = -1

grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on the train data

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, Y_train)
```

Out[220…

```
DecisionTreeClassifier(class_weight='balanced', max_depth=10, max_leaf_nodes=
5,
                       min_impurity_decrease=0.0001, min_samples_leaf=3,
                       random_state=1)
```

In [221…

```
confusion_matrix_sklearn(
    dtree_estimator, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [222…

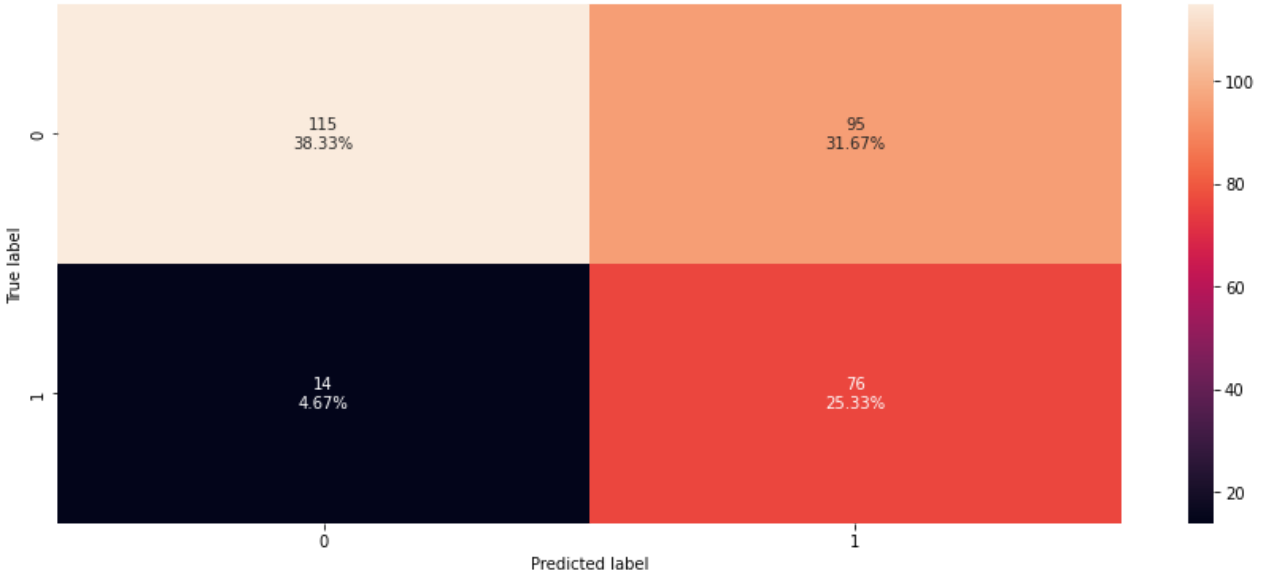
```
# Tunned Training Performance Measures
dtree_estimator_model_train_perf = model_performance_classification_sklearn(
    dtree_estimator, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
dtree_estimator_model_train_perf
```

Out[222…

	Accuracy	Recall	Precision	F1
0	0.64	0.86	0.45	0.59

In [223…

```
confusion_matrix_sklearn(
    dtree_estimator, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [224…

```
# Tunned Test Performance Measures
dtree_estimator_model_test_perf = model_performance_classification_sklearn(
    dtree_estimator, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
dtree_estimator_model_test_perf
```

Out[224…

	Accuracy	Recall	Precision	F1
0	0.64	0.84	0.44	0.58

Hyperparameter Tuning Weighted Decision Tree

In [225…

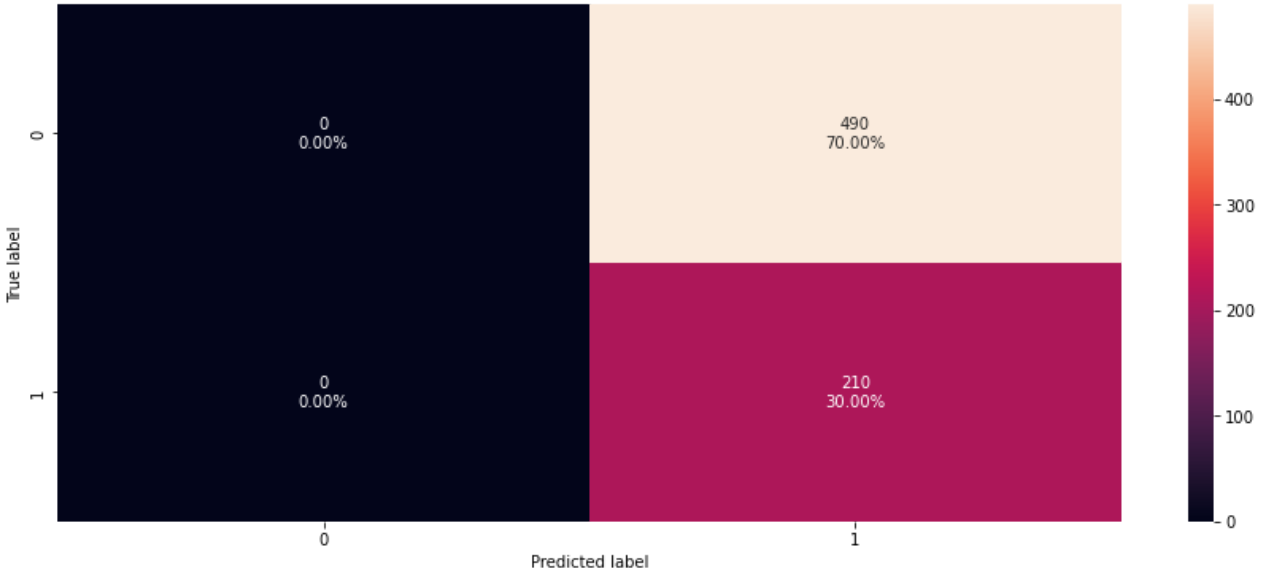
```
# Choose the type of classifier.
dtree_estimator = DecisionTreeClassifier(
    class_weight={0: 0.17, 1: 0.83}, random_state=1
)

# Grid of parameters to choose from
parameters = {
    "max_depth": np.arange(2, 30),
    "min_samples_leaf": [1, 2, 5, 7, 10],
    "max_leaf_nodes": [2, 3, 5, 10, 15],
    "min_impurity_decrease": [0.0001, 0.001, 0.01, 0.1],
}

# Type of scoring used to compare parameter combinations
scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(dtree_estimator, parameters, scoring=scorer)
grid_obj = grid_obj.fit(X_train, Y_train)

# Set the clf to the best combination of parameters
dtree_estimator = grid_obj.best_estimator_

# Fit the best algorithm to the data.
dtree_estimator.fit(X_train, Y_train)
```

Out[225…

```
DecisionTreeClassifier(class_weight={0: 0.17, 1: 0.83}, max_depth=2,
                       max_leaf_nodes=2, min_impurity_decrease=0.1,
                       random_state=1)
```

In [226…

```
confusion_matrix_sklearn(dtree_estimator, X_train, Y_train)
```



In [227…

```
# Training Performance Measures
dtree_estimator_model_train_perf = model_performance_classification_sklearn(
    dtree_estimator, X_train, Y_train
)

print("Training performance \n")
dtree_estimator_model_train_perf
```

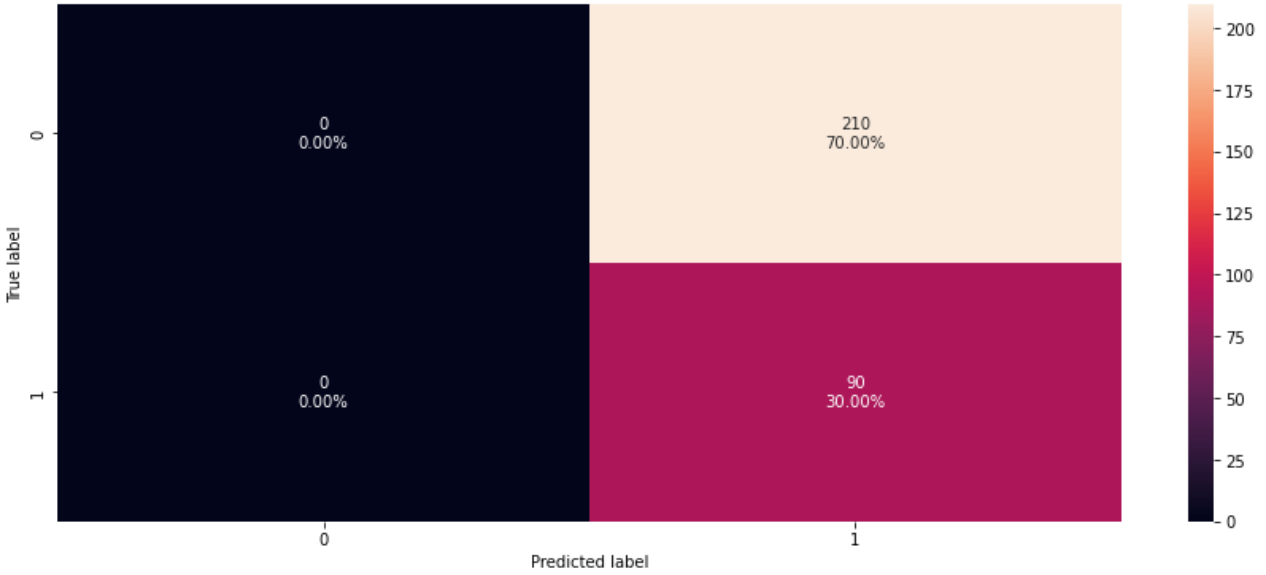
```
Training performance
```

Out[227…

	Accuracy	Recall	Precision	F1
0	0.30	1.00	0.30	0.46

In [228…

```
confusion_matrix_sklearn(dtree_estimator, X_test, Y_test)
```



In [229…
```
# Testing Performance Measures
dtree_estimator_model_test_perf = model_performance_classification_sklearn(
    dtree_estimator, X_test, Y_test
)
print("Testing performance \n")
dtree_estimator_model_test_perf
```

```
Testing performance
```

Out[229…

	Accuracy	Recall	Precision	F1
0	0.30	1.00	0.30	0.46

- The decision tree model has a high recall but, the precision is quite lower.
- The performance of the model after hyperparameter tuning can be generalized.

Important Features for Predicting Loan Default

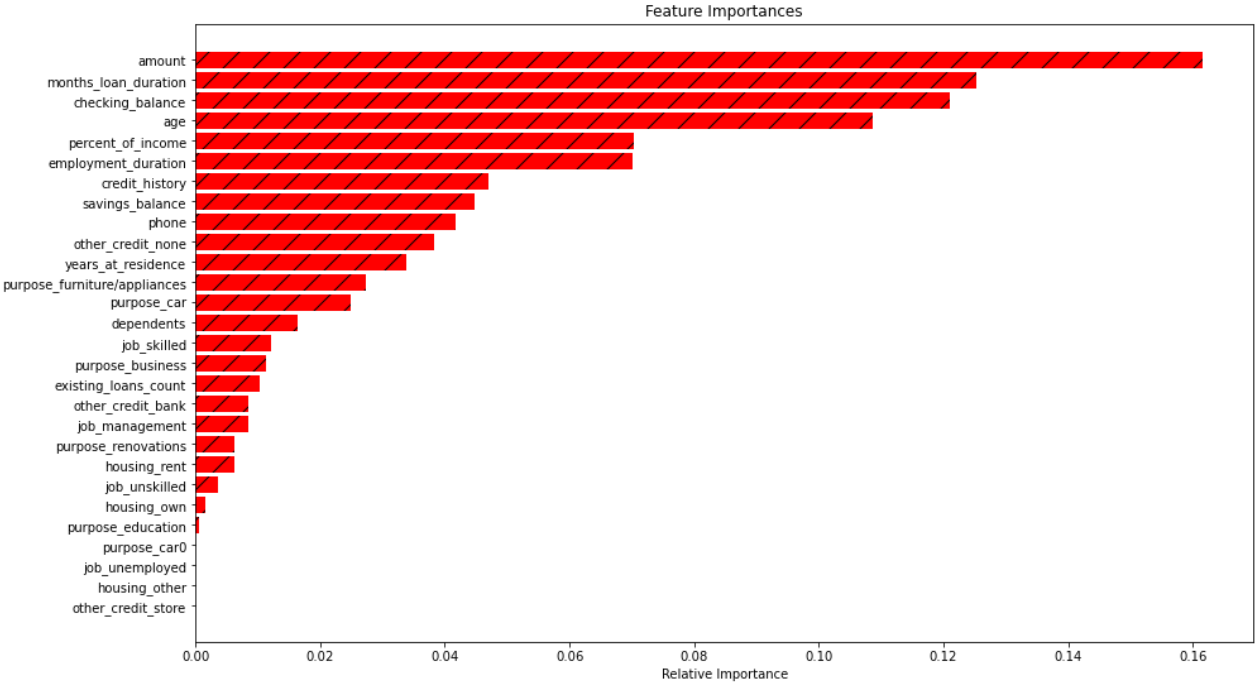
In [230…
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        model.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                                Imp
amount                         0.16
months_loan_duration           0.13
checking_balance               0.12
age                            0.11
percent_of_income              0.07
employment_duration            0.07
credit_history                 0.05
savings_balance                0.04
phone                          0.04
other_credit_none              0.04
years_at_residence             0.03
purpose_furniture/appliances   0.03
purpose_car                    0.02
dependents                     0.02
job_skilled                    0.01
purpose_business               0.01
existing_loans_count           0.01
other_credit_bank              0.01
job_management                 0.01
purpose_renovations            0.01
housing_rent                   0.01
job_unskilled                  0.00
housing_own                    0.00
purpose_education              0.00
other_credit_store             0.00
housing_other                  0.00
```

```
job_unemployed              0.00
purpose_car0                0.00
```

In [231…

```
# Extracting the Important Features for Prediction Diabetes Using Gini Criteria Decisio
importances = model.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="red", align="center", hatch="/"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



In [232…

```
# Text report showing the rules of a decision tree -
feature_names = list(X_train.columns)
print(tree.export_text(model, feature_names=feature_names, show_weights=True))
```

```
|--- checking_balance <= 0.00
|   |--- other_credit_none <= 0.50
|   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |--- age <= 43.50
|   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |--- years_at_residence <= 3.00
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |--- years_at_residence >  3.00
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- percent_of_income >  1.50
|   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |--- months_loan_duration <= 39.00
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
```

```
|   |   |   |   |   |   |   |--- months_loan_duration >  39.00
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |   |   |--- savings_balance <= 3.50
|   |   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 8.00] class: 1
|   |   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |   |--- other_credit_bank <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- other_credit_bank >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |--- savings_balance >  3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- age >  43.50
|   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |   |--- amount <= 1005.00
|   |   |   |   |   |--- employment_duration <= 4.00
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- employment_duration >  4.00
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- amount >  1005.00
|   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |--- other_credit_none >  0.50
|   |   |   |--- age <= 29.50
|   |   |   |   |--- months_loan_duration <= 34.50
|   |   |   |   |   |--- age <= 22.50
|   |   |   |   |   |   |--- percent_of_income <= 2.50
|   |   |   |   |   |   |   |--- age <= 19.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- age >  19.50
|   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |--- percent_of_income >  2.50
|   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |--- age >  22.50
|   |   |   |   |   |   |--- amount <= 2226.50
|   |   |   |   |   |   |   |--- savings_balance <= 2.50
|   |   |   |   |   |   |   |   |--- amount <= 1922.00
|   |   |   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [17.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  1922.00
|   |   |   |   |   |   |   |   |   |--- percent_of_income <= 3.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- savings_balance >  2.50
|   |   |   |   |   |   |   |   |--- employment_duration <= 4.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- employment_duration >  4.00
```

```
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- amount >  2226.50
|   |   |   |   |   |   |   |--- weights: [22.00, 0.00] class: 0
|   |   |   |   |--- months_loan_duration >  34.50
|   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |--- job_skilled <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- job_skilled >  0.50
|   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |--- age >  29.50
|   |   |   |   |--- age <= 66.00
|   |   |   |   |   |--- purpose_renovations <= 0.50
|   |   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |   |--- amount <= 4602.50
|   |   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [15.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 16.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- months_loan_duration >  16.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- amount >  4602.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |   |--- purpose_education <= 0.50
|   |   |   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |   |   |--- weights: [98.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |   |   |--- age <= 46.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [18.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- age >  46.00
|   |   |   |   |   |   |   |   |   |   |--- employment_duration <= 4.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- employment_duration >  4.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- purpose_education >  0.50
|   |   |   |   |   |   |   |   |--- job_management <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- job_management >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- purpose_renovations >  0.50
|   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |--- age >  66.00
|   |   |   |   |--- existing_loans_count <= 1.50
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- existing_loans_count >  1.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|--- checking_balance >  0.00
|   |--- months_loan_duration <= 26.50
|   |   |--- credit_history <= 3.50
|   |   |   |--- months_loan_duration <= 11.50
|   |   |   |   |--- amount <= 10931.50
|   |   |   |   |   |--- age <= 29.50
|   |   |   |   |   |   |--- amount <= 1013.50
|   |   |   |   |   |   |   |--- job_skilled <= 0.50
```

```
|   |   |   |   |   |   |   |   |   |--- amount <= 727.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- amount >  727.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- job_skilled >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- amount >  1013.50
|   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |   |--- age >  29.50
|   |   |   |   |   |   |   |--- housing_own <= 0.50
|   |   |   |   |   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |   |   |   |   |--- existing_loans_count <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |--- existing_loans_count >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- percent_of_income >  1.50
|   |   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- housing_own >  0.50
|   |   |   |   |   |   |   |   |--- weights: [40.00, 0.00] class: 0
|   |   |   |   |   |--- amount >  10931.50
|   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |--- months_loan_duration >  11.50
|   |   |   |   |   |--- amount <= 1374.50
|   |   |   |   |   |   |--- purpose_car <= 0.50
|   |   |   |   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |--- credit_history <= 2.00
|   |   |   |   |   |   |   |   |   |   |--- purpose_renovations <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- purpose_renovations >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- credit_history >  2.00
|   |   |   |   |   |   |   |   |   |   |--- purpose_business <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 10.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- purpose_business >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |   |   |   |--- age <= 32.50
|   |   |   |   |   |   |   |   |   |--- weights: [10.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- age >  32.50
|   |   |   |   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |   |   |   |   |   |   |   |   |--- credit_history <= 2.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- credit_history >  2.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- purpose_car >  0.50
|   |   |   |   |   |   |   |--- age <= 45.00
|   |   |   |   |   |   |   |   |--- age <= 22.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- age >  22.50
```

```
|   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 20.00] class: 1
|   |   |   |   |   |   |   |--- age >  45.00
|   |   |   |   |   |   |   |   |--- months_loan_duration <= 22.50
|   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- months_loan_duration >  22.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- amount >  1374.50
|   |   |   |   |   |   |   |--- amount <= 8472.00
|   |   |   |   |   |   |   |   |--- housing_rent <= 0.50
|   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- purpose_business <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |   |--- purpose_business >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- savings_balance <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount <= 4764.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [33.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount >  4764.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |   |--- savings_balance >  3.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount <= 1726.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount >  1726.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |   |   |   |   |   |   |   |--- employment_duration <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- age <= 53.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- age >  53.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- employment_duration >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- amount <= 1931.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- amount >  1931.50
|   |   |   |   |   |   |   |   |   |   |   |--- amount <= 2100.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |   |--- amount >  2100.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 8
|   |   |   |   |   |   |   |--- housing_rent >  0.50
|   |   |   |   |   |   |   |   |--- other_credit_none <= 0.50
|   |   |   |   |   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |   |--- other_credit_none >  0.50
|   |   |   |   |   |   |   |   |   |--- percent_of_income <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
```

```
|   |   |   |   |   |   |   |   |   |   |--- savings_balance <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- savings_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |--- percent_of_income >  2.50
|   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 16.50
|   |   |   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- months_loan_duration >  16.50
|   |   |   |   |   |   |   |   |   |   |--- amount <= 5186.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |--- amount >  5186.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |--- amount >  8472.00
|   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |--- credit_history >  3.50
|   |   |   |--- other_credit_none <= 0.50
|   |   |   |   |--- months_loan_duration <= 15.00
|   |   |   |   |   |--- amount <= 386.00
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- amount >  386.00
|   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |--- months_loan_duration >  15.00
|   |   |   |   |   |--- age <= 40.00
|   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- age >  40.00
|   |   |   |   |   |   |--- employment_duration <= 3.50
|   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- employment_duration >  3.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- other_credit_none >  0.50
|   |   |   |   |--- age <= 41.00
|   |   |   |   |   |--- weights: [0.00, 14.00] class: 1
|   |   |   |   |--- age >  41.00
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |--- months_loan_duration >  26.50
|   |   |--- checking_balance <= 2.50
|   |   |   |--- savings_balance <= 0.00
|   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |--- amount <= 11257.00
|   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |--- amount >  11257.00
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |--- savings_balance >  0.00
|   |   |   |   |--- months_loan_duration <= 47.50
|   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |--- purpose_car <= 0.50
|   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |--- savings_balance <= 1.50
|   |   |   |   |   |   |   |   |   |--- amount <= 3262.00
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  1.50
```

