CREDIT RISK ANALYSIS USING MACHINE LEARNING

Project by:
Rahul Barpat,
PGDM,
Institute of Public Enterprise.

INDEX

ABSTRACT

A key activity within the banking industry is to extend credit to customers, hence, credit risk analysis is critical for financial risk management. There are various methods used to perform credit risk analysis. In this project, we analyze German data from UC Irvine Machine Learning repository, reproducing results previously published in literature. Further, using the same dataset and various machine learning algorithms, we attempt to create better models by tuning available parameters.

In this report, we have explained the algorithms that goes behind developing the machine learning models. We conclude with a discussion and comparison of summarizing the best approach to classify these datasets. Simple Linear Regression, Multiple Linear Regression

Logistic Linear Regression, K- Nearest Neighbors (KNN), Decision Tree, Random Forest, Adaboost, K-Means Clustering and Hierarchical Clustering are the machine learning models used for this report.

MOTIVATION:

A significant activity of the banking industry is to extend credit to customers. Credit risk management evaluates available data and decides the credibility of a customer, with the intent of protecting the financial institution against fraud.

OBJECTIVE OF ANALYSIS:

1. **Prediction of Credit Amount.**

Prediction of the Credit amount with the help of independent variables which are

- Age
- Duration of credit
- Duration of Current Employment

2. **Minimization of risk and maximization of profit on behalf of the bank.**

To minimize loss from the bank's perspective, the bank needs a decision rule regarding who to give approval of the loan and who not to. An applicant's demographic and socio-economic profiles are considered by loan managers before a decision is taken regarding his/her loan application.
The German Credit Data contains data on 20 variables and the classification whether an applicant is considered a Good or a Bad credit risk for 1000 loan applicants. Here is a link to the German Credit data. A predictive model developed on this data is expected to provide a bank manager guidance for deciding whether to approve a loan to a prospective applicant based on his/her profiles.
If the applicant has a good credit risk - likely to repay loan - not approving loan is a loss of business to bank
If the applicant has a bad credit risk - not likely to repay loan - approving loan results in a financial loss to bank Applicant's demographic & socio-economic profiles are studied by loan managers before granting loan. German Credit Data contains data on 20 variables & the classification whether an applicant is considered a Good or a Bad credit risk for 1000 loan applicants. A predictive model developed on this data is expected to provide a bank manager guidance for making a decision whether to approve a loan to a prospective applicant based on his/her profiles.

3. **Segmentation of the Customers.**

In marketing, segmentation refers to the process of dividing a market into smaller groups of consumers with similar needs or characteristics. This is often done to identify specific target markets and to tailor marketing strategies to better meet the needs and preferences of these groups.

DATA SET

The German Credit data set is a publicly available data set downloaded from the UCI Machine Learning Repository. The German Credit Data contains data on 20 variables and the classification of whether an applicant is considered a Good or Bad credit risk for 1000 loan applicants. The task requires exploring the data and building a predictive model to provide a bank manager guidance for deciding on whether to approve a loan to a prospective applicant based on his/her profile.