```
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- amount >  3262.00
|   |   |   |   |   |   |   |   |   |   |   |--- age <= 47.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- age >  47.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- savings_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |   |   |   |   |   |--- purpose_car >  0.50
|   |   |   |   |   |   |   |   |--- amount <= 7759.00
|   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  7759.00
|   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 39.00
|   |   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 33.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- months_loan_duration >  33.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |--- months_loan_duration >  39.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |--- age <= 29.50
|   |   |   |   |   |   |   |   |--- amount <= 2364.50
|   |   |   |   |   |   |   |   |   |--- age <= 23.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- age >  23.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  2364.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 12.00] class: 1
|   |   |   |   |   |   |   |--- age >  29.50
|   |   |   |   |   |   |   |   |--- amount <= 2319.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |--- amount >  2319.50
|   |   |   |   |   |   |   |   |   |--- credit_history <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- credit_history >  2.50
|   |   |   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |--- months_loan_duration >  47.50
|   |   |   |   |   |   |--- amount <= 3705.00
|   |   |   |   |   |   |   |--- credit_history <= 3.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- credit_history >  3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- amount >  3705.00
|   |   |   |   |   |   |   |--- weights: [0.00, 18.00] class: 1
|   |   |   |--- checking_balance >  2.50
|   |   |   |   |--- amount <= 2462.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- amount >  2462.50
|   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
```

Bagging Classifier

- ***Bagging refers to bootstrap sampling and aggregation. This means that in bagging at the beginning samples are chosen randomly with replacement to train the individual models and then model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes.***

- ***Bagging makes the model more robust since the final prediction is made on the basis of a number of outputs that have been given by a large number of independent models. It prevents overfitting the model to the original data since the individual models do not have access to the original data and are only built on samples that have been randomly chosen from the original data with replacement. Bagging follows the parallel model building i.e the output of individual models is independent of each other.***

Some of the important hyperparameters available for bagging classifier are:

- base_estimator: The base estimator to fit on random subsets of the dataset. If None(default), then the base estimator is a decision tree.
- n_estimators: The number of trees in the forest, default = 100.
- max_features: The number of features to consider when looking for the best split.
- bootstrap: Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- bootstrap_features: If it is true, then features are drawn with replacement. Default value is False.
- max_samples: If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.
- oob_score: Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.

In [233…

```
bagging_classifier = BaggingClassifier(
    random_state=1
)  ## Complete the code to define bagging classifier with random state = 1
bagging_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit bagging classifier on the train data
```

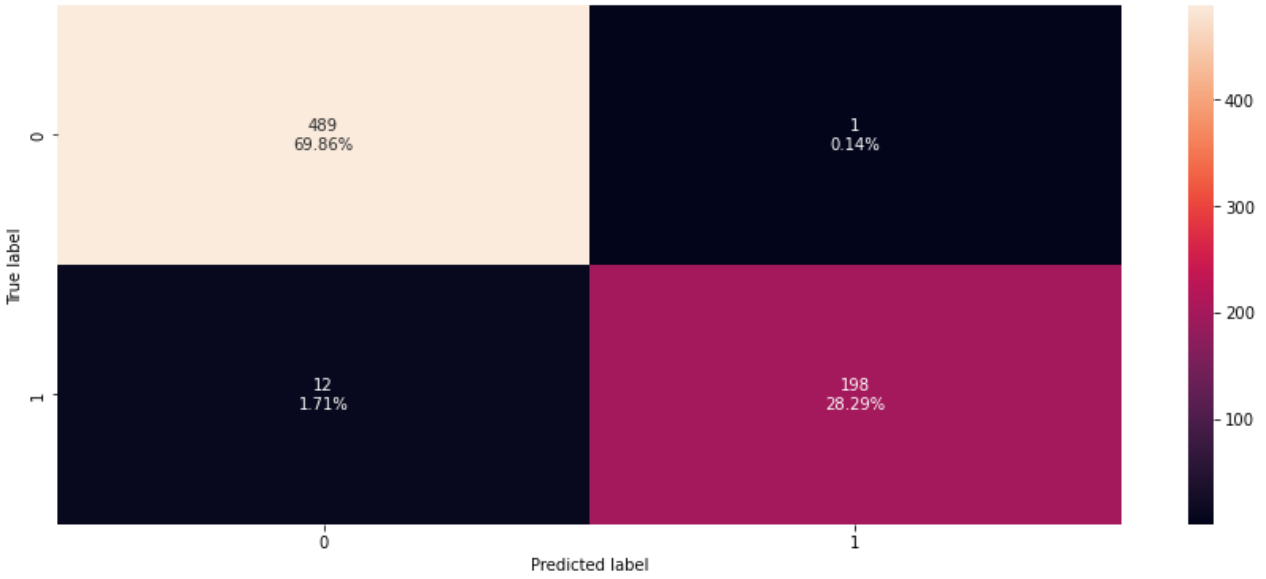
Out[233…

```
BaggingClassifier
BaggingClassifier(random_state=1)
```

Checking model performance on training set

In [234…

```
confusion_matrix_sklearn(
    bagging_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [235…

```
bagging_classifier_model_train_perf = model_performance_classification_sklearn(
    bagging_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
bagging_classifier_model_train_perf
```

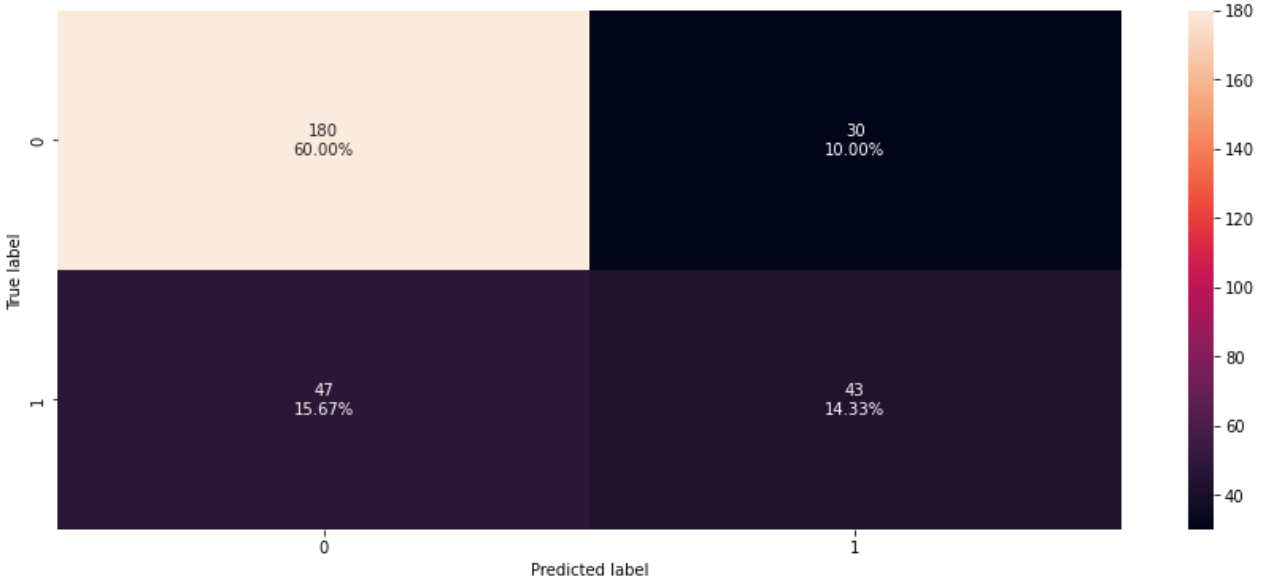
Out[235…

	Accuracy	Recall	Precision	F1
0	0.98	0.94	0.99	0.97

Checking model performance on test set

In [236…

```
confusion_matrix_sklearn(
    bagging_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [237…
```
bagging_classifier_model_test_perf = model_performance_classification_sklearn(bagging_c
bagging_classifier_model_test_perf
```

Out[237…

	Accuracy	Recall	Precision	F1
0	0.74	0.48	0.59	0.53

- The overfitting has decrease slightly in the training data
- The test model performance is lower than in hyperparamenter tuned Decision tree

---

Method2: Bagging Classifier - Weighted Decision Tree

Bagging Classifier with weighted decision tree

In [238…
```
bagging_wt = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(
        criterion="gini", class_weight={0: 0.17, 1: 0.83}, random_state=1
    ),
    random_state=1,
)
bagging_wt.fit(X_train, Y_train)
```

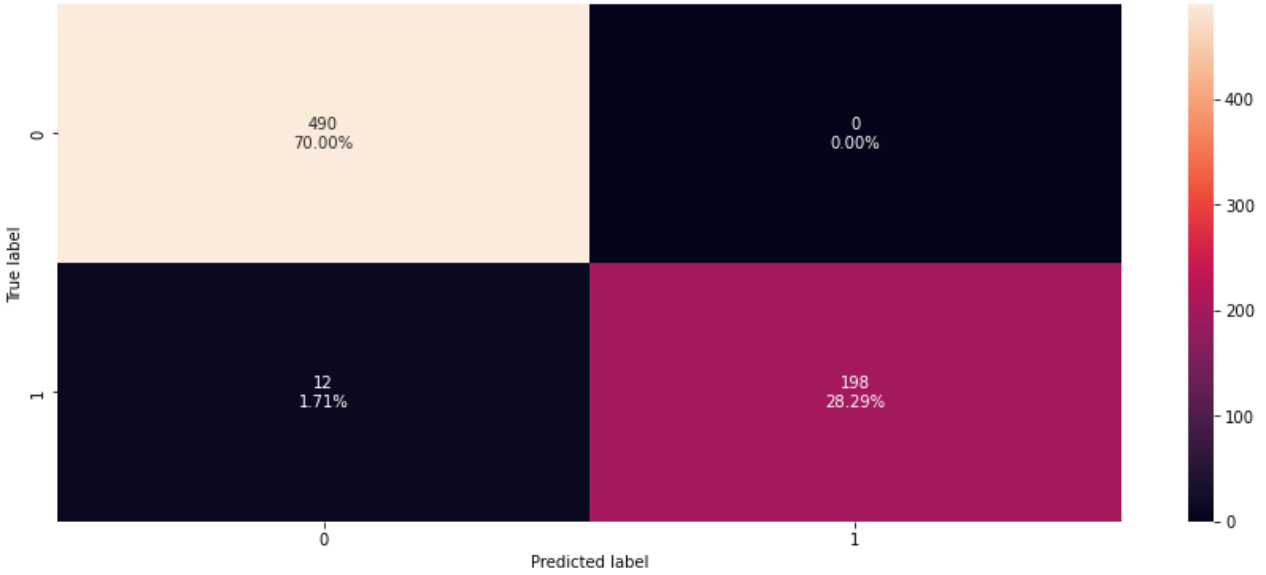
Out[238…

BaggingClassifier

base_estimator: DecisionTreeClassifier

DecisionTreeClassifier

In [239…
```
confusion_matrix_sklearn(bagging_wt, X_train, Y_train)
```



In [240…

```
# Training Performance Measures
bagging_wt_model_train_perf = model_performance_classification_sklearn(
    bagging_wt, X_train, Y_train
)
print("Training performance \n")
bagging_wt_model_train_perf
```

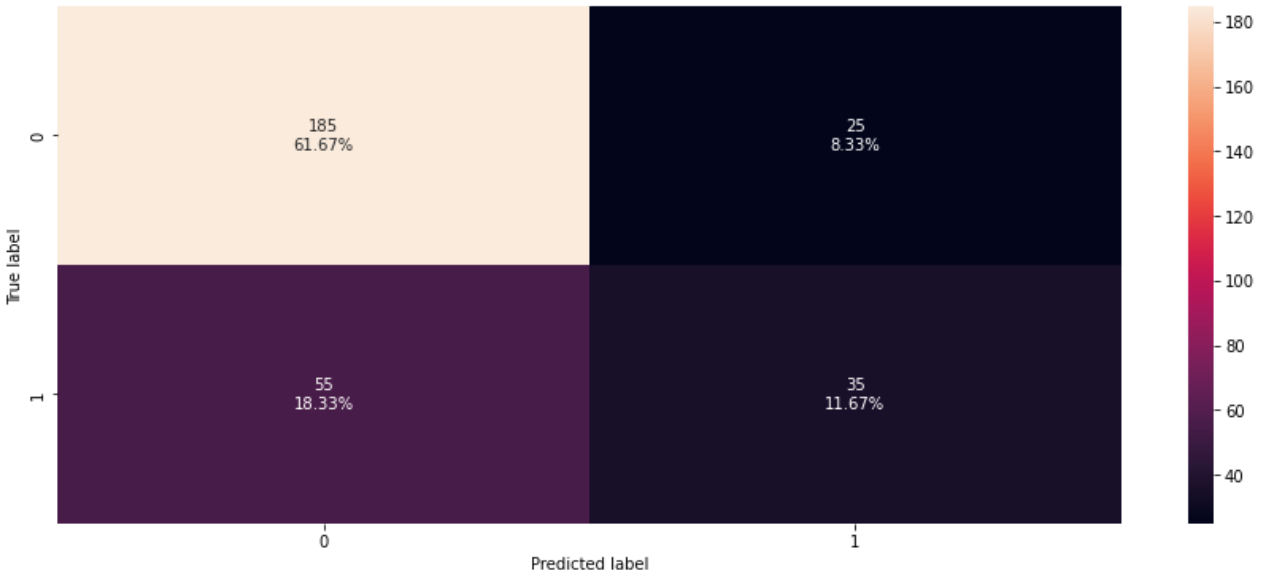
```
Training performance
```

Out[240…

	Accuracy	Recall	Precision	F1
0	0.98	0.94	1.00	0.97

In [241…

```
confusion_matrix_sklearn(bagging_wt, X_test, Y_test)
```



In [242…

```
# Testing Performance Measures
```

```
bagging_wt_model_test_perf = model_performance_classification_sklearn(
    bagging_wt, X_test, Y_test
)
print("Testing performance \n")
bagging_wt_model_test_perf
```

```
Testing performance
```

Out[242...

	Accuracy	Recall	Precision	F1
0	0.73	0.39	0.58	0.47

Hyperparameter Tuning - Bagging Classifier

Bagging Classifier

Some of the important hyperparameters available for bagging classifier are:

- `base_estimator:` The base estimator to fit on random subsets of the dataset. If None(default), then the base estimator is a decision tree.
- `n_estimators:` The number of trees in the forest, default = 100. max_features: The number of features to consider when looking for the best split.
- `bootstrap:` Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- `bootstrap_features:` If it is true, then features are drawn with replacement. Default value is False.
- `max_samples:` If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.
- `oob_score:` Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.

In [243...

```
# Choose the type of classifier.
bagging_estimator_tuned = BaggingClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    "max_samples": [0.7, 0.8, 0.9],
    "max_features": [0.7, 0.8, 0.9],
    "n_estimators": np.arange(90, 120, 10),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
```

```
grid_obj = GridSearchCV(
    bagging_estimator_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
bagging_estimator_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
bagging_estimator_tuned.fit(X_train, Y_train)
```

Out[243…

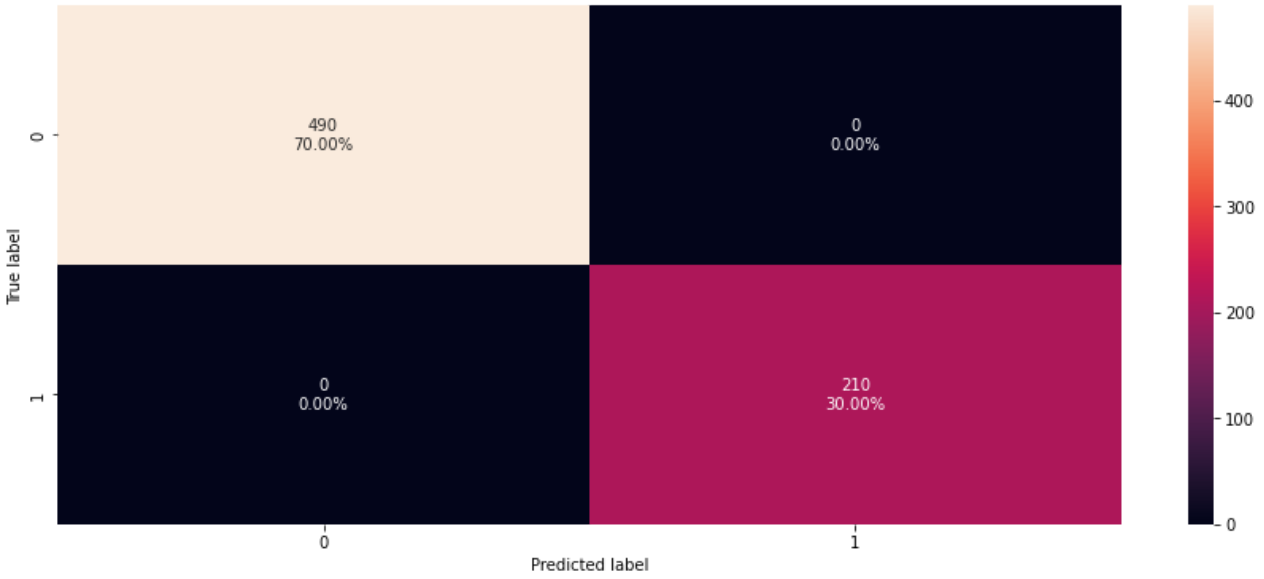
BaggingClassifier

```
BaggingClassifier(max_features=0.9, max_samples=0.9, n_estimators=110,
                  random_state=1)
```

Checking model performance on training set

In [244…

```
confusion_matrix_sklearn(
    bagging_estimator_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [245…

```
# Training Performance Mearsures
bagging_estimator_tuned_model_train_perf = model_performance_classification_sklearn(
    bagging_estimator_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
bagging_estimator_tuned_model_train_perf
```

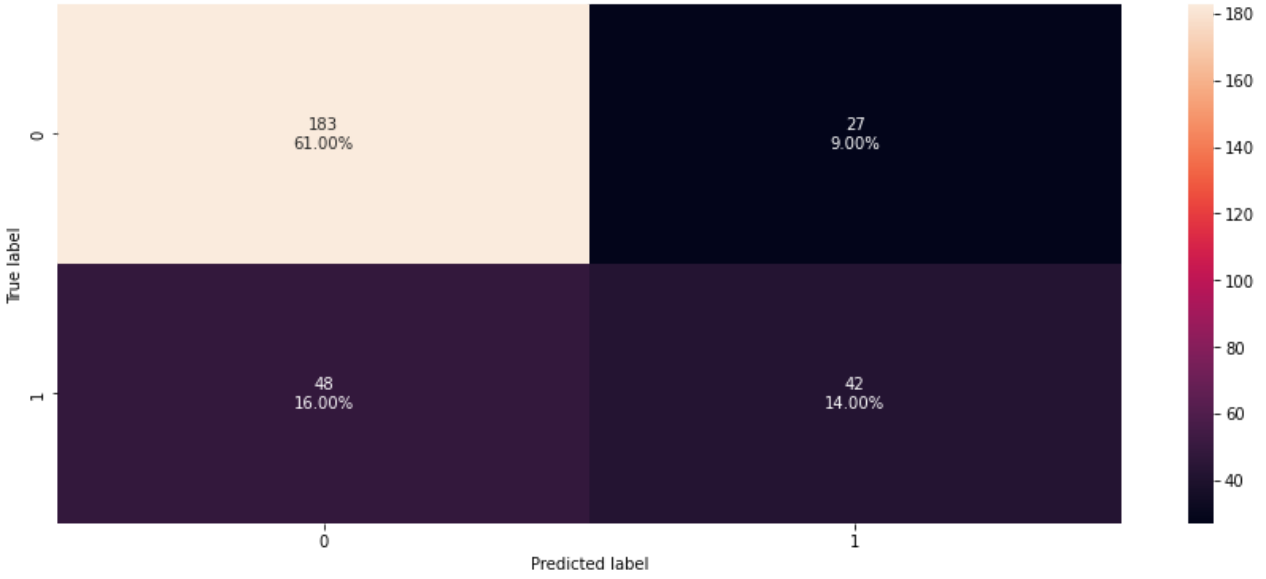
Out[245…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Checking model performance on test set

In [246…

```
confusion_matrix_sklearn(
    bagging_estimator_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [247…

```
# Testing Performce Measures
bagging_estimator_tuned_model_test_perf = model_performance_classification_sklearn(
    bagging_estimator_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
bagging_estimator_tuned_model_test_perf
```

Out[247…

	Accuracy	Recall	Precision	F1
0	0.75	0.47	0.61	0.53

The model performance has increased but the training data is still overfitting

Insights

- We can see that train accuracy and recall for the bagging classifier have increased slightly after hyperparameter tuning but the test recall has decreased.
- The model is overfitting the data, as train accuracy and recall are much higher than the test accuracy and test recall.
- The confusion matrix shows that the model is better at identifying non-defaulters as compared to defaulters.

Method 2: Logistic Regression as the base estimator for Bagging Classifier

Let's try using logistic regression as the base estimator for bagging

classifier:

- Now, let's try and change the `base_estimator` of the bagging classifier, which is a decision tree by default.
- We will pass the logistic regression as the base estimator for bagging classifier.

In [248...

```
bagging_lr = BaggingClassifier(
    base_estimator=LogisticRegression(
        solver="liblinear", random_state=1, max_iter=1000
    ),
    random_state=1,
)
bagging_lr.fit(X_train, Y_train)
```

Out[248...

BaggingClassifier

base_estimator: LogisticRegression

LogisticRegression

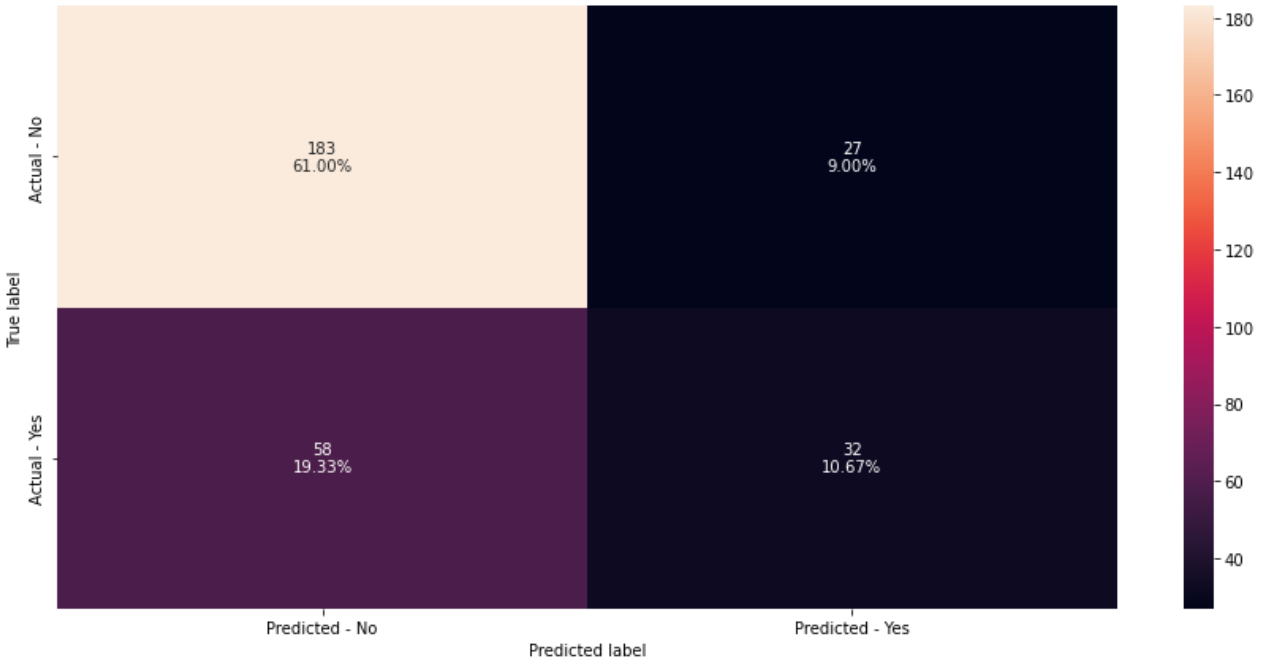
In [249...

```
# Using above defined function to get accuracy, recall and precision on train and test
bagging_lr_score = get_metrics_score(bagging_lr)
```

```
Accuracy on training set :  0.7414285714285714
Accuracy on test set :  0.7166666666666667
Recall on training set :  0.3380952380952381
Recall on test set :  0.35555555555555557
Precision on training set :  0.6283185840707964
Precision on test set :  0.5423728813559322
```

In [250...

```
make_confusion_matrix(bagging_lr, Y_test)
```



Insights

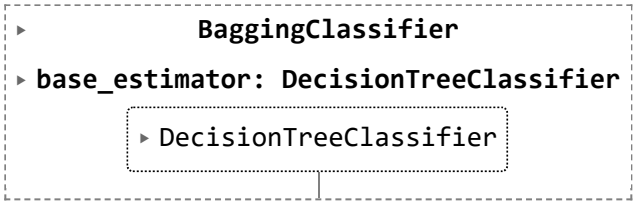
- Bagging classifier with logistic regression as base_estimator is not overfitting the data but the test recall is very low.
- Ensemble models are less interpretable than decision tree but bagging classifier is even less interpretable than random forest. It does not even have a feature importance attribute.

Tuning Bagging Classifier- Weighte Model

In [251…

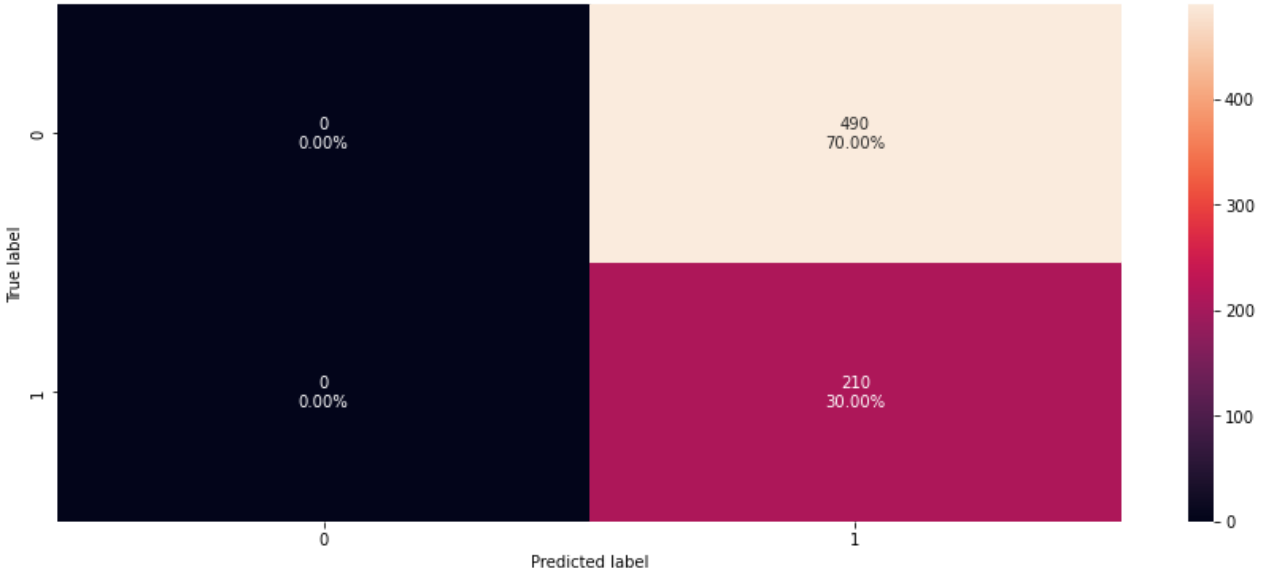
```
# grid search for bagging classifier
bagging_estimator_weighted = DecisionTreeClassifier(
    class_weight={0: 0.13, 1: 0.87}, random_state=1
)
param_grid = {
    "base_estimator": [bagging_estimator_weighted],
    "n_estimators": [5, 7, 15, 51, 101],
    "max_features": [0.7, 0.8, 0.9, 1],
}

grid = GridSearchCV(
    BaggingClassifier(random_state=1, bootstrap=True),
    param_grid=param_grid,
    scoring="recall",
    cv=5,
)
grid.fit(X_train, Y_train)

## getting the best estimator
bagging_estimator = grid.best_estimator_
bagging_estimator.fit(X_train, Y_train)
```

Out[251…



In [252…

```
confusion_matrix_sklearn(bagging_estimator, X_train, Y_train)
```



In [253…

```
# Training Performance Measures
bagging_estimator_model_train_perf = model_performance_classification_sklearn(
    bagging_estimator, X_train, Y_train
)
print("Training performance: \n ")
bagging_estimator_model_train_perf
```

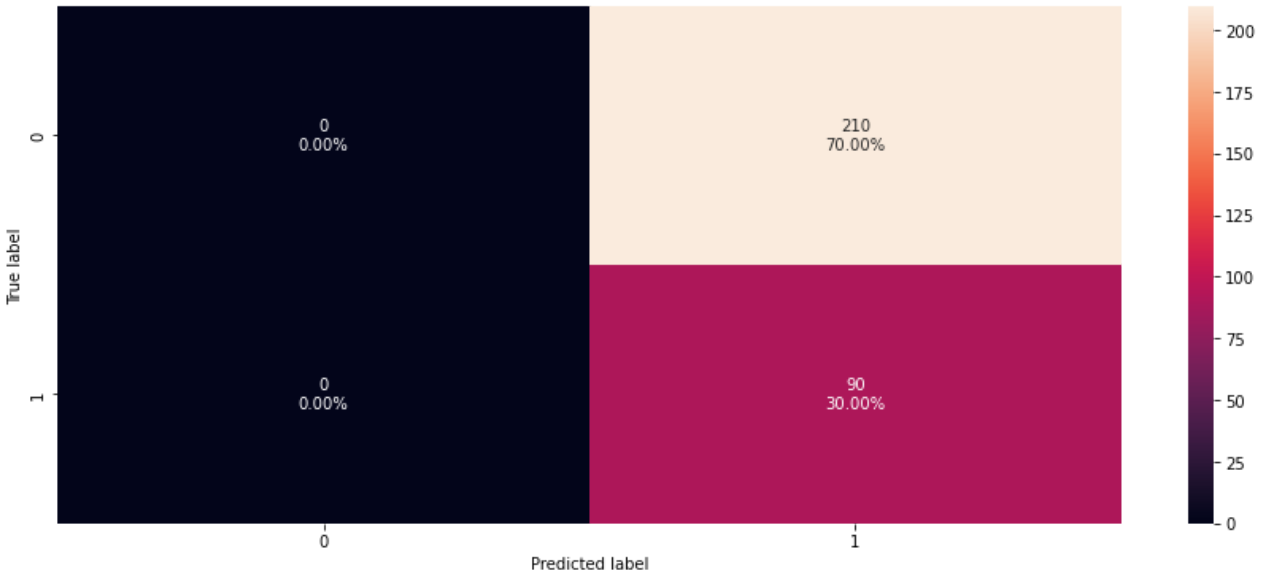
Training performance:

Out[253…

	Accuracy	Recall	Precision	F1
0	0.30	1.00	0.30	0.46

In [254…

```
confusion_matrix_sklearn(bagging_estimator, X_test, Y_test)
```



In [255…

```
# Testing Performance Mearsures
```

```
bagging_estimator_model_test_perf = model_performance_classification_sklearn(
    bagging_estimator, X_test, Y_test
)
print("Testing performance \n")
bagging_estimator_model_test_perf
```

```
Testing performance
```

Out[255...

	Accuracy	Recall	Precision	F1
0	0.30	1.00	0.30	0.46

- Recall has improved but the accuracy and precision of the model has dropped drastically which is an indication that overall the model is making many mistakes.

Random Forest

- ***Random forest randomly picks a subset of independent variables for each node's split, where m is the size of the subset and M is the total number of independent variables, where m is generally less than M. This is done to make the individual trees even more independent/different from each other and incorporate more diversity in our final prediction thereby, making the entire model more robust.***
- ***In Random Forest, to get different n-models with the same algorithm, we use Bootstrap aggregation. This means that at the beginning samples are chosen randomly with replacement to train the individual models and then model predictions undergo aggregation to combine them for the final prediction to consider all the possible outcomes.***
- ***The problem of overfitting in a decision tree can be overcome by random forest since the individual trees in a random forest do not have access to the original dataset and are only built on observations that have been sampled with replacement from the original dataset.***
- ***Since the random forest uses multiple tree models to reach a final prediction, it is more robust than a single decision tree model and prevents instabilities due to changes in data. Random forest is less interpretable and has higher computational complexity than decision trees as it utilizes multiple tree models to reach a prediction.***
- ***Random forest prevents overfitting since the individual trees in a random forest do not have access to the original dataset and are only built on observations that have been sampled with replacement from the original dataset. Moreover, aggregation of results from different trees in a random forest reduces the chances of overfitting and so there is no need to prune a random forest.***

- ***In a classification setting, for a new test data point, the final prediction by a random forest is done by taking the mode of the individual predictions while in a regression setting, for a new test data point, the final prediction by a random forest is done by taking the average of individual predictions.***

Random Forest Classifier

Now, let's see if we can get a better model by tuning the random forest classifier. Some of the important hyperparameters available for random forest classifier are:

- n_estimators: The number of trees in the forest, default = 100.
- max_features: The number of features to consider when looking for the best split.
- class_weight: Weights associated with classes in the form {class_label: weight}.If not given, all classes are supposed to have weight one.
- For example: If the frequency of class 0 is 80% and the frequency of class 1 is 20% in the data, then class 0 will become the dominant class and the model will become biased toward the dominant classes. In this case, we can pass a dictionary {0:0.2,1:0.8} to the model to specify the weight of each class and the random forest will give more weightage to class 1.
- bootstrap: Whether bootstrap samples are used when building trees. If False, the entire dataset is used to build each tree, default=True.
- max_samples: If bootstrap is True, then the number of samples to draw from X to train each base estimator. If None (default), then draw N samples, where N is the number of observations in the train data.
- oob_score: Whether to use out-of-bag samples to estimate the generalization accuracy, default=False.

- Note: A lot of hyperparameters of Decision Trees are also available to tune Random Forest like max_depth, min_sample_split etc.

In [256…

```
# Fitting the model
rf_estimator = RandomForestClassifier(
    random_state=1
)  ## Complete the code to define random forest with random state = 1 and class_weight
rf_estimator.fit(
    X_train, Y_train
)  ## Complete the code to fit random forest on the train data
```

Out[256…

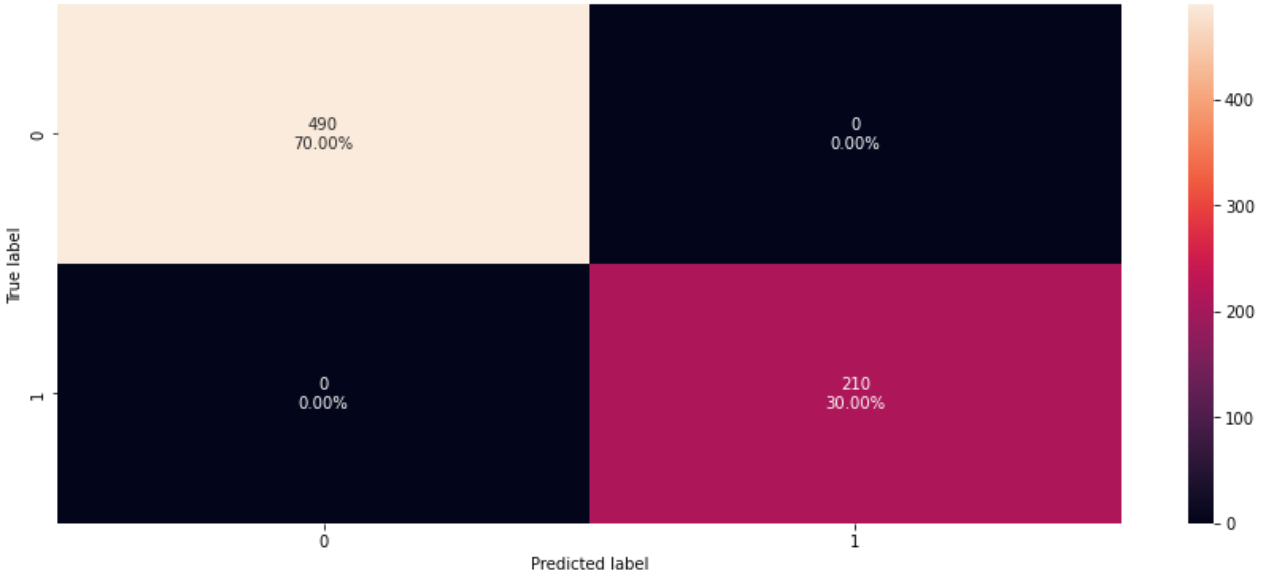
```
▾        RandomForestClassifier

RandomForestClassifier(random_state=1)
```

Checking model performance on training set

In [257…

```
confusion_matrix_sklearn(
    rf_estimator, X_train, Y_train
```

```
)  ## Complete the code to create confusion matrix for train data
```



In [258…

```
# Training Performance Measures
rf_estimator_model_train_perf = model_performance_classification_sklearn(
    rf_estimator, X_train, Y_train
)  ## Complete the code to check performance on train data
rf_estimator_model_train_perf
```

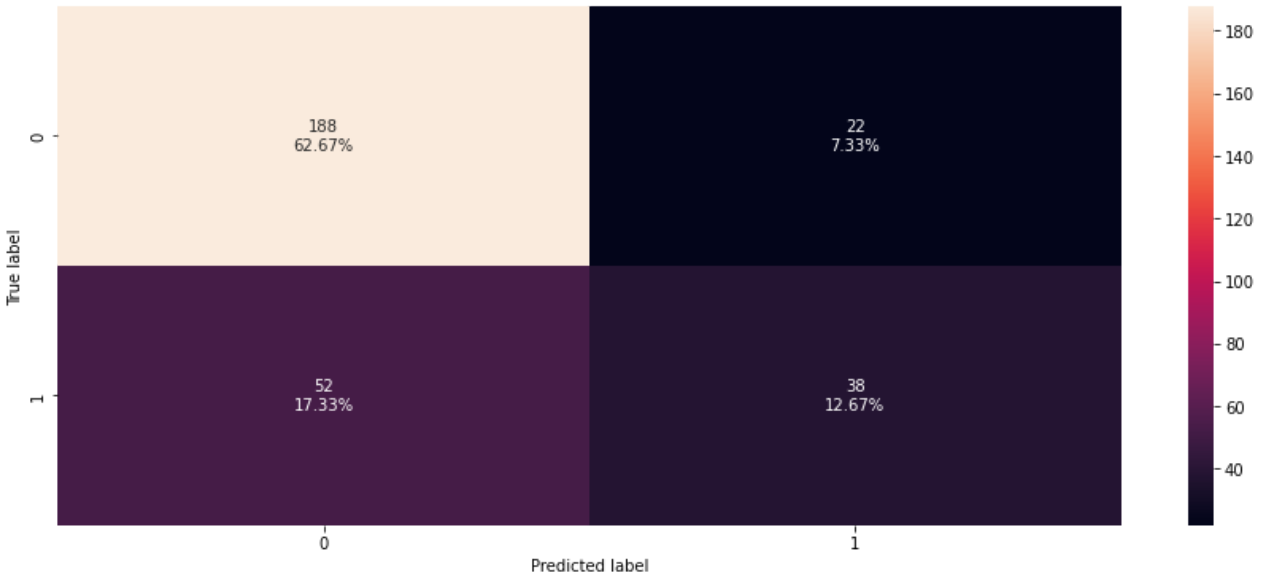
Out[258…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Checking model performance on test set

In [259…

```
confusion_matrix_sklearn(
    rf_estimator, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [260…

```
# Testing Prformance Measures
rf_estimator_model_test_perf = model_performance_classification_sklearn(
    rf_estimator, X_test, Y_test
)  ## Complete the code to check performance for test data
rf_estimator_model_test_perf
```

Out[260…

	Accuracy	Recall	Precision	F1
0	0.75	0.42	0.63	0.51

Random Forest -Weighted Class

Random forest with class weights

In [261…

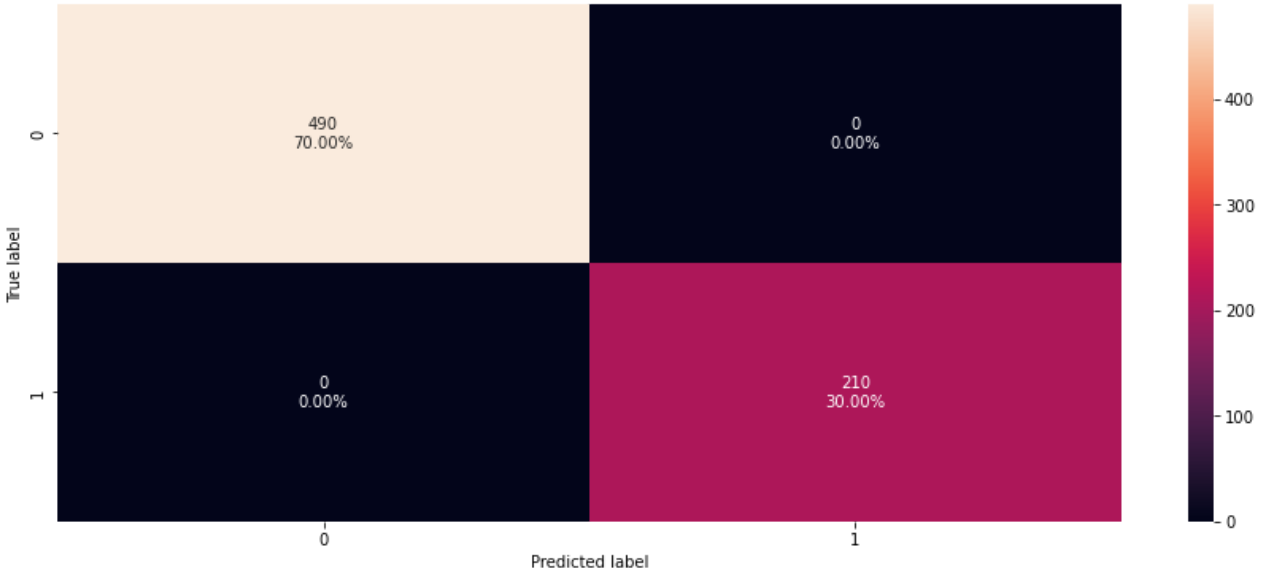
```
rf_wt = RandomForestClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
rf_wt.fit(X_train, Y_train)
```

Out[261…

```
RandomForestClassifier(class_weight={0: 0.17, 1: 0.83}, random_state=1)
```

In [262…

```
confusion_matrix_sklearn(rf_wt, X_train, Y_train)
```



In [263…

```
# Training Performance Measures
rf_wt_model_train_perf = model_performance_classification_sklearn(
    rf_wt, X_train, Y_train
)
print("Training performance \n")
rf_wt_model_train_perf
```

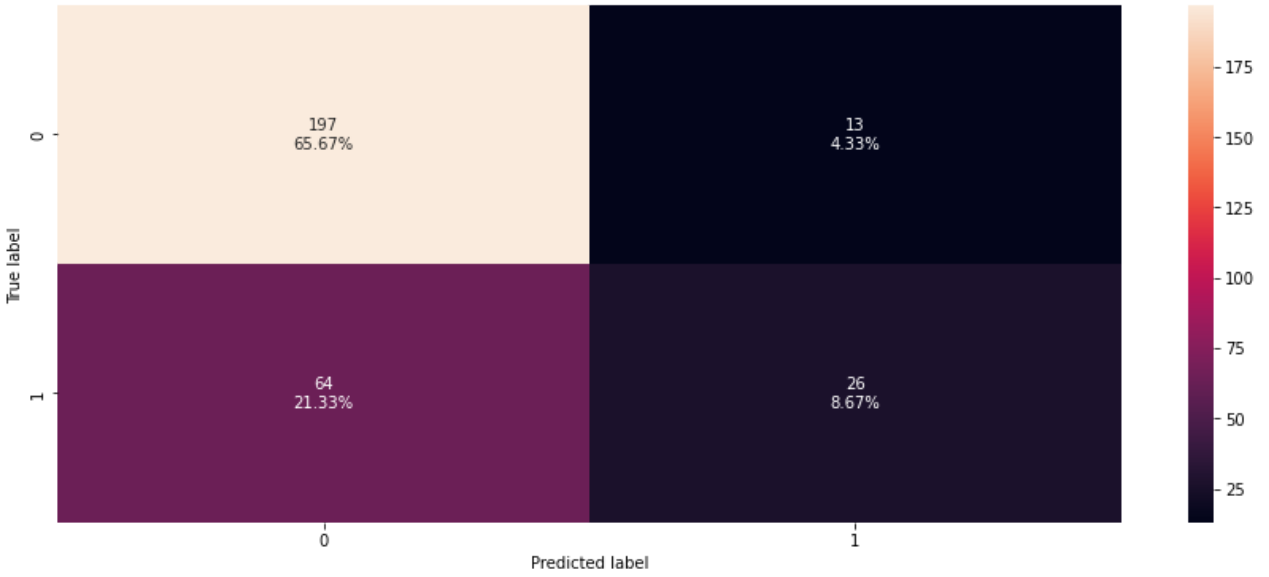
```
Training performance
```

Out[263...