Data dictionary - Business meaning of each column

Attribute 1: (qualitative) Account status of existing checking account based on Debit Memorandum. Debit Memorandum notifies the customers about the debit adjustment. 1 : ... < 0 DM 2 : 0 <= ... < 200 DM 3 : ... >= 200 DM / salary assignments for at least 1 year 4 : no checking account
Attribute 2: (numerical) Duration_of_credit Duration of loan in months
Attribute 3: (qualitative) Payment_status_of_previous_credit Credit history of the applicant 0 : no credits taken/all credits paid back duly 1 : all credits at this bank paid back duly 2 : existing credits paid back duly till now 3 : delay in paying off in the past 4 : critical account/other credits existing (not at this bank)
Attribute 4: (qualitative) Purpose Purpose for the loan 0 : car (new) 1 : car (used) 2 : furniture/equipment 3 : radio/television 4 : others 5 : repairs 6 : education 7 : vacation 8 : business 9 : retraining 10 : domestic appliances
Attribute 5: (numerical) Credit_amount Amount taken as loan
Attibute 6: (qualitative) Value_savings_stocks Savings account and bonds 1 : ... < 100 DM 2 : 100 <= ... < 500 DM 3 : 500 <= ... < 1000 DM 4 : .. >= 1000 DM 5 : unknown/ no savings account
Attribute 7: (qualitative) Duration_of_current_employment Number of years worked in the present job 1 : unemployed 2 : ... < 1 year 3 : 1 <= ... < 4 years 4 : 4 <= ... < 7 years 5 : .. >= 7 years
Attribute 8: (numerical) Instalment_percent Installment rate in percentage of disposable income
Attribute 9: (qualitative) Marital_status_gender Personal status and gender 1 : male : divorced/separated 2 : female : divorced/separated/married 3 : male : single 4 : male : married/widowed 5 : female : single
Attribute 10: (qualitative) Guarantors Other debtors / guarantors for the applicant 1 : none 2 : co-applicant 3 : guarantor
Attribute 11: (numerical) Duration_in_current_address Number of years lived at the present address
Attribute 12: (qualitative) Property Property type of applicant A1 : real estate 2 : if not A121 : building society savings agreement/life insurance 3 : if not A121/A122 : car or other, not in attribute 6 4 : unknown / no property
Attribute 13: (numerical) Age Age in years
Attribute 14: (qualitative) Concurrent_credits Other installment plans 1 : bank 2 : stores 3 : none
Attribute 15: (qualitative) Housing 1 : rent 2 : own 3 : for free
Attribute 16: (numerical) No_of_credits_at_this_bank Number of existing credits at this bank
Attribute 17: (qualitative) Occupation 1 : unemployed/ unskilled - non-resident 2 : unskilled - resident 3 : skilled employee / official 4 : management/ self-employed/highly qualified employee/ officer
Attribute 18: (numerical) No_of_dependents Number of people being liable to provide maintenance for
Attribute 19: (qualitative) Telephone Is the Telephone registered or not 1 : none 2 : yes, registered under the customers name
Attribute 20: (qualitative) foreign worker Is the applicant a foreign worker 1 : yes 2 : no
Creditability: Whether the issued loan was a good decision or bad 1 = Good Credit Risk 0 = Bad Credit Risk

METHODOLOG

1. Frequency Distribution:

Frequency distributions tell us how frequencies are distributed over the values. That is how many values lie between different intervals.

They give us an idea about the range where most of the values fall and the ranges where values are scarce.

A frequency distribution is an overview of all values of some variable and the number of timesthey occur.

2. Contingency Table:

Estimations like mean, median, standard deviation, and variance are very much useful in case ofthe univariate data analysis.

But in the case of bivariate analysis (comparing two variables) correlation comes into play.

Contingency Table is one of the techniques for exploring two or even more variables. It is basically a tally of counts between two or more categorical variables.

A contingency table, sometimes called a two-way frequency table, is a tabular mechanism withat at least two rows and two columns used in statistics to present categorical data in terms of frequency counts. More precisely, an r×c contingency table shows the observed frequency of two variables, the observed frequencies of which are arranged into r rows and c columns. The intersection of a row and a column of acontingency table is called a cell.

3. Chi Square Test:

The Chi-square test is intended to test how likely it is that an observed distribution is due to chance. It is also called a "goodness of fit" statistic because it measures how well the observed distribution of data fits with the distribution that is expected if the variables are independent.

A Chi-square test is designed to analyze categorical data. That means that the data has been counted and divided into categories. It will not work with parametric or continuous data (such as height in inches). For example, if you want to test whether attending class influences how students perform on an exam, using test scores (from 0-100) as data would not be appropriate for a Chi-square test. However, arranging students into the categories "Pass" and "Fail" would.

Additionally, the data in a Chi-square grid should not be in the form of percentages, or anything other than frequency (count) data.

4. Null Hypothesis:

The null hypothesis states that there is no statistical significance exists between sets of data which implies that the population parameter will be equal to a hypothesized value.