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

In [264...

```
confusion_matrix_sklearn(rf_wt, X_test, Y_test)
```



In [265...

```
# Testing Performance Measures
rf_wt_model_test_perf = model_performance_classification_sklearn(rf_wt, X_test, Y_test)
print("Testing performance \n")

rf_wt_model_test_perf
```

```
Testing performance
```

Out[265...

	Accuracy	Recall	Precision	F1
0	0.74	0.29	0.67	0.40

- There is not much improvement in metrics of weighted random forest as compared to the unweighted random forest.

- Random forest is overfitting the training data as there is a huge difference between training and test scores for all the metrics.
- The test recall is even lower than the decision tree but has a higher test precision.

Hyperparameter Tuning - Random Forest

- ***The class_weight is the hyperparameter of Random Forest which is useful in dealing with***

imbalanced data by giving more importance to the minority class. By giving more class_weight to a certain class than the other class, we tell the model that it is more important to correctly predict a certain class than the other class.

Let's try using class_weights for random forest:

- The model performance is not very good. This may be due to the fact that the classes are imbalanced with 70% non-defaulters and 30% defaulters.
- We should make the model aware that the class of interest here is 'defaulters'.
- We can do so by passing the parameter `class_weights` available for random forest. This parameter is not available for the bagging classifier.
- class_weight specifies the weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.
- We can choose class_weights={0:0.3,1:0.7} because that is the original imbalance in our data.

In [266…

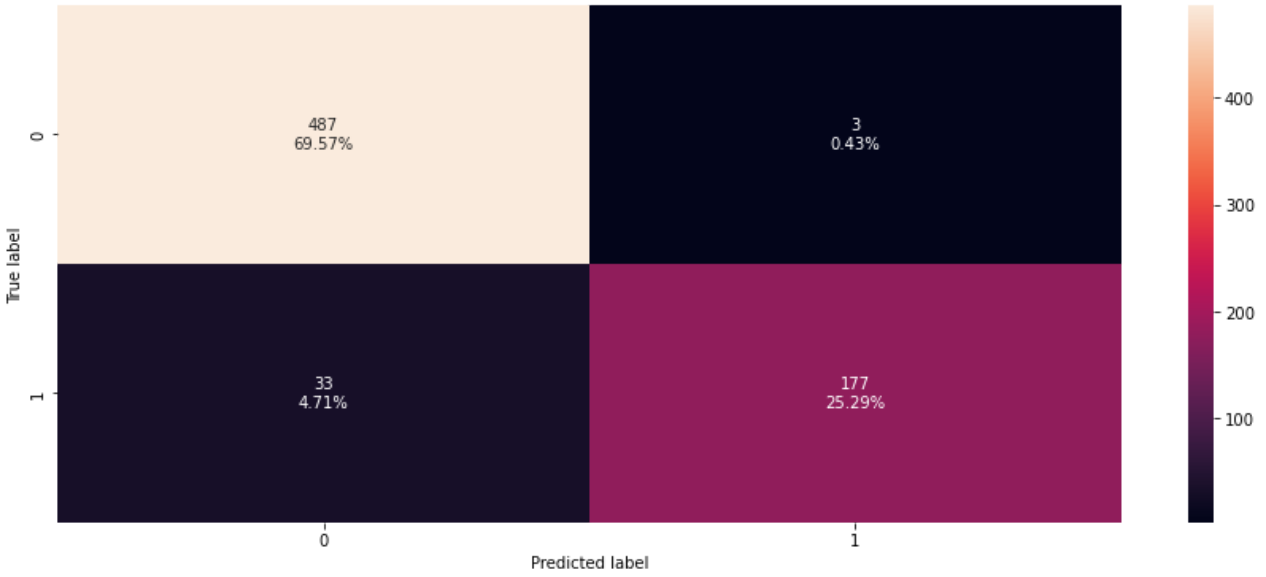
```
# Choose the type of classifier.
rf_tuned = RandomForestClassifier(random_state=1, oob_score=True, bootstrap=True)

parameters = {
    "max_depth": list(np.arange(5, 15, 5)),
    "max_features": ["sqrt", "log2"],
    "min_samples_split": [3, 5, 7],
    "n_estimators": np.arange(10, 40, 10),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    rf_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5 and n_jobs = -1
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on the train data

# Set the clf to the best combination of parameters
rf_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_tuned.fit(X_train, Y_train)
```

Out[266…

```
RandomForestClassifier(max_depth=10, min_samples_split=3, n_estimators=10,
                       oob_score=True, random_state=1)
```

Checking model performance on training set

In [267…
```
confusion_matrix_sklearn(
    rf_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



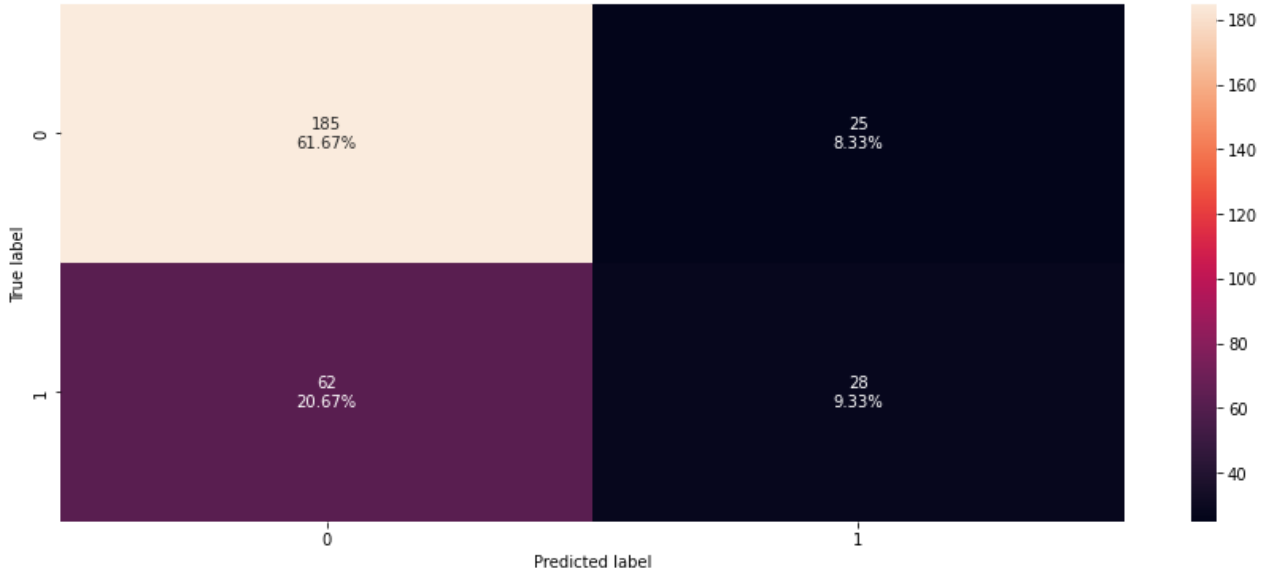
In [268…
```
# Training Performance Measures
rf_tuned_model_train_perf = model_performance_classification_sklearn(
    rf_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
rf_tuned_model_train_perf
```

Out[268…

	Accuracy	Recall	Precision	F1
0	0.95	0.84	0.98	0.91

Checking model performance on test set

In [269…
```
confusion_matrix_sklearn(
    rf_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [270…

```
# Test Performance Measures
rf_tuned_model_test_perf = model_performance_classification_sklearn(
    rf_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
rf_tuned_model_test_perf
```

Out[270…

	Accuracy	Recall	Precision	F1
0	0.71	0.31	0.53	0.39

Method 2

Class_Weights for Random Forest - Hyperparameter Tuning

Let's try using class_weights for random forest:

- The model performance is not very good. This may be due to the fact that the classes are imbalanced with 70% non-defaulters and 30% defaulters.
- We should make the model aware that the class of interest here is 'defaulters'.
- We can do so by passing the parameter `class_weights` available for random forest. This parameter is not available for the bagging classifier.
- class_weight specifies the weights associated with classes in the form {class_label: weight}. If not given, all classes are supposed to have weight one.
- We can choose class_weights={0:0.3,1:0.7} because that is the original imbalance in our data.

In [271…

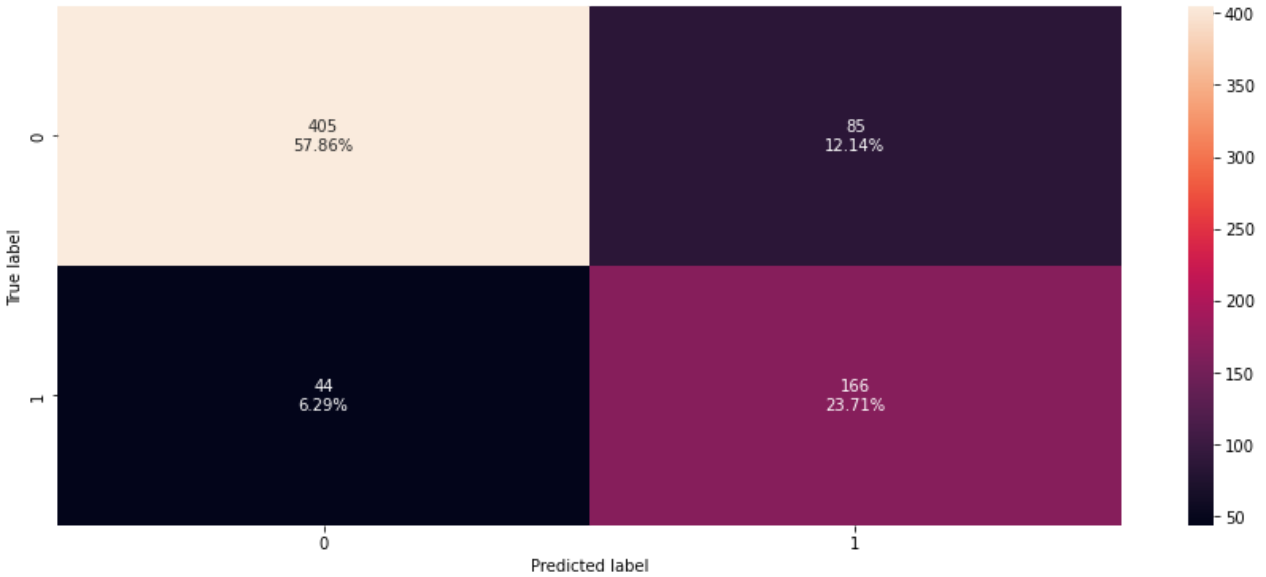
```
# Choose the type of classifier.
rf_estimator_weighted = RandomForestClassifier(random_state=1)

# Grid of parameters to choose from
## add from article
parameters = {
    "class_weight": [{0: 0.3, 1: 0.7}],
    "n_estimators": [100, 150, 200, 250],
    "min_samples_leaf": np.arange(5, 10),
    "max_features": np.arange(0.2, 0.7, 0.1),
    "max_samples": np.arange(0.3, 0.7, 0.1),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.recall_score)

# Run the grid search
grid_obj = GridSearchCV(rf_estimator_weighted, parameters, scoring=acc_scorer, cv=5)
grid_obj = grid_obj.fit(X_train, Y_train)

# Set the clf to the best combination of parameters
rf_estimator_weighted = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rf_estimator_weighted.fit(X_train, Y_train)
```

Out[271…

```
RandomForestClassifier(class_weight={0: 0.3, 1: 0.7}, max_features=0.2,
                       max_samples=0.5, min_samples_leaf=7, random_state=1)
```

In [272…

```
confusion_matrix_sklearn(rf_estimator_weighted, X_train, Y_train)
```



In [273…

```
# Training Performance Measures
rf_wt_model_train_perf = model_performance_classification_sklearn(
    rf_estimator_weighted, X_train, Y_train
```

```
)
print("Training performance \n")
rf_wt_model_train_perf
```

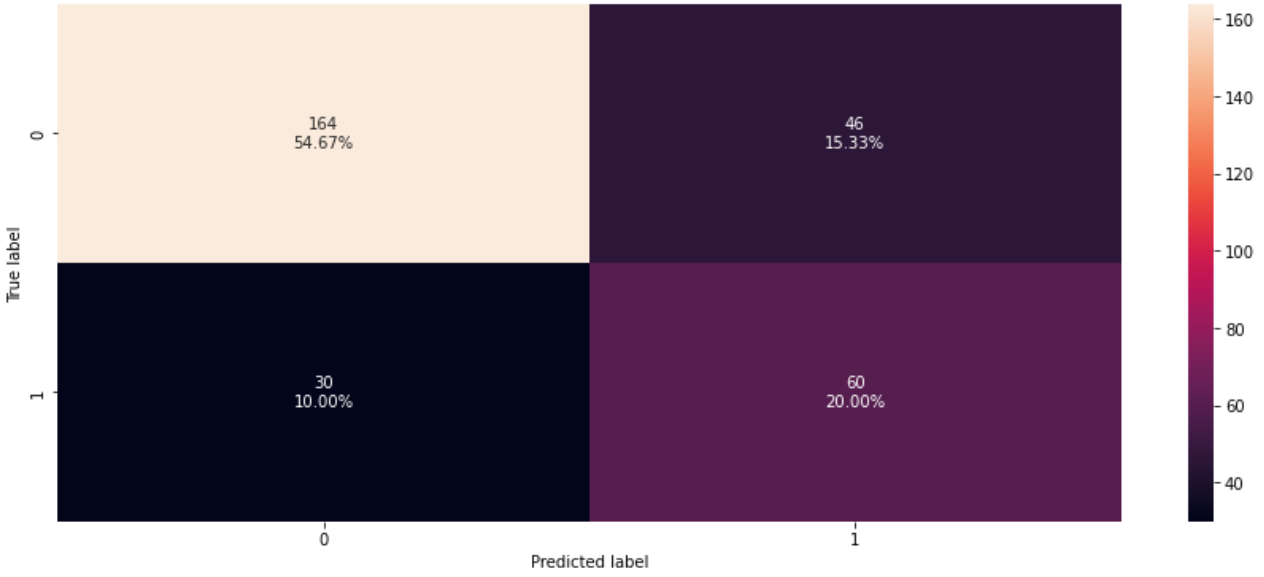
```
Training performance
```

Out[273...

	Accuracy	Recall	Precision	F1
0	0.82	0.79	0.66	0.72

In [274...

```
confusion_matrix_sklearn(rf_estimator_weighted, X_test, Y_test)
```



In [275...

```
# Testing Performance Measures
rf_wt_model_test_perf = model_performance_classification_sklearn(
    rf_estimator_weighted, X_test, Y_test
)
print("Testing performance \n")
rf_wt_model_test_perf
```

```
Testing performance
```

Out[275...

	Accuracy	Recall	Precision	F1
0	0.75	0.67	0.57	0.61

- Random forest after tuning has given same performance as un-tuned random forest.

Insights

- The model accuracy has decreased a bit but the overfitting has also been reduced and the model is generalizing well.
- The train and test recall both have increased significantly.

- We can see from the confusion matrix that the random forest model with class weights is now better at identifying the defaulters as compared to other models.

Important Features for Predicting Loan Default - Random Forest Model

In [276…

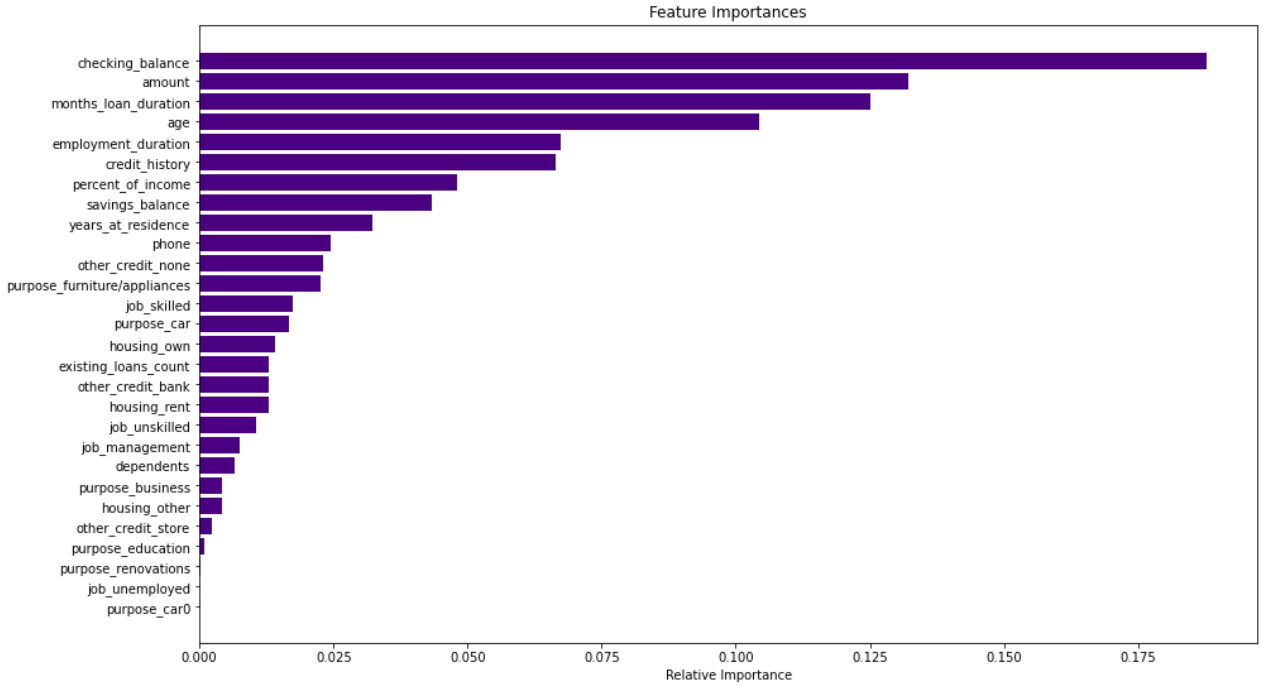
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        rf_estimator_weighted.feature_importances_,
        columns=["Imp"],
        index=X_train.columns,
    ).sort_values(by="Imp", ascending=False)
)
```

```
                              Imp
checking_balance             0.19
amount                       0.13
months_loan_duration         0.13
age                          0.10
employment_duration          0.07
credit_history               0.07
percent_of_income            0.05
savings_balance              0.04
years_at_residence           0.03
phone                        0.02
other_credit_none            0.02
purpose_furniture/appliances 0.02
job_skilled                  0.02
purpose_car                  0.02
housing_own                  0.01
existing_loans_count         0.01
other_credit_bank            0.01
housing_rent                 0.01
job_unskilled                0.01
job_management               0.01
dependents                   0.01
purpose_business             0.00
housing_other                0.00
other_credit_store           0.00
purpose_education            0.00
purpose_renovations          0.00
job_unemployed               0.00
purpose_car0                 0.00
```

In [277…

```
importances = rf_estimator_weighted.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(range(len(indices)), importances[indices], color="indigo", align="center")
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
```

```
plt.xlabel("Relative Importance")
plt.show()
```



- Random forest is overfitting the training data
- Model performance slightly lower than tuned Bagging Classifier

- Overfitting is pretty much reduced in tuned Random Forest model
- F1 score improved but precision has slight reduced
- The confustion matrix shows that the model can predict visa certification

The hyperparameter tuning on Random forest reduces the overfitting

---

Summary Performance Measures of Bagging Models Train vs Test Models

---

In [278…

```
# Performance comparison of Bagging models

bagging_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        decision_tree_perf_test.T,
        rf_estimator_model_train_perf.T,
        rf_estimator_model_test_perf.T,
        bagging_classifier_model_train_perf.T,
```

```
        bagging_classifier_model_test_perf.T,
    ],
    axis=1,
)
bagging_comp_df.columns = [
    "Decision Tree (train)",
    "Decision Tree (test)",
    "Random Forest (train)",
    "Random Forest (test)",
    "Bagging Classifier (train)",
    "Bagging Classifier (test)",
]
print("Training performance comparison:")
bagging_comp_df
```

```
Training performance comparison:
```

Out[278...

	Decision Tree (train)	Decision Tree (test)	Random Forest (train)	Random Forest (test)	Bagging Classifier (train)	Bagging Classifier (test)
Accuracy	1.00	0.69	1.00	0.75	0.98	0.74
Recall	1.00	0.47	1.00	0.42	0.94	0.48
Precision	1.00	0.48	1.00	0.63	0.99	0.59
F1	1.00	0.47	1.00	0.51	0.97	0.53

Summary Performance Measures of Bagging Models Tuning Train vs Tuning Test Models

---

In [279...

```
bagging_tuned_comp_df = pd.concat(
    [
        dtree_estimator_model_train_perf.T,
        dtree_estimator_model_test_perf.T,
        rf_tuned_model_train_perf.T,
        rf_tuned_model_test_perf.T,
        bagging_estimator_tuned_model_train_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
    ],
    axis=1,
)
bagging_tuned_comp_df.columns = [
    "Decision Tree Tuned(train)",
    "Decision Tree Tuned(test)",
    "Random Forest Tuned(train)",
    "Random Forest Tuned(test)",
    "Bagging Classifier Tuned (train)",
    "Bagging Classifier Tuned (test)",
    ]
print("Bagging tuned model performance comparison:")
bagging_tuned_comp_df
```

```
Bagging tuned model performance comparison:
```

Out[279...

	Decision Tree Tuned(train)	Decision Tree Tuned(test)	Random Forest Tuned(train)	Random Forest Tuned(test)	Bagging Classifier Tuned (train)	Bagging Classifier Tuned (test)
Accuracy	0.30	0.30	0.95	0.71	1.00	0.75
Recall	1.00	1.00	0.84	0.31	1.00	0.47
Precision	0.30	0.30	0.98	0.53	1.00	0.61
F1	0.46	0.46	0.91	0.39	1.00	0.53

Note:

- Hyperparameter tuning is tricky and exhaustive in the sense that there is no direct way to calculate how a change in the hyperparameter value will reduce the loss of your model until you try those hyperparameters.
- The final results depend on the parameters used/checked using GridSearchCV.
- There may be yet better parameters which may result in a better accuracy and recall. Students can explore this further.

Boosting Decision Tree Models

AdaBoost Classifier

In [280…

```
ab_classifier = AdaBoostClassifier(
    random_state=1
)  ## Complete the code to define AdaBoost Classifier with random state = 1
ab_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit AdaBoost Classifier on the train data
```

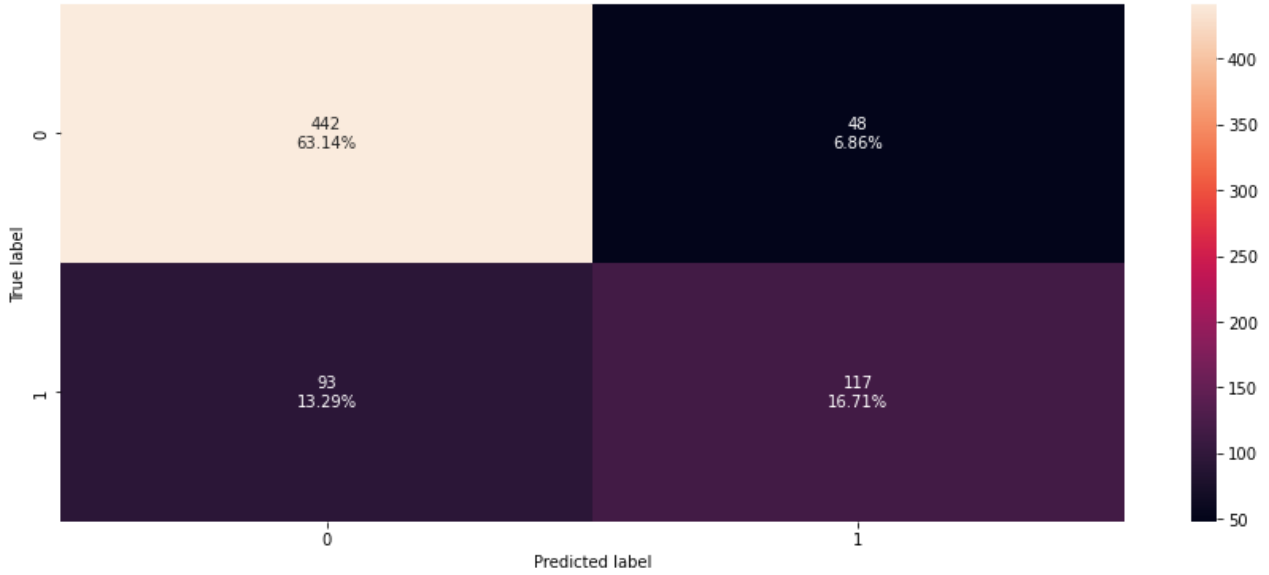
Out[280…

```
AdaBoostClassifier
AdaBoostClassifier(random_state=1)
```

Checking model performance on training set

In [281…

```
confusion_matrix_sklearn(
    ab_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [282…

```
ab_classifier_model_train_perf = model_performance_classification_sklearn(
    ab_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
ab_classifier_model_train_perf
```

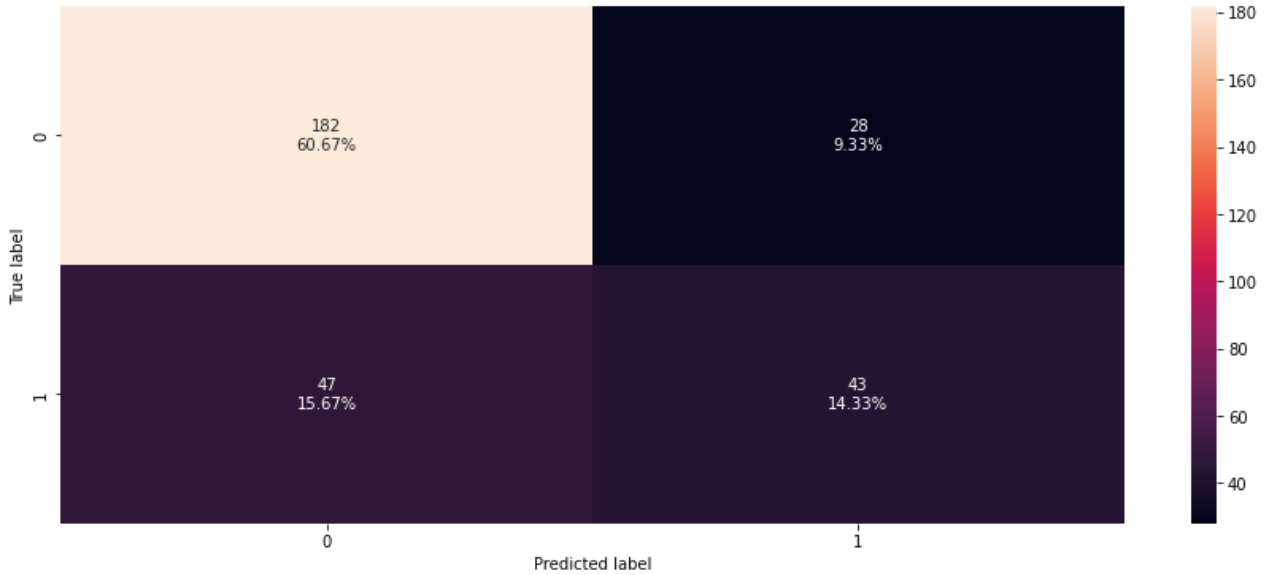
Out[282…

	Accuracy	Recall	Precision	F1
0	0.80	0.56	0.71	0.62

Checking model performance on test set

In [283…

```
confusion_matrix_sklearn(
    ab_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [284…

```
ab_classifier_model_test_perf = model_performance_classification_sklearn(
```

```
    ab_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
ab_classifier_model_test_perf
```

Out[284…

	Accuracy	Recall	Precision	F1
0	0.75	0.48	0.61	0.53

Hyperparameter Tuning - AdaBoost Classifier

- An AdaBoost classifier is a meta-estimator that begins by fitting a classifier on the original dataset and then fits additional copies of the classifier on the same dataset but where the weights of incorrectly classified instances are adjusted such that subsequent classifiers focus more on difficult cases.
- Some important hyperparamters are:
  - base_estimator: The base estimator from which the boosted ensemble is built. By default the base estimator is a decision tree with max_depth=1
  - n_estimators: The maximum number of estimators at which boosting is terminated. Default value is 50.
  - learning_rate: Learning rate shrinks the contribution of each classifier by learning_rate. There is a trade-off between learning_rate and n_estimators.

In [285…

```
# Choose the type of classifier.
abc_tuned = AdaBoostClassifier(random_state=1)

# Grid of parameters to choose from
parameters = {
    # Let's try different max_depth for base_estimator
    "base_estimator": [
        DecisionTreeClassifier(max_depth=1, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=2, class_weight="balanced", random_state=1),
        DecisionTreeClassifier(max_depth=3, class_weight="balanced", random_state=1),
    ],
    "n_estimators": np.arange(60, 100, 10),
    "learning_rate": np.arange(0.1, 0.4, 0.1),
}

# Type of scoring used to compare parameter  combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    abc_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
abc_tuned = grid_obj.best_estimator_
```

```
# Fit the best algorithm to the data.
abc_tuned.fit(X_train, Y_train)
```

Out[285…

AdaBoostClassifier

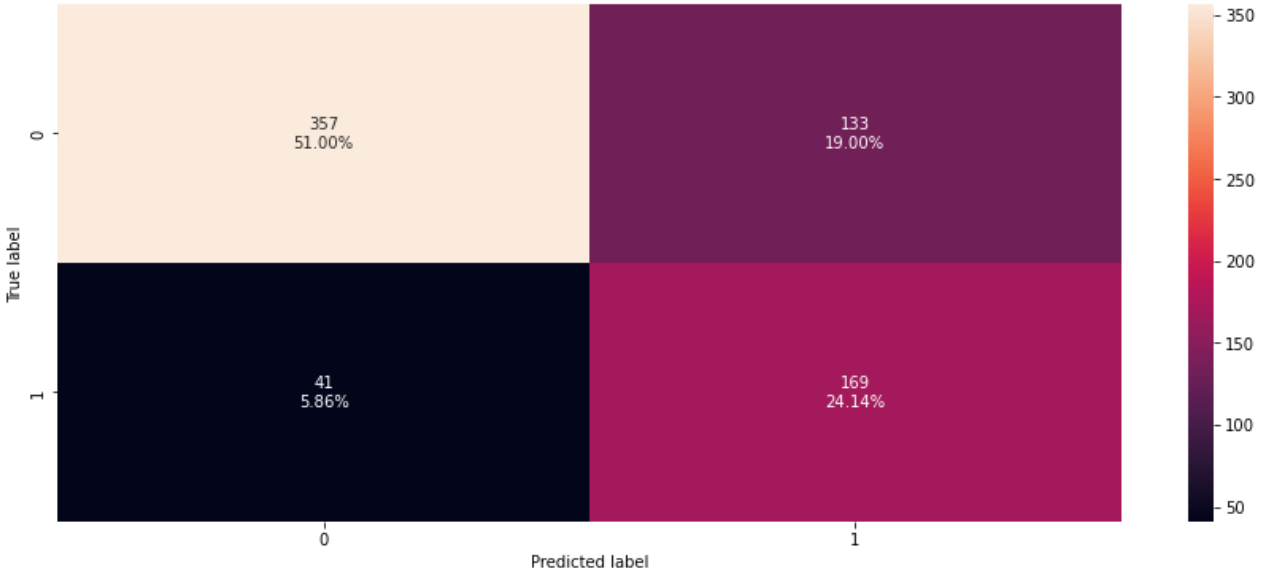
base_estimator: DecisionTreeClassifier

DecisionTreeClassifier

Checking model performance on training set

In [286…

```
confusion_matrix_sklearn(
    abc_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [287…

```
abc_tuned_model_train_perf = model_performance_classification_sklearn(
    abc_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
abc_tuned_model_train_perf
```

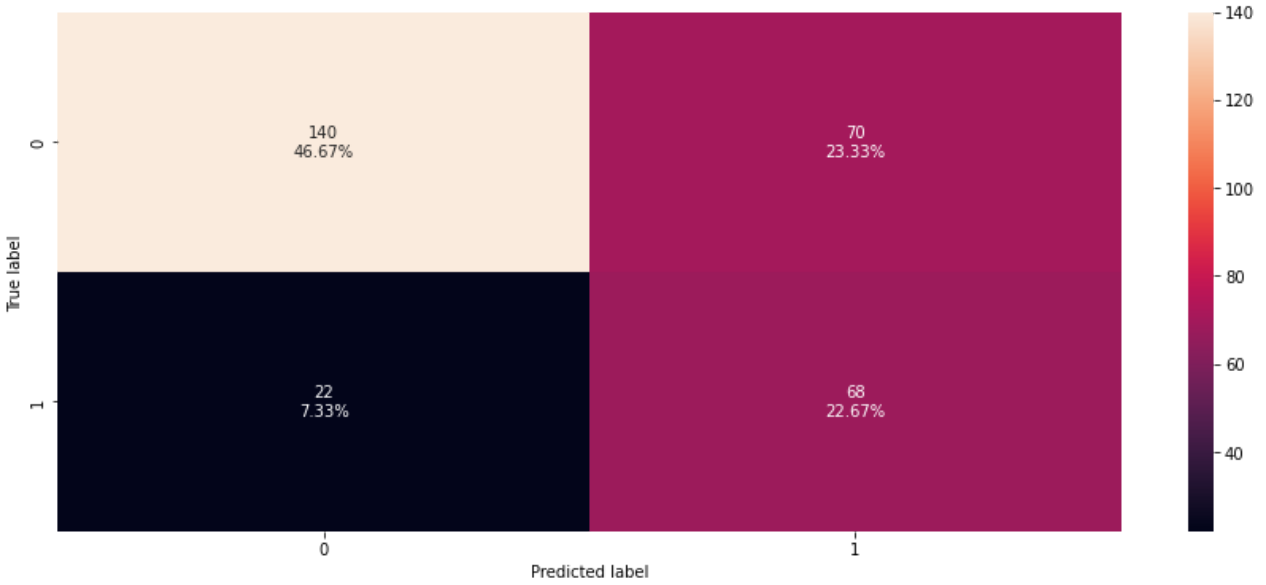
Out[287…

	Accuracy	Recall	Precision	F1
0	0.75	0.80	0.56	0.66

Checking model performance on test set

In [288…

```
confusion_matrix_sklearn(
    abc_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [289…

```
abc_tuned_model_test_perf = model_performance_classification_sklearn(
    abc_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
abc_tuned_model_test_perf
```

Out[289…

	Accuracy	Recall	Precision	F1
0	0.69	0.76	0.49	0.60

Insights

- The model is overfitting the train data as train accuracy is much higher than the test accuracy.
- The model has low test recall. This implies that the model is not good at identifying defaulters.

Important Features for Predicting Loan Default - AdaBoost Classifier

In [290…

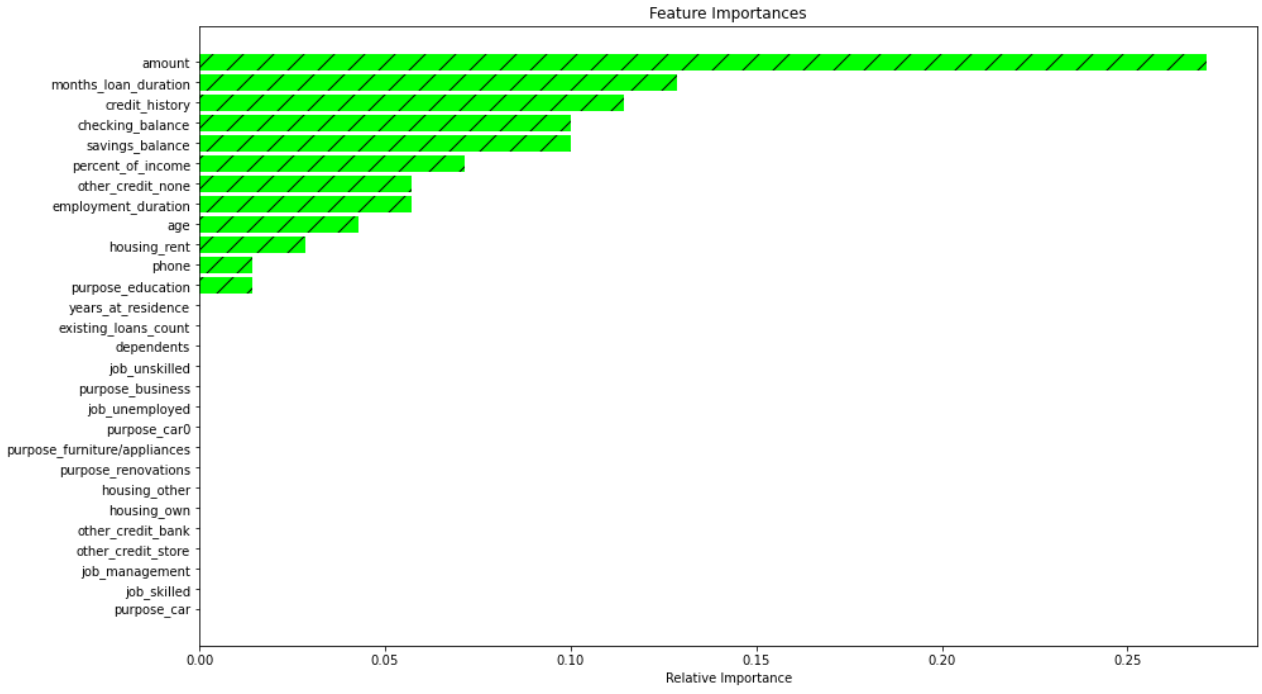
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        abc_tuned.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                            Imp
amount                     0.27
months_loan_duration       0.13
credit_history             0.11
checking_balance           0.10
savings_balance            0.10
percent_of_income          0.07
other_credit_none          0.06
```

```
employment_duration           0.06
age                           0.04
housing_rent                  0.03
phone                         0.01
purpose_education             0.01
job_management                0.00
other_credit_store            0.00
housing_own                   0.00
job_unemployed                0.00
other_credit_bank             0.00
job_skilled                   0.00
purpose_car0                  0.00
housing_other                 0.00
purpose_renovations           0.00
purpose_furniture/appliances  0.00
purpose_car                   0.00
purpose_business              0.00
dependents                    0.00
existing_loans_count          0.00
years_at_residence            0.00
job_unskilled                 0.00
```

In [291…

```
importances = abc_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="lime", align="center", hatch="/"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Gradient Boosting Classifier

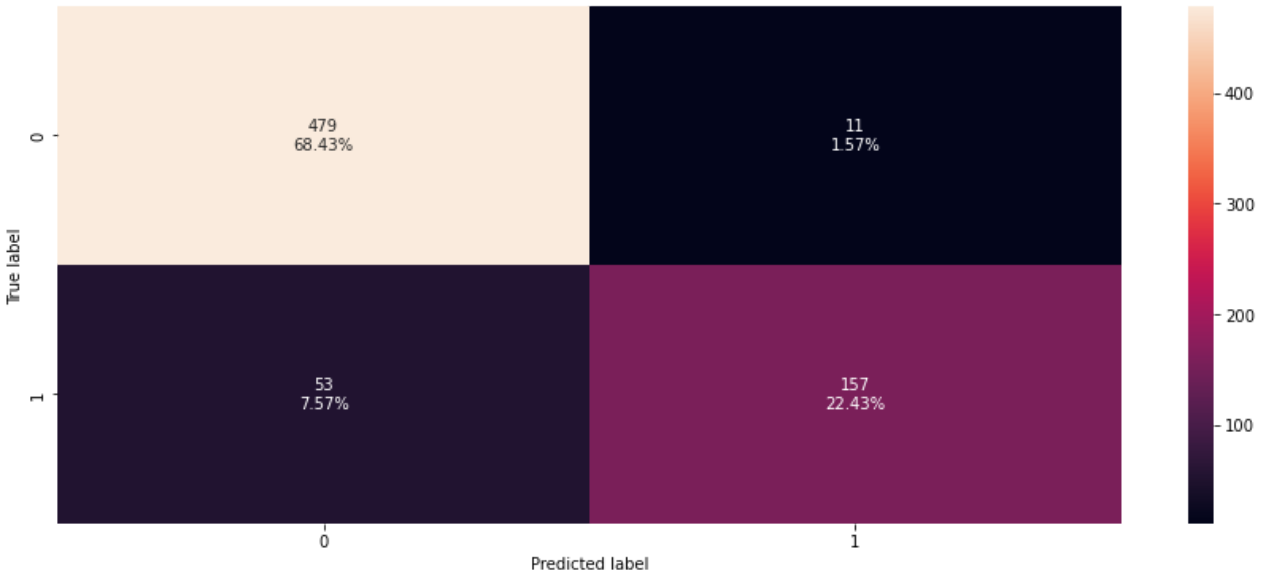
- Most of the hyperparameters available are same as random forest classifier.
- init: An estimator object that is used to compute the initial predictions. If 'zero', the initial raw predictions are set to zero. By default, a DummyEstimator predicting the classes priors is used.
- There is no class_weights parameter in gradient boosting.

In [292...
```
gb_classifier = GradientBoostingClassifier(
    random_state=1
)  ## Complete the code to define Gradient Boosting Classifier with random state = 1
gb_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit Gradient Boosting Classifier on the train data
```

Out[292...
```
GradientBoostingClassifier
GradientBoostingClassifier(random_state=1)
```

Checking model performance on training set

In [293...
```
confusion_matrix_sklearn(
    gb_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [294...
```
gb_classifier_model_train_perf = model_performance_classification_sklearn(
    gb_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
gb_classifier_model_train_perf
```

Out[294...