The null hypothesis assumes that any kind of difference between the chosen characteristics that you see in a set of data is due to chance.

For example, if the expected earnings for the gambling game are truly equal to zero, then anydifference between the average earnings in the data and zero is due to chance.

5. Categorical Plots:

Plots are basically used for visualizing the relationship between variables. Those variables can be either completely numerical or a category like a group, class or division. This article deals with categorical variables and how they can be visualized using the Seaborn library provided by Python.
Seaborn besides being a statistical plotting library also provides some default datasets. We will be using one such default dataset called 'tips'.
The 'tips' dataset contains information about people who probably had food at a restaurant and whether they left a tip for the waiters, their gender, whether they smoke, and so on.

6. Numerical Plots:

Numerical data represent values that can be measured and put into a logical order. Examples of numerical data are height, weight, age, number of movies watched, IQ, etc. To graph numerical data, one uses dot plots, stem and leaf graphs, histograms, box plots, ogive graphs, and scatter plot.

7. Descriptive Analysis:

Python Descriptive Statistics process describes the basic features of data in a study. It delivers summaries on the sample and the measures and does not use the data to learn about the population it represents.
Under descriptive statistics, fall two sets of properties- central tendency and dispersion. Python Central tendency characterizes one central value for the entire distribution.
Measures under this include mean, median, and mode. Python Dispersion is the term for a practice that characterizes how apart the members of the distribution are from the center and from each other. Variance/Standard Deviation is one such measure of variability.
describe() is used for Exploratory Data Analysis
Function for Exploratory Analysis or Descriptive/ Summary Statistics
describe(include='all') - Summary statistics for both numeric/ categorical data items/features
describe() - Default EDA - Calculates summary statistics for numerical features count, mean, standard deviation, minimum, maximum, 25%, 50%, 75% quantiles
describe(include=['object']) - Summary statistics for categorical columns (object datatype)
count, unique, top, frequency

```
data.describe(include='all')            # EDA for numeric & categorical features too
data.describe(include=['int64'])        # EDA for int64 features
data.describe(include=['float64'])      # EDA for float64 features
data.describe()                         # EDA for numeric features only
data.describe(include=['object'])       # EDA for categorical features
```

8. Diagnostic Analysis:

Diagnostic analytics is the process of using data to determine the causes of trends and correlations between variables. It can be viewed as a logical next step after using descriptive analytics to identify trends. Diagnostic analysis can be done manually, using an algorithm, or withstatistical software (such as Microsoft Excel).

There are several concepts to understand before diving into diagnostic analytics: hypothesis testing, the difference between correlation and causation, and diagnostic regression analysis. The Explanation of the root cause behind the outcome is considered under descriptiveanalytics.

Correlation coefficient is denoted by r

If [r=1], features have a perfect positive correlation i.e., if one variable moves a given amount, the second moves proportionally in the same direction

If [r=0], no relationship exists between the features. If one variable moves, you can makeno predictions about the movement of the other variable; they are uncorrelated

If [r=–1], the variables are perfectly negatively correlated (or inversely correlated) & movein opposition to each other. If one variable increases, the other variable decreases proportionally

Measuring correlation graphically using scatter_matrix() & heatmap() functions

Correlation = 0 - No relationship exists between the features

Correlation = 1 - Perfect positive relationship

Correlation = -1 - Perfect negative relationship

Correlation in Python:

corr() - Karl Pearson's Coefficient of Correlation
Selects only numeric features for further analysis

num_feature - Set of numeric features - int64 or float64 - 12 features (11are int64 & 1 is float64)

scatter() - Graphically plot 2 variables
Plot independent variable on x-axis & dependent variable on y axis
Limitation - Only 2 variables can be plotted

scatter_matrix() - diagonal - kde = kernel density estimation

hist = histogram

pairplot() - Measures association between multiple features

heatmap() - Measures association between multiple features

9. Standardization

Standardization is a technique in machine learning that is used to transform a dataset so that it has a mean of 0 and a standard deviation of 1. This is often useful when the features in the dataset have different scales, as it can help to balance the importance of the features in the model.