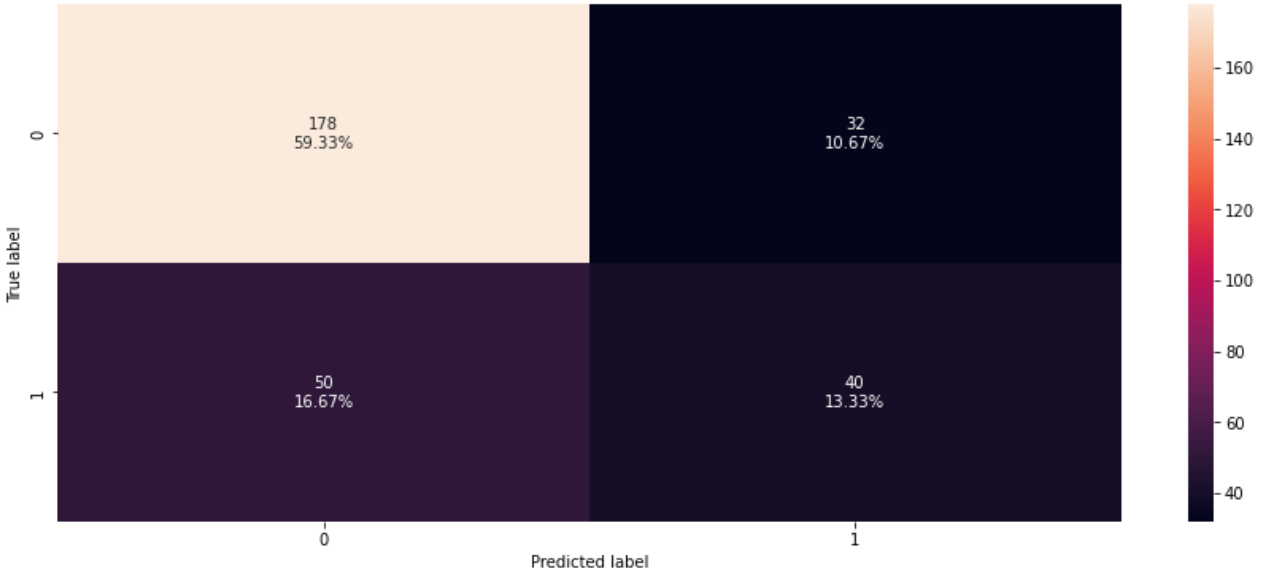
	Accuracy	Recall	Precision	F1

	Accuracy	Recall	Precision	F1
0	0.91	0.75	0.93	0.83

Checking model performance on test set

In [295…

```
confusion_matrix_sklearn(
    gb_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [296…

```
gb_classifier_model_test_perf = model_performance_classification_sklearn(
    gb_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
gb_classifier_model_test_perf
```

Out[296…

	Accuracy	Recall	Precision	F1
0	0.73	0.44	0.56	0.49

- No significant increase in the model performance

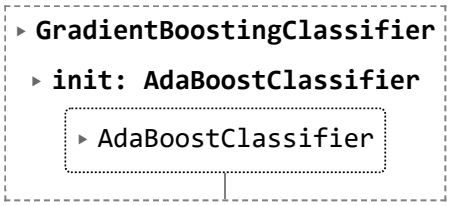
Hyperparameter Tuning - Gradient Boosting Classifier

In [297…

```
# Choose the type of classifier.
gbc_tuned = GradientBoostingClassifier(init=AdaBoostClassifier(random_state=1))

# Grid of parameters to choose from
parameters = {
    "n_estimators": [200, 250, 300],
    "subsample": [0.8, 0.9, 1],
    "max_features": [0.7, 0.8, 0.9, 1],
```

```
    "learning_rate": np.arange(0.1, 0.4, 0.1),
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    gbc_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
gbc_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
gbc_tuned.fit(X_train, Y_train)
```

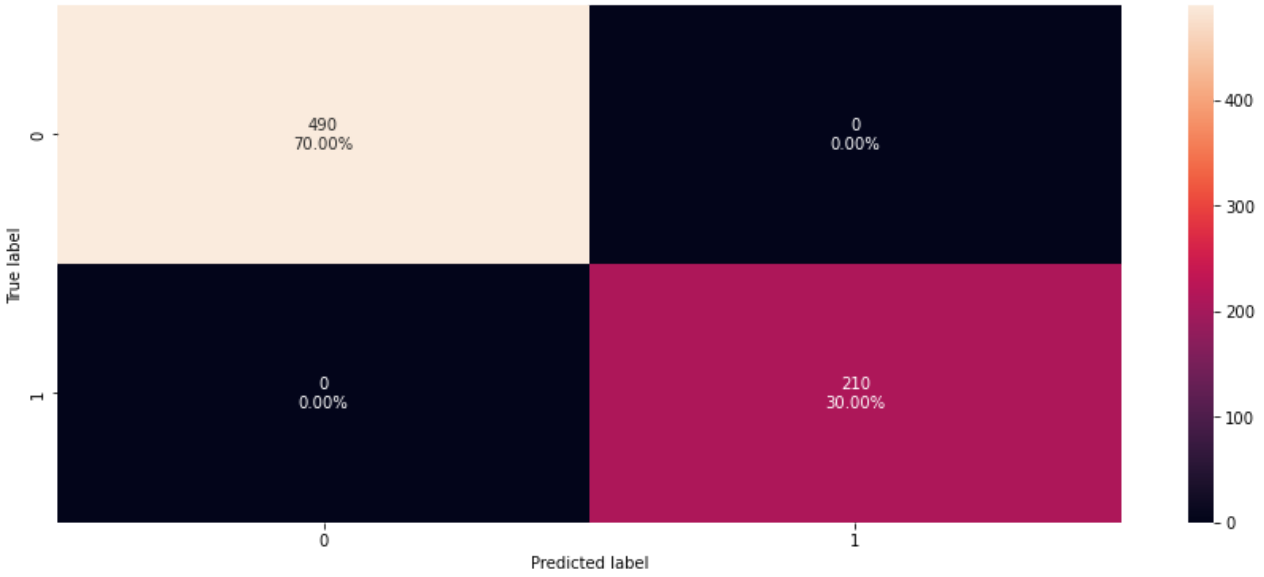
Out[297…



Checking model performance on training set

In [298…

```
confusion_matrix_sklearn(
    gbc_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [299…

```
gbc_tuned_model_train_perf = model_performance_classification_sklearn(
    gbc_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
gbc_tuned_model_train_perf
```

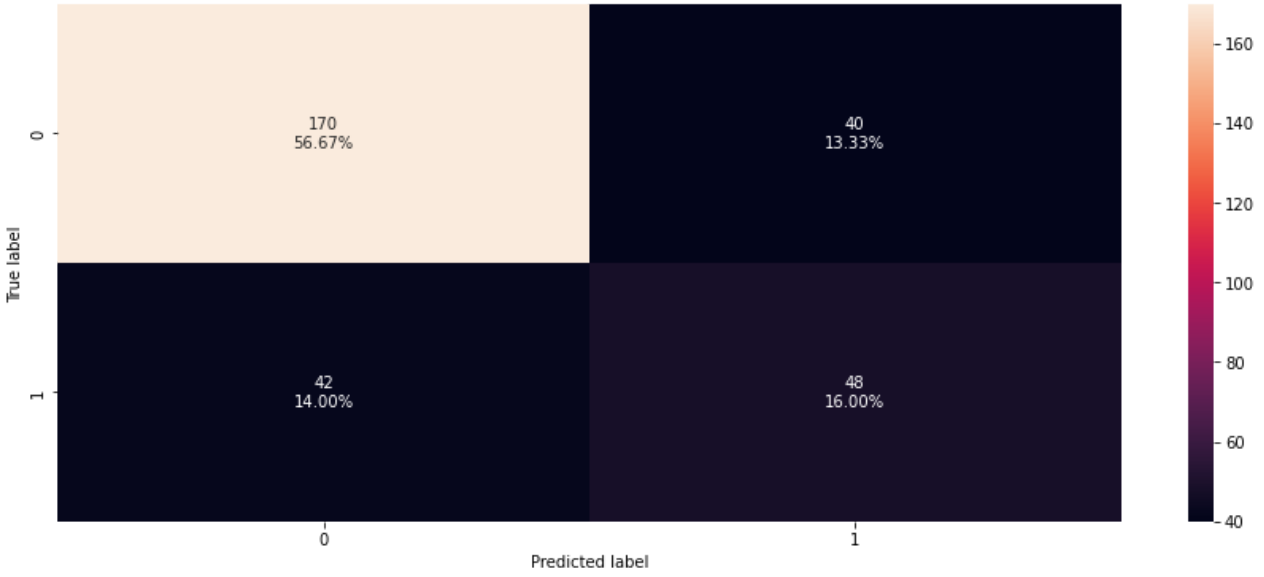
Out[299…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Checking model performance on test set

In [300…

```
confusion_matrix_sklearn(
    gbc_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [301…

```
gbc_tuned_model_test_perf = model_performance_classification_sklearn(
    gbc_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
gbc_tuned_model_test_perf
```

Out[301…

	Accuracy	Recall	Precision	F1
0	0.73	0.53	0.55	0.54

Performance of Gradient Boster remains the same after hyperparameter tuning

Insights

- The model performace has not increased by much.
- The model has started to overfit the train data in terms of recall.
- It is better at identifying non-defaulters than identifying defaulters which is the opposite of the result we need.

Important Features for Predicting Loan Default - Gradient Boost Classifier

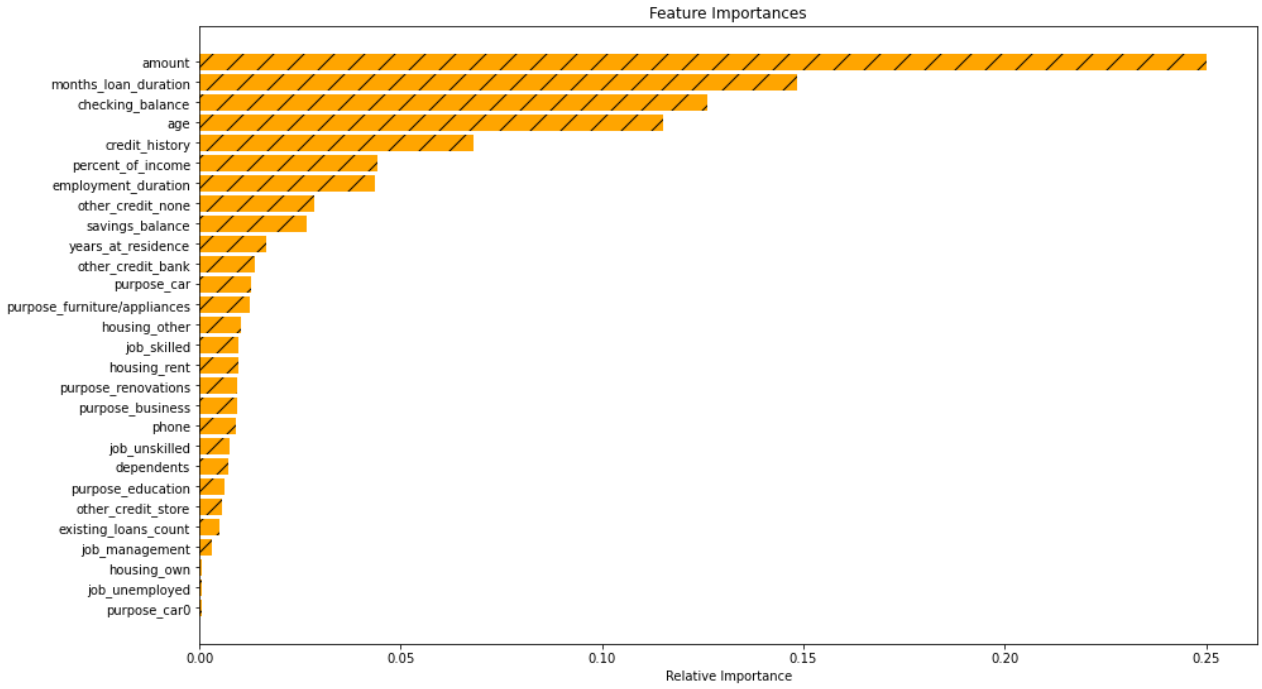
In [302…

```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        gbc_tuned.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

	Imp
amount	0.25
months_loan_duration	0.15
checking_balance	0.13
age	0.12
credit_history	0.07
percent_of_income	0.04
employment_duration	0.04
other_credit_none	0.03
savings_balance	0.03
years_at_residence	0.02
other_credit_bank	0.01
purpose_car	0.01
purpose_furniture/appliances	0.01
housing_other	0.01
job_skilled	0.01
housing_rent	0.01
purpose_renovations	0.01
purpose_business	0.01
phone	0.01
job_unskilled	0.01
dependents	0.01
purpose_education	0.01
other_credit_store	0.01
existing_loans_count	0.01
job_management	0.00
housing_own	0.00
job_unemployed	0.00
purpose_car0	0.00

In [303…

```
importances = gbc_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="orange", align="center", hatch="/
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



- Amount is the most important feature, followed by loan duration and checking balance, as per the tuned gradient boosting model

---

XGBoost Classifier

XGBoost has many hyper parameters which can be tuned to increase the model performance. Some of the important parameters are:

- scale_pos_weight:Control the balance of positive and negative weights, useful for unbalanced classes. It has range from 0 to ∞.
- subsample: Corresponds to the fraction of observations (the rows) to subsample at each step. By default it is set to 1 meaning that we use all rows.
- colsample_bytree: Corresponds to the fraction of features (the columns) to use.
- colsample_bylevel: The subsample ratio of columns for each level. Columns are subsampled from the set of columns chosen for the current tree.
- colsample_bynode: The subsample ratio of columns for each node (split). Columns are subsampled from the set of columns chosen for the current level.
- max_depth: is the maximum number of nodes allowed from the root to the farthest leaf of a tree.
- learning_rate/eta: Makes the model more robust by shrinking the weights on each step.

- gamma: A node is split only when the resulting split gives a positive reduction in the loss function. Gamma specifies the minimum loss reduction required to make a split.

In [304…

```
xgb_classifier = XGBClassifier(
    random_state=1
)  ## Complete the code to define XGBoost Classifier with random state = 1 and eval_met
xgb_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit XGBoost Classifier on the train data
```

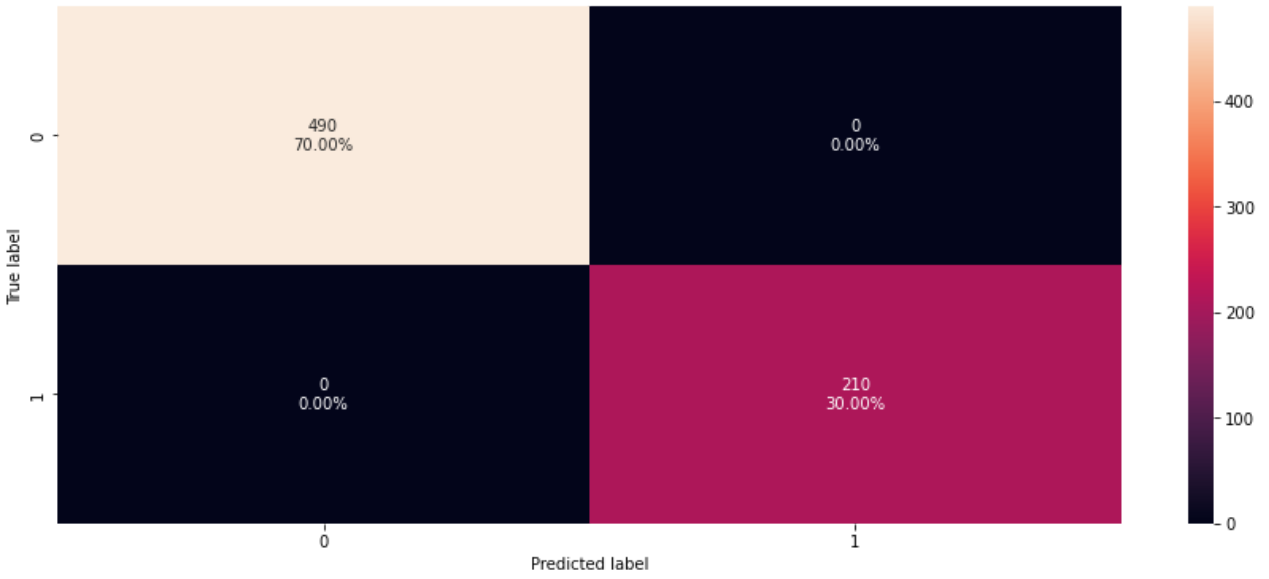
Out[304…

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwise',
              importance_type=None, interaction_constraints='',
              learning_rate=0.300000012, max_bin=256, max_cat_to_onehot=4,
              max_delta_step=0, max_depth=6, max_leaves=0, min_child_weight=1,
              missing=nan, monotone_constraints='()', n_estimators=100,
              n_jobs=0, num_parallel_tree=1, predictor='auto', random_state=1,
              reg_alpha=0, reg_lambda=1, ...)
```

Checking model performance on training set

In [305…

```
confusion_matrix_sklearn(
    xgb_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```



In [306…

```
xgb_classifier_model_train_perf = model_performance_classification_sklearn(
    xgb_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
xgb_classifier_model_train_perf
```

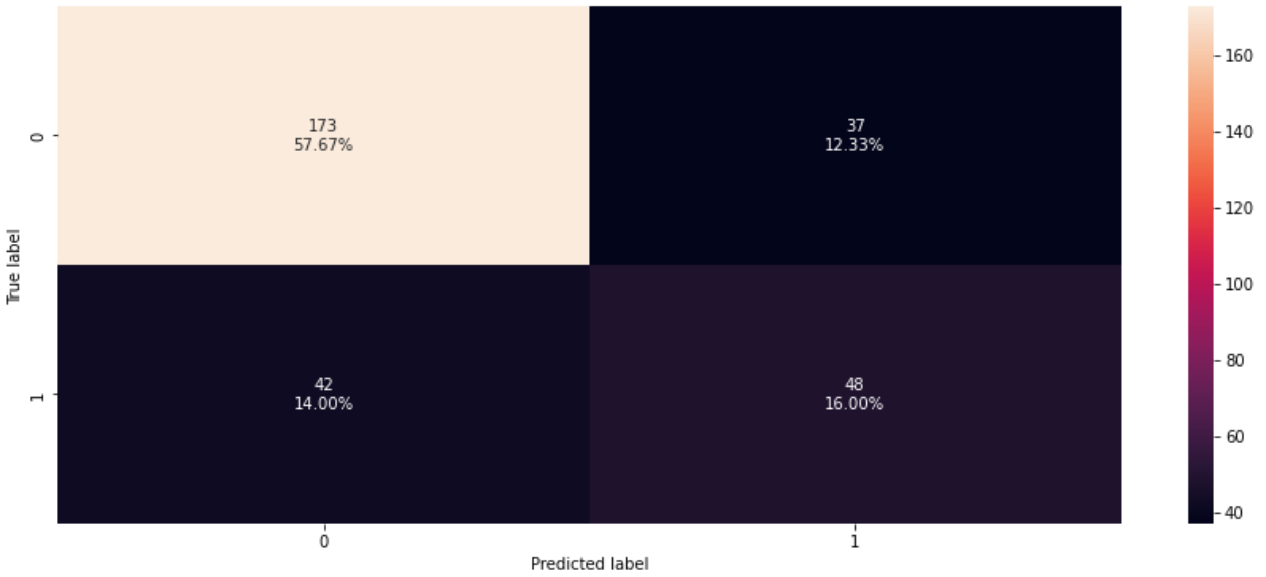
Out[306…

	Accuracy	Recall	Precision	F1
0	1.00	1.00	1.00	1.00

Checking model performance on test set

In [307…

```
confusion_matrix_sklearn(
    xgb_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [308…

```
xgb_classifier_model_test_perf = model_performance_classification_sklearn(
    xgb_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
xgb_classifier_model_test_perf
```

Out[308…

	Accuracy	Recall	Precision	F1
0	0.74	0.53	0.56	0.55

- xgb_classifier model is slightly overfitting
- Peformance is only slightly lower than hyperparameter tuned Gradient Boosting Classifier

---

Hyperparameter Tuning - XGBoost Classifier

In [309…

```
# Choose the type of classifier.
xgb_tuned = XGBClassifier(random_state=1, eval_metric="logloss")

# Grid of parameters to choose from
parameters = {
```

```
    "n_estimators": np.arange(150, 250, 50),
    "scale_pos_weight": [1, 2],
    "subsample": [0.7, 0.9, 1],
    "learning_rate": np.arange(0.1, 0.4, 0.1),
    "gamma": [1, 3, 5],
    "colsample_bytree": [0.7, 0.8, 0.9],
    "colsample_bylevel": [0.8, 0.9, 1],
}

# Type of scoring used to compare parameter combinations
acc_scorer = metrics.make_scorer(metrics.f1_score)

# Run the grid search
grid_obj = GridSearchCV(
    xgb_tuned, parameters, scoring=scorer, cv=5
)  ## Complete the code to run grid search with cv = 5
grid_obj = grid_obj.fit(
    X_train, Y_train
)  ## Complete the code to fit the grid_obj on train data

# Set the clf to the best combination of parameters
xgb_tuned = grid_obj.best_estimator_

# Fit the best algorithm to the data.
xgb_tuned.fit(X_train, Y_train)
```

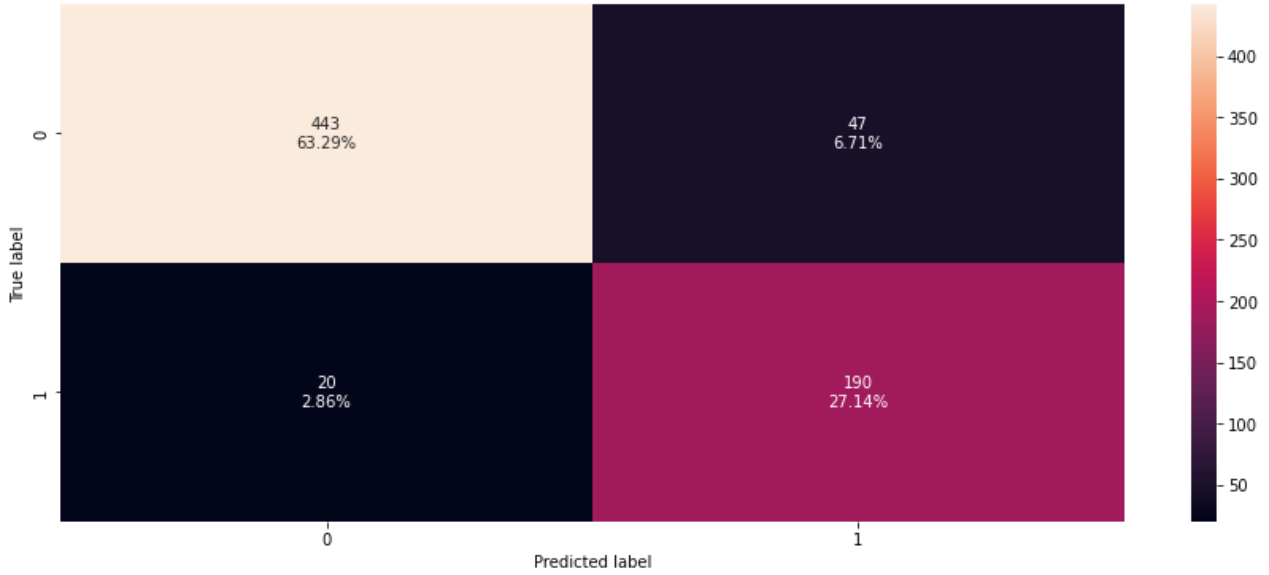
Out[309…

```
XGBClassifier(base_score=0.5, booster='gbtree', callbacks=None,
              colsample_bylevel=0.9, colsample_bynode=1, colsample_bytree=0.
7,
              early_stopping_rounds=None, enable_categorical=False,
              eval_metric='logloss', gamma=5, gpu_id=-1,
              grow_policy='depthwise', importance_type=None,
              interaction_constraints='', learning_rate=0.1, max_bin=256,
              max_cat_to_onehot=4, max_delta_step=0, max_depth=6, max_leaves
=0,
              min_child_weight=1, missing=nan, monotone_constraints='()',
              n_estimators=150, n_jobs=0, num_parallel_tree=1, predictor='au
```

Checking model performance on training set

In [310…

```
confusion_matrix_sklearn(
    xgb_tuned, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data on tuned estimator
```



In [311…

```
xgb_tuned_model_train_perf = model_performance_classification_sklearn(
    xgb_tuned, X_train, Y_train
)  ## Complete the code to check performance for train data on tuned estimator
xgb_tuned_model_train_perf
```

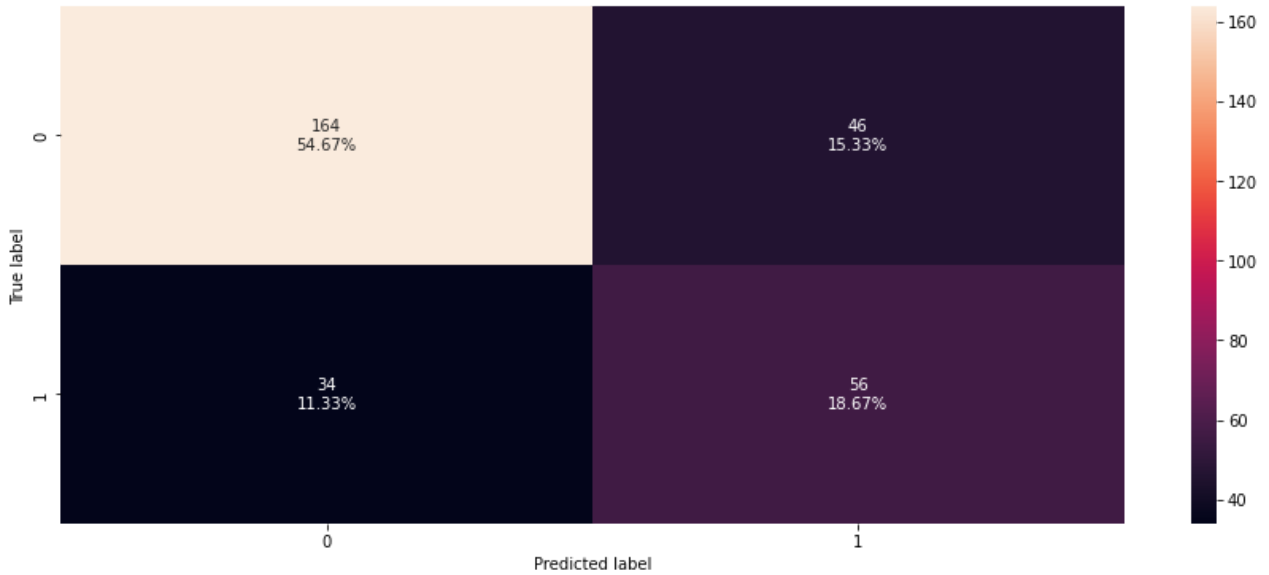
Out[311…

	Accuracy	Recall	Precision	F1
0	0.90	0.90	0.80	0.85

Checking model performance on test set

In [312…

```
confusion_matrix_sklearn(
    xgb_tuned, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data on tuned estimator
```



In [313…

```
xgb_tuned_model_test_perf = model_performance_classification_sklearn(
```

```
    xgb_tuned, X_test, Y_test
)  ## Complete the code to check performance for test data on tuned estimator
xgb_tuned_model_test_perf
```

Out[313…

	Accuracy	Recall	Precision	F1
0	0.73	0.62	0.55	0.58

- Hypeparameter tuning of XG model reduces overfitting
- The model performance for hypertuned XGBoost is lower than XGBoost

Insights

- The test accuracy of the model has reduced as compared to the model with default parameters but the recall has increased significantly and the model is able to identify most of the defaulters.
- Decreasing number of false negatives has increased the number of false positives here.
- The tuned model is not overfitting and generalizes well.

Important Features for Prediction Loan Default - XGBoost Classifier

In [314…

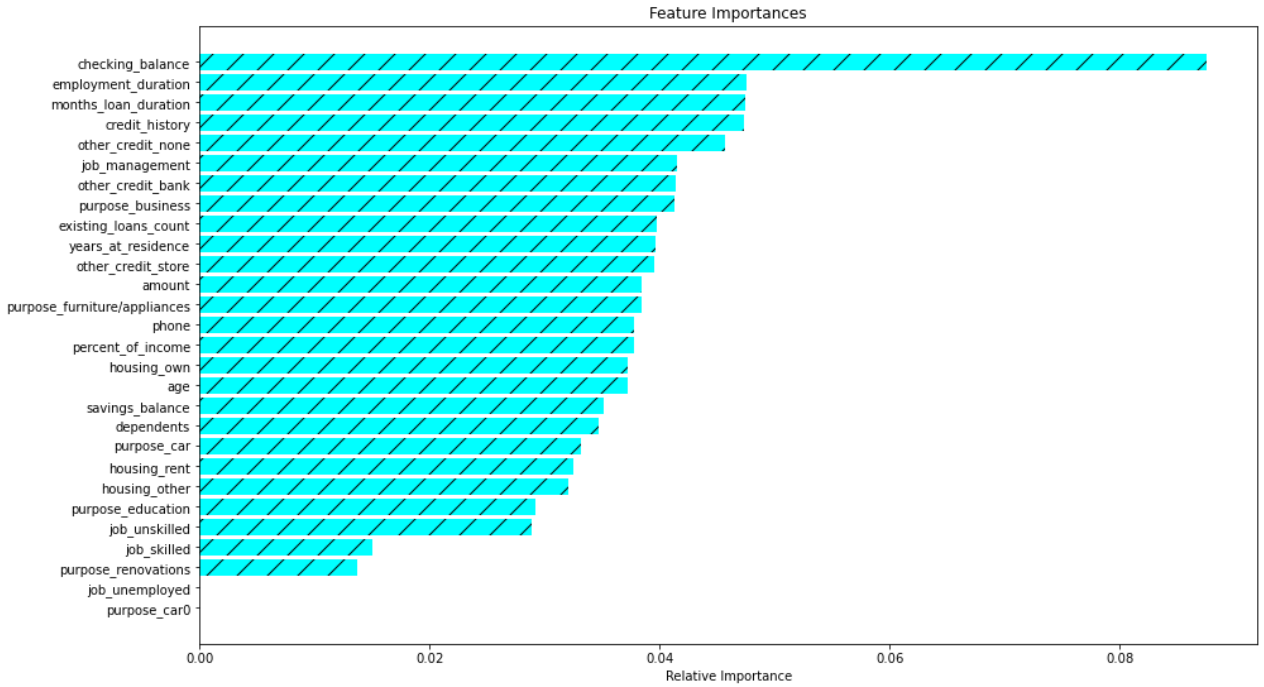
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        xgb_tuned.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                              Imp
checking_balance             0.09
employment_duration          0.05
months_loan_duration         0.05
credit_history               0.05
other_credit_none            0.05
job_management               0.04
other_credit_bank            0.04
purpose_business             0.04
existing_loans_count         0.04
years_at_residence           0.04
other_credit_store           0.04
amount                       0.04
purpose_furniture/appliances 0.04
phone                        0.04
percent_of_income            0.04
housing_own                  0.04
age                          0.04
savings_balance              0.04
dependents                   0.03
purpose_car                  0.03
```

```
housing_rent                0.03
housing_other               0.03
purpose_education           0.03
job_unskilled               0.03
job_skilled                 0.02
purpose_renovations         0.01
job_unemployed              0.00
purpose_car0                0.00
```

In [315…

```
importances = xgb_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="aqua", align="center", hatch="/"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



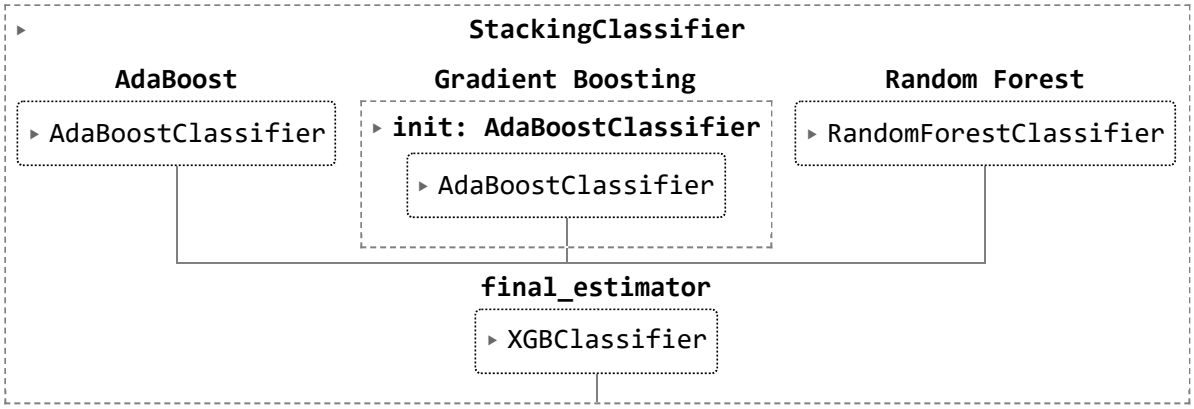
Stacking Classifier

In [316…

```
estimators = [
    ("AdaBoost", ab_classifier),
    ("Gradient Boosting", gbc_tuned),
    ("Random Forest", rf_tuned),
]

final_estimator = xgb_tuned
```

```
stacking_classifier = StackingClassifier(
    estimators=estimators, final_estimator=final_estimator, cv=5
)  ## Complete the code to define Stacking Classifier

stacking_classifier.fit(
    X_train, Y_train
)  ## Complete the code to fit Stacking Classifier on the train data
```

Out[316...



Checking model performance on training set

In [317...

```
confusion_matrix_sklearn(
    stacking_classifier, X_train, Y_train
)  ## Complete the code to create confusion matrix for train data
```

In [318...

```
stacking_classifier_model_train_perf = model_performance_classification_sklearn(
    stacking_classifier, X_train, Y_train
)  ## Complete the code to check performance on train data
stacking_classifier_model_train_perf
```

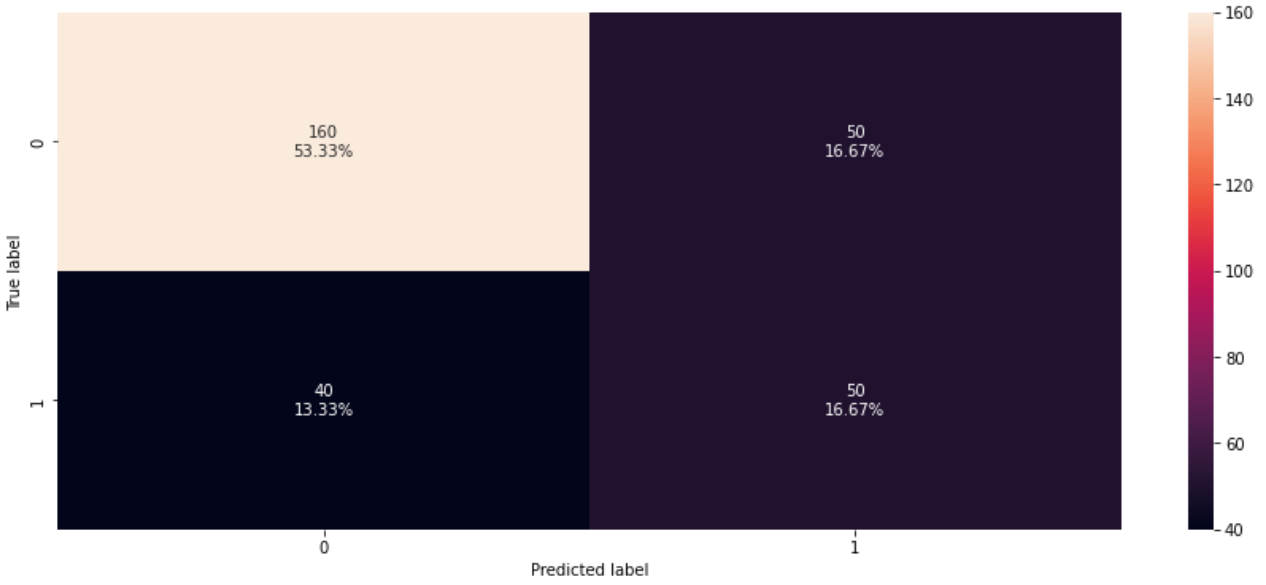
Out[318...

	Accuracy	Recall	Precision	F1
0	0.95	0.89	0.94	0.92

Checking model performance on test set

In [319…

```
confusion_matrix_sklearn(
    stacking_classifier, X_test, Y_test
)  ## Complete the code to create confusion matrix for test data
```



In [320…

```
stacking_classifier_model_test_perf = model_performance_classification_sklearn(
    stacking_classifier, X_test, Y_test
)  ## Complete the code to check performance for test data
stacking_classifier_model_test_perf
```

Out[320…

	Accuracy	Recall	Precision	F1
0	0.70	0.56	0.50	0.53

- Model performance is similar to hypertuned XG Boost

Summary Performance Comparison Measures of Boosting Train vs Test

In [321…

```
boosting_comp_df = pd.concat(
    [
        ab_classifier_model_train_perf.T,
        ab_classifier_model_test_perf.T,
        gb_classifier_model_train_perf.T,
        gb_classifier_model_test_perf.T,
        xgb_classifier_model_train_perf.T,
        xgb_classifier_model_test_perf.T,
    ],
    axis=1,
)
```

```
boosting_comp_df.columns = [
    "Adaboost Classifier (train)",
    "Adabosst Classifier (test)",
    "Gradient Boost Classifier (train)",
    "Gradient Boost Classifier (test)",
    "XGBoost Classifier (train)",
    "XGBoost Classifier (test)",
]
print("Boosting performance comparison:")
boosting_comp_df
```

```
Boosting performance comparison:
```

Out[321…

	Adaboost Classifier (train)	Adabosst Classifier (test)	Gradient Boost Classifier (train)	Gradient Boost Classifier (test)	XGBoost Classifier (train)	XGBoost Classifier (test)
Accuracy	0.80	0.75	0.91	0.73	1.00	0.74
Recall	0.56	0.48	0.75	0.44	1.00	0.53
Precision	0.71	0.61	0.93	0.56	1.00	0.56
F1	0.62	0.53	0.83	0.49	1.00	0.55

Summary Performance Comparison Measures of Boosting Tuned Training vs Tuned Testing

In [322…

```
boosting_tuned_comp_df = pd.concat(
    [
        abc_tuned_model_train_perf.T,
        abc_tuned_model_test_perf.T,
        gbc_tuned_model_train_perf.T,
        gbc_tuned_model_test_perf.T,
        xgb_tuned_model_train_perf.T,
        xgb_tuned_model_test_perf.T,
        stacking_classifier_model_train_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
boosting_tuned_comp_df.columns = [
    "Adaboost Classifier tuned (train)",
    "Adabosst Classifier tuned (test)",
    "Gradient Boost Classifier tuned (train)",
    "Gradient Boost Classifier tuned (test)",
    "XGBoost Classifier tuned (train)",
    "XGBoost Classifier tuned (test)",
    "Stacking Classifier (train)",
    "Stacking Classifier (test)",
]
print("Boosting Tuned performance comparison:")
boosting_tuned_comp_df
```

```
Boosting Tuned performance comparison:
```

Out[322…

	Adaboost Classifier tuned (train)	Adabosst Classifier tuned (test)	Gradient Boost Classifier tuned (train)	Gradient Boost Classifier tuned (test)	XGBoost Classifier tuned (train)	XGBoost Classifier tuned (test)	Stacking Classifier (train)	Stacking Classifier (test)
Accuracy	0.75	0.69	1.00	0.73	0.90	0.73	0.95	0.70
Recall	0.80	0.76	1.00	0.53	0.90	0.62	0.89	0.56
Precision	0.56	0.49	1.00	0.55	0.80	0.55	0.94	0.50
F1	0.66	0.60	1.00	0.54	0.85	0.58	0.92	0.53

Summary Performance Measures of all Bagging and Boosting: All Training Models

In [323…

```
# training performance comparison

models_train_comp_df = pd.concat(
    [
        decision_tree_perf_train.T,
        dtree_estimator_model_train_perf.T,
        rf_estimator_model_train_perf.T,
        rf_tuned_model_train_perf.T,
        bagging_classifier_model_train_perf.T,
        bagging_estimator_tuned_model_train_perf.T,
        ab_classifier_model_train_perf.T,
        abc_tuned_model_train_perf.T,
        gb_classifier_model_train_perf.T,
        gbc_tuned_model_train_perf.T,
        xgb_classifier_model_train_perf.T,
        xgb_tuned_model_train_perf.T,
        stacking_classifier_model_train_perf.T,
    ],
    axis=1,
)
models_train_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest ",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Classifier Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier",
]
print("Training performance comparison:")
models_train_comp_df
```

Training performance comparison:

Out[323…

	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Classifier Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	1.00	0.30	1.00	0.95	0.98	1.00	0.80	0.75	0.91
Recall	1.00	1.00	1.00	0.84	0.94	1.00	0.56	0.80	0.75
Precision	1.00	0.30	1.00	0.98	0.99	1.00	0.71	0.56	0.93
F1	1.00	0.46	1.00	0.91	0.97	1.00	0.62	0.66	0.83

Summary Performance Measures of all Bagging and Boosting: All Testing Models

In [324…

```
# testing performance comparison

models_test_comp_df = pd.concat(
    [
        decision_tree_perf_test.T,
        dtree_estimator_model_test_perf.T,
        rf_estimator_model_test_perf.T,
        rf_tuned_model_test_perf.T,
        bagging_classifier_model_test_perf.T,
        bagging_estimator_tuned_model_test_perf.T,
        ab_classifier_model_test_perf.T,
        abc_tuned_model_test_perf.T,
        gb_classifier_model_test_perf.T,
        gbc_tuned_model_test_perf.T,
        xgb_classifier_model_test_perf.T,
        xgb_tuned_model_test_perf.T,
        stacking_classifier_model_test_perf.T,
    ],
    axis=1,
)
models_test_comp_df.columns = [
    "Decision Tree",
    "Decision Tree Tuned",
    "Random Forest",
    "Random Forest Tuned",
    "Bagging Classifier",
    "Bagging Classifier Tuned",
    "Adaboost Classifier",
    "Adabosst Classifier Tuned",
    "Gradient Boost Classifier",
    "Gradient Boost Classifier Tuned",
    "XGBoost Classifier",
    "XGBoost Classifier Tuned",
    "Stacking Classifier",
]
print("Testing performance comparison:")
models_test_comp_df
```

Out[324...

Testing performance comparison:

	Decision Tree	Decision Tree Tuned	Random Forest	Random Forest Tuned	Bagging Classifier	Bagging Classifier Tuned	Adaboost Classifier	Adabosst Classifier Tuned	Gradient Boost Classifier
Accuracy	0.69	0.30	0.75	0.71	0.74	0.75	0.75	0.69	0.73
Recall	0.47	1.00	0.42	0.31	0.48	0.47	0.48	0.76	0.44
Precision	0.48	0.30	0.63	0.53	0.59	0.61	0.61	0.49	0.56
F1	0.47	0.46	0.51	0.39	0.53	0.53	0.53	0.60	0.49

Observations:

- Accuracy appears to be similar in all Bagging and Boosting models and lower in Decision Tree models.
- Recall seems to be the highest in Tuned Decision Tree Classifier (1.00), followed by Tuned AdaBoost Classifier
- Precision looks similar in all Bagging and Boosting models and slightly lower in Tuned Decision Tree model.
- F1 coefficient appears to be higher in Boosting models than Bagging models.
- Interestingly, there is virtually no overfitting in AdaBoost and Gradient Boost models (both default and tuned). AdaBoost is very slightly overfitting than Gradient Boost models.