Standardization is typically performed by subtracting the mean of each feature from each value and then dividing by the standard deviation. This results in a transformed dataset with zero mean and unit variance

10. One Hot Encoding:

One hot encoding is a technique in machine learning that is used to encode categorical variables as numerical data. It is often used as a preprocessing step before training a model, as many machine learning algorithms do not work well with categorical data in its raw form.
One hot encoding works by converting each categorical value into a new binary column, with a value of 1 indicating the presence of the categorical value and a value of 0 indicating its absence.

11. Simple Linear Regression Model:

It is used to estimate the relationship between two quantitative variables. You can use simple linear regression when you want to know:
How strong the relationship is between two variables (e.g., the relationship between rainfall andsoil erosion).
The value of the dependent variable at a certain value of the independent variable (e.g., the amount of soil erosion at a certain level of rainfall).
Regression models describe the relationship between variables by fitting a line to the observed data. Linear regression models use a straight line, while logistic and nonlinear regression modelsuse a curved line. Regression allows you to estimate how a dependent variable changes as the independent variable(s) change.

12. Multi Linear Regression Model:

Multiple regression is like linear regression, but with more than one independent value, meaningthat we try to predict a value based on two or more variables.
Multiple Linear Regression attempts to model the relationship between two or more features and a response by fitting a linear equation to observed data.
The steps to perform multiple linear Regression are almost like that of simple linear Regression. The Difference Lies in the evaluation. We can use it to find out which factor has the highest impacton the predicted output and how different variables relate to each other.

13. Logistic Regression:

Logistic regression is a supervised machine learning algorithm that is used to predict a binary outcome. It is a type of regression analysis that is used to predict the probability of an event occurring based on certain independent variables.
In logistic regression, the outcome is a binary variable that can take on only two values, such as 0 or 1, or "Yes" or "No". The independent variables, also known as the predictors, are used to predict the probability of the outcome occurring.
The logistic regression model is based on the concept of the logit function, which is used to model the probability of an event occurring. The logit function is defined as the natural logarithm of the odds of an event occurring, and it takes on values between -infinity and +infinity.

In logistic regression, the goal is to find the optimal values for the model's parameters that maximize the probability of the outcome occurring. This is done using an optimization algorithm, such as gradient descent, to minimize the error between the predicted probabilities and the actual outcomes

14. KNN Classification model:

K-nearest neighbors (KNN) is a supervised machine learning algorithm that is used for classification and regression tasks. It is a simple and effective approach that is based on the idea of using the class labels of the "k" nearest training examples to predict the class label of a new example.

In KNN classification, the model makes predictions based on the class labels of the "k" nearest training examples in the feature space. The value of "k" is a hyperparameter that is chosen by the user. A larger value of "k" means that the model will be more resistant to noise, but it may also be less sensitive to subtle patterns in the data.

Decision Tree Classification model:

Decision tree classification is a supervised machine learning algorithm that is used to predict a categorical outcome. It is a tree-based model that makes predictions based on a series of decisions based on the values of the features in the data.

In a decision tree classification model, the data is split into smaller subgroups based on the values of the features. At each step in the tree, a decision is made based on the value of a feature, and the data is partitioned into the branches of the tree based on this decision. This process is repeated until the tree is fully grown, and the final leaf nodes represent the predicted class labels for the data.

15. Random Forest classification model:

Random forest classification is a supervised machine learning algorithm that is used to predict a categorical outcome. It is an ensemble method that combines the predictions of multiple decision tree classifiers to improve the overall accuracy and stability of the model.

In a random forest classification model, a large number of decision trees are trained on randomly selected subsets of the data. At each split in the tree, a random subset of the features is chosen as the split point, rather than using the best split point based on all features as in a single decision tree. This process is repeated for each tree in the forest, and the final prediction is made by aggregating the predictions of all the trees

16. AdaBoost Classification model:

AdaBoost (Adaptive Boosting) is a supervised machine learning algorithm that is used for classification and regression tasks. It is an ensemble method that combines the predictions of multiple weak learners to improve the overall accuracy and stability of the model.

In AdaBoost, weak learners are simple models that are trained on the data and make predictions. These predictions are then used to weight the training examples, with the goal of misclassified examples receiving more weight in subsequent iterations. This process is repeated until the desired number of weak learners has been trained, and the final prediction is made by aggregating the predictions of all the weak learners

17. K Means Clustering Model:

K-means clustering is an unsupervised machine learning algorithm that is used to group a set of data points into "k" clusters based on their similarity. It is a popular method for clustering and is based on the idea of iteratively assigning each point to the nearest cluster center and then updating the cluster centers based on the mean of the points assigned to them.

The algorithm starts by randomly initializing "k" cluster centers, and then it iteratively performs the following steps until convergence:

Assign each data point to the nearest cluster center.

Update the cluster centers by taking the mean of all the data points assigned to each cluster.

Repeat these steps until the cluster centers do not change or a maximum number of iterations is reached.

18. Hierarchical Clustering Model:

Hierarchical clustering is an unsupervised machine learning algorithm that is used to group a set of data points into a tree-like structure called a dendrogram. It is a popular method for clustering and is based on the idea of building a hierarchy of clusters, where each cluster is a sub-cluster of the next higher level cluster.

There are two main types of hierarchical clustering:

Agglomerative: This method starts with each data point as a separate cluster and then merges the clusters iteratively based on their similarity.

Divisive: This method starts with all the data points in a single cluster and then splits the cluster into sub-clusters iteratively based on their dissimilarity.

.

19. Cross Validation:

Cross-validation is a technique in machine learning that is used to evaluate the performance of a model and to tune its hyperparameters. It involves dividing the training dataset into a set of smaller subsets, and then training and evaluating the model on each subset. The results are then averaged to get an overall estimate of the model's performance.

There are several types of cross-validation, including:

- K-fold cross-validation
- Stratified K-fold cross-validation
- Leave-one-out cross-validation

20. Grid Search Cross Validation:

Grid search cross-validation is a technique in machine learning that is used to tune the hyperparameters of a model. It involves evaluating the model on a grid of hyperparameter values using cross-validation, and selecting the best set of hyperparameters based on the model performance.

ANALYSIS AND INTERPRETATION

1. **Importing the data**.

Load the libraries

In [1]:
```
import warnings                                  # Supressing the warning messages
warnings.filterwarnings('ignore')
import pandas            as pd
import numpy             as np
import scipy             as sp
import matplotlib.pyplot as plt
import seaborn           as sns
import statsmodels.api   as sm

from scipy                                 import stats
from scipy.cluster                         import hierarchy as sch
from sklearn.preprocessing                 import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.model_selection               import train_test_split, KFold, GridSearchCV
from sklearn.feature_selection             import RFE

from sklearn.linear_model                  import LinearRegression, LogisticRegression
from sklearn.neighbors                     import KNeighborsClassifier
from sklearn.tree                          import DecisionTreeClassifier
from sklearn.ensemble                      import RandomForestClassifier
from sklearn.ensemble                      import AdaBoostClassifier
from sklearn.cluster                       import KMeans, AgglomerativeClustering

from sklearn.metrics                       import r2_score, mean_squared_error, max_error, mean_absolute_error
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from sklearn.metrics                       import roc_auc_score, classification_report, confusion_matrix, accuracy_score, silhou
```

In [2]:
```
# Read the dataset - German_Credit

gc = pd.read_excel(r"C:\Users\user\Desktop\coding\ML\ML Models\Regression\2022_ICSSR_ExcelWB_DataAnalysis.xls", sheet_name = 'Ger
gc.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Creditability                      1000 non-null   int64
 1   Account_status                     1000 non-null   int64
 2   Duration_of_credit                 1000 non-null   int64
 3   Payment_status_of_previous_credit  1000 non-null   int64
 4   Purpose                            1000 non-null   int64
 5   Credit_amount                      1000 non-null   int