Important features of the final selected model

Selected Model - Tunned Random Forest

Important Features for Predicting Loan Default - Tuned AdaBoost Classifier

In [325...

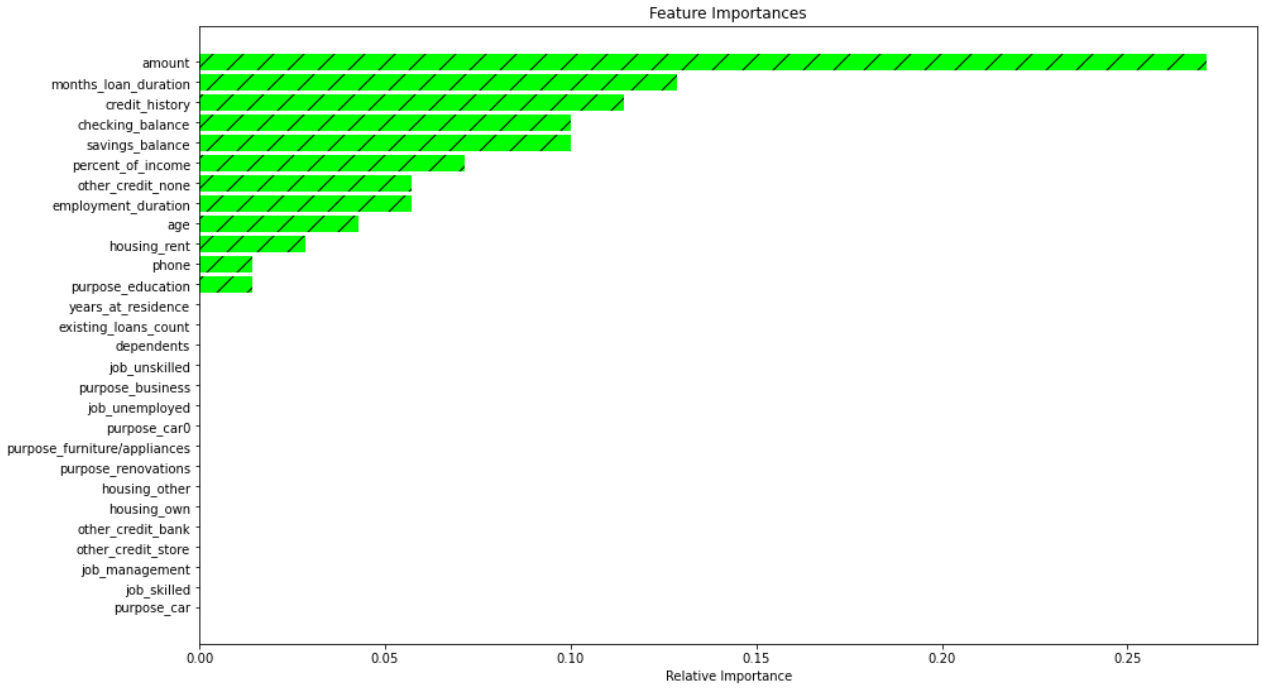
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        abc_tuned.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                          Imp
amount                   0.27
months_loan_duration     0.13
```

```
credit_history               0.11
checking_balance             0.10
savings_balance              0.10
percent_of_income            0.07
other_credit_none            0.06
employment_duration          0.06
age                          0.04
housing_rent                 0.03
phone                        0.01
purpose_education            0.01
job_management               0.00
other_credit_store           0.00
housing_own                  0.00
job_unemployed               0.00
other_credit_bank            0.00
job_skilled                  0.00
purpose_car0                 0.00
housing_other                0.00
purpose_renovations          0.00
purpose_furniture/appliances 0.00
purpose_car                  0.00
purpose_business             0.00
dependents                   0.00
existing_loans_count         0.00
years_at_residence           0.00
job_unskilled                0.00
```

In [326…

```
importances = abc_tuned.feature_importances_
indices = np.argsort(importances)
feature_names = list(X.columns)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="lime", align="center", hatch="/"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Second Model Selection- Gradient Boost Classifier

In [327…

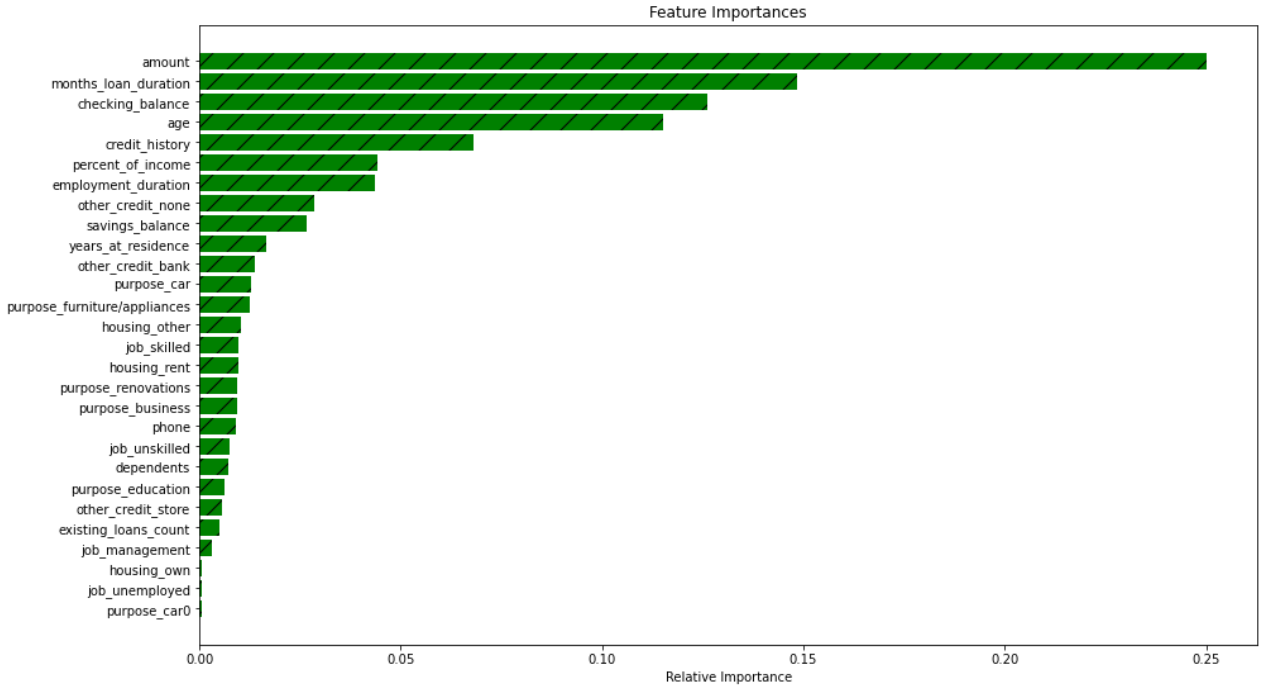
```
# importance of features in the tree building ( The importance of a feature is computed
# (normalized) total reduction of the criterion brought by that feature. It is also kno

print(
    pd.DataFrame(
        gbc_tuned.feature_importances_, columns=["Imp"], index=X_train.columns
    ).sort_values(by="Imp", ascending=False)
)
```

```
                                 Imp
amount                          0.25
months_loan_duration            0.15
checking_balance                0.13
age                             0.12
credit_history                  0.07
percent_of_income               0.04
employment_duration             0.04
other_credit_none               0.03
savings_balance                 0.03
years_at_residence              0.02
other_credit_bank               0.01
purpose_car                     0.01
purpose_furniture/appliances    0.01
housing_other                   0.01
job_skilled                     0.01
housing_rent                    0.01
purpose_renovations             0.01
purpose_business                0.01
phone                           0.01
job_unskilled                   0.01
dependents                      0.01
purpose_education               0.01
other_credit_store              0.01
```

```
existing_loans_count          0.01
job_management                0.00
housing_own                   0.00
job_unemployed                0.00
purpose_car0                  0.00
```

In [328…

```
feature_names = X_train.columns
importances = gbc_tuned.feature_importances_
indices = np.argsort(importances)

plt.figure(figsize=(15, 9))
plt.title("Feature Importances")
plt.barh(
    range(len(indices)), importances[indices], color="green", align="center", hatch="/"
)
plt.yticks(range(len(indices)), [feature_names[i] for i in indices])
plt.xlabel("Relative Importance")
plt.show()
```



Using Decision Tree for Prediction Diabetes Outcome

In [329…

```
# Text report showing the rules of a decision tree -
feature_names = list(X_train.columns)
print(tree.export_text(model, feature_names=feature_names, show_weights=True))
```

```
|--- checking_balance <= 0.00
|   |--- other_credit_none <= 0.50
|   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |--- age <= 43.50
|   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |--- years_at_residence <= 3.00
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
```

```
|   |   |   |   |   |--- years_at_residence >  3.00
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |--- percent_of_income >  1.50
|   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |--- months_loan_duration <= 39.00
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- months_loan_duration >  39.00
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |   |--- savings_balance <= 3.50
|   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 8.00] class: 1
|   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |--- other_credit_bank <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- other_credit_bank >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- savings_balance >  3.50
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |--- age >  43.50
|   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |--- amount <= 1005.00
|   |   |   |   |--- employment_duration <= 4.00
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- employment_duration >  4.00
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |--- amount >  1005.00
|   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |--- phone >  1.50
|   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |--- other_credit_none >  0.50
|   |   |--- age <= 29.50
|   |   |   |--- months_loan_duration <= 34.50
|   |   |   |   |--- age <= 22.50
|   |   |   |   |   |--- percent_of_income <= 2.50
|   |   |   |   |   |   |--- age <= 19.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- age >  19.50
|   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |--- percent_of_income >  2.50
|   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |--- age >  22.50
|   |   |   |   |   |--- amount <= 2226.50
|   |   |   |   |   |   |--- savings_balance <= 2.50
|   |   |   |   |   |   |   |--- amount <= 1922.00
|   |   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [17.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- amount >  1922.00
|   |   |   |   |   |   |   |   |--- percent_of_income <= 3.00
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
```

```
|   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- savings_balance >  2.50
|   |   |   |   |   |   |   |   |   |--- employment_duration <= 4.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |--- employment_duration >  4.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- amount >  2226.50
|   |   |   |   |   |   |   |--- weights: [22.00, 0.00] class: 0
|   |   |   |   |--- months_loan_duration >  34.50
|   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |--- job_skilled <= 0.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- job_skilled >  0.50
|   |   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |--- age >  29.50
|   |   |   |   |--- age <= 66.00
|   |   |   |   |   |--- purpose_renovations <= 0.50
|   |   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |   |--- amount <= 4602.50
|   |   |   |   |   |   |   |   |--- job_unskilled <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [15.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- job_unskilled >  0.50
|   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 16.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- months_loan_duration >  16.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- amount >  4602.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |   |--- purpose_education <= 0.50
|   |   |   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |   |   |--- weights: [98.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |   |   |--- age <= 46.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [18.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- age >  46.00
|   |   |   |   |   |   |   |   |   |   |--- employment_duration <= 4.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- employment_duration >  4.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- purpose_education >  0.50
|   |   |   |   |   |   |   |   |--- job_management <= 0.50
|   |   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- job_management >  0.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- purpose_renovations >  0.50
|   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |--- age >  66.00
|   |   |   |   |--- existing_loans_count <= 1.50
|   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- existing_loans_count >  1.50
|   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|--- checking_balance >  0.00
|   |--- months_loan_duration <= 26.50
```

```
|   |   |--- credit_history <= 3.50
|   |   |   |--- months_loan_duration <= 11.50
|   |   |   |   |--- amount <= 10931.50
|   |   |   |   |   |--- age <= 29.50
|   |   |   |   |   |   |--- amount <= 1013.50
|   |   |   |   |   |   |   |--- job_skilled <= 0.50
|   |   |   |   |   |   |   |   |--- amount <= 727.00
|   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  727.00
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |--- job_skilled >  0.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- amount >  1013.50
|   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |--- age >  29.50
|   |   |   |   |   |   |--- housing_own <= 0.50
|   |   |   |   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |   |   |   |--- existing_loans_count <= 1.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- existing_loans_count >  1.50
|   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- percent_of_income >  1.50
|   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |   |--- housing_own >  0.50
|   |   |   |   |   |   |   |--- weights: [40.00, 0.00] class: 0
|   |   |   |   |--- amount >  10931.50
|   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |--- months_loan_duration >  11.50
|   |   |   |   |--- amount <= 1374.50
|   |   |   |   |   |--- purpose_car <= 0.50
|   |   |   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |--- credit_history <= 2.00
|   |   |   |   |   |   |   |   |   |--- purpose_renovations <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- purpose_renovations >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- credit_history >  2.00
|   |   |   |   |   |   |   |   |   |--- purpose_business <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 10.00] class: 1
|   |   |   |   |   |   |   |   |   |--- purpose_business >  0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |--- employment_duration <= 2.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |--- employment_duration >  2.50
|   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |   |   |--- age <= 32.50
|   |   |   |   |   |   |   |   |--- weights: [10.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- age >  32.50
|   |   |   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [5.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |   |   |   |   |   |   |   |--- credit_history <= 2.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- credit_history >  2.00
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |--- dependents >  1.50
```

```
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |--- purpose_car >  0.50
|   |   |   |   |   |   |   |--- age <= 45.00
|   |   |   |   |   |   |   |   |--- age <= 22.50
|   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- age >  22.50
|   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 20.00] class: 1
|   |   |   |   |   |   |   |--- age >  45.00
|   |   |   |   |   |   |   |   |--- months_loan_duration <= 22.50
|   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- months_loan_duration >  22.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |--- amount >  1374.50
|   |   |   |   |   |   |--- amount <= 8472.00
|   |   |   |   |   |   |   |--- housing_rent <= 0.50
|   |   |   |   |   |   |   |   |--- purpose_furniture/appliances <= 0.50
|   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |   |   |   |   |--- purpose_business <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- purpose_business >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
|   |   |   |   |   |   |   |   |   |   |--- savings_balance <= 3.50
|   |   |   |   |   |   |   |   |   |   |   |--- amount <= 4764.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [33.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- amount >  4764.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- savings_balance >  3.50
|   |   |   |   |   |   |   |   |   |   |   |--- amount <= 1726.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- amount >  1726.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- purpose_furniture/appliances >  0.50
|   |   |   |   |   |   |   |   |   |--- employment_duration <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- age <= 53.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- age >  53.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- employment_duration >  1.50
|   |   |   |   |   |   |   |   |   |   |--- amount <= 1931.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [9.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- amount >  1931.50
|   |   |   |   |   |   |   |   |   |   |   |--- amount <= 2100.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- amount >  2100.50
|   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 8
|   |   |   |   |   |   |   |--- housing_rent >  0.50
|   |   |   |   |   |   |   |   |--- other_credit_none <= 0.50
|   |   |   |   |   |   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- savings_balance >  0.00
```

```
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |   |   |--- other_credit_none >  0.50
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 2.50
|   |   |   |   |   |   |   |   |   |   |   |--- years_at_residence <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- years_at_residence >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- savings_balance <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [12.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |   |--- savings_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  2.50
|   |   |   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 16.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [4.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |   |   |--- months_loan_duration >  16.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount <= 5186.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 4
|   |   |   |   |   |   |   |   |   |   |   |   |--- amount >  5186.50
|   |   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- amount >  8472.00
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |--- credit_history >  3.50
|   |   |   |   |--- other_credit_none <= 0.50
|   |   |   |   |   |--- months_loan_duration <= 15.00
|   |   |   |   |   |   |--- amount <= 386.00
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- amount >  386.00
|   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |--- months_loan_duration >  15.00
|   |   |   |   |   |   |--- age <= 40.00
|   |   |   |   |   |   |   |--- dependents <= 1.50
|   |   |   |   |   |   |   |   |--- weights: [8.00, 0.00] class: 0
|   |   |   |   |   |   |   |--- dependents >  1.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |--- age >  40.00
|   |   |   |   |   |   |   |--- employment_duration <= 3.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- employment_duration >  3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |--- other_credit_none >  0.50
|   |   |   |   |   |--- age <= 41.00
|   |   |   |   |   |   |--- weights: [0.00, 14.00] class: 1
|   |   |   |   |   |--- age >  41.00
|   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |--- months_loan_duration >  26.50
|   |   |   |--- checking_balance <= 2.50
|   |   |   |   |--- savings_balance <= 0.00
|   |   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |   |--- amount <= 11257.00
|   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |--- amount >  11257.00
|   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |   |--- weights: [6.00, 0.00] class: 0
|   |   |   |   |--- savings_balance >  0.00
|   |   |   |   |   |--- months_loan_duration <= 47.50
|   |   |   |   |   |   |--- percent_of_income <= 3.50
|   |   |   |   |   |   |   |--- purpose_car <= 0.50
```

```
|   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |--- savings_balance <= 1.50
|   |   |   |   |   |   |   |   |   |   |--- amount <= 3262.00
|   |   |   |   |   |   |   |   |   |   |   |--- percent_of_income <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- percent_of_income >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- amount >  3262.00
|   |   |   |   |   |   |   |   |   |   |   |--- age <= 47.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- age >  47.00
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- savings_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 7.00] class: 1
|   |   |   |   |   |   |   |--- purpose_car >  0.50
|   |   |   |   |   |   |   |   |--- amount <= 7759.00
|   |   |   |   |   |   |   |   |   |--- weights: [7.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  7759.00
|   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 39.00
|   |   |   |   |   |   |   |   |   |   |--- months_loan_duration <= 33.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |--- months_loan_duration >  33.00
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 3.00] class: 1
|   |   |   |   |   |   |   |   |   |--- months_loan_duration >  39.00
|   |   |   |   |   |   |   |   |   |   |--- weights: [2.00, 0.00] class: 0
|   |   |   |   |   |   |--- percent_of_income >  3.50
|   |   |   |   |   |   |   |--- age <= 29.50
|   |   |   |   |   |   |   |   |--- amount <= 2364.50
|   |   |   |   |   |   |   |   |   |--- age <= 23.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |   |   |   |   |   |   |--- age >  23.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |--- amount >  2364.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 12.00] class: 1
|   |   |   |   |   |   |   |--- age >  29.50
|   |   |   |   |   |   |   |   |--- amount <= 2319.50
|   |   |   |   |   |   |   |   |   |--- weights: [0.00, 4.00] class: 1
|   |   |   |   |   |   |   |   |--- amount >  2319.50
|   |   |   |   |   |   |   |   |   |--- credit_history <= 2.50
|   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |--- credit_history >  2.50
|   |   |   |   |   |   |   |   |   |   |--- phone <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |--- weights: [0.00, 5.00] class: 1
|   |   |   |   |   |   |   |   |   |   |--- phone >  1.50
|   |   |   |   |   |   |   |   |   |   |   |--- checking_balance <= 1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- weights: [3.00, 0.00] class: 0
|   |   |   |   |   |   |   |   |   |   |   |--- checking_balance >  1.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- truncated branch of depth 2
|   |   |   |   |   |--- months_loan_duration >  47.50
|   |   |   |   |   |   |--- amount <= 3705.00
|   |   |   |   |   |   |   |--- credit_history <= 3.50
|   |   |   |   |   |   |   |   |--- weights: [0.00, 2.00] class: 1
|   |   |   |   |   |   |   |--- credit_history >  3.50
|   |   |   |   |   |   |   |   |--- weights: [1.00, 0.00] class: 0
|   |   |   |   |   |   |--- amount >  3705.00
|   |   |   |   |   |   |   |--- weights: [0.00, 18.00] class: 1
|   |   |   |--- checking_balance >  2.50
|   |   |   |   |--- amount <= 2462.50
```

```
|   |   |   |   |--- weights: [0.00, 1.00] class: 1
|   |   |   |--- amount >  2462.50
|   |   |   |   |--- weights: [7.00, 0.00] class: 0
```

General Models Observations

Remark:

- A `Cost Function` quantifies the error between predicted values and expected values and presents it in the form of a single real number.
- The bank can choose a model depending on their cost function which they want to minimize. The cost function can depend on the geographical and economical conditions due to changes in interest rates, government rules etc.
- Bank's main aim would be to balance the trade off between losing an opportunity (to gain money by giving loans) in case of FP and losing the money in case of FN.
- We emphasized that recall is the metric of interest here and we tuned our model on recall. But this does not mean that other metrics should be ignored completely.
- Here, we assumed that the $cost\ on\ FN > cost\ on\ FP$, but we do not want to misclassify so many non-defaulters that the equation is reversed i.e. $cost\ on\ FP > cost\ on\ FN$, hence, the bank will actually be losing money in the longer run.
- `For example:` Let's say a bank gains 4% interest on the amount while giving loans to non-defaulters and lose 70% on the amount while giving loans to defaulters and also assume that the amount of loan is also fixed. Let's say we use a model which is only good at identifying defaulters and we get the following result:
  - The model identifies 10 out of 10 defaulters
  - The model identifies 10 out of 190 non-defaulters i.e. misclassified 180 non-defaulters
  - Then, the money saved is $0.7 * 10 * amount = 7 * amount$ and the money lost is $0.04 * 180 * amount = 7.2 * amount$
  - As the value of money lost is greater than the value of money saved, bank will actually lose money even after identifying all defaulters

---

Busines Recommendations

In [ ]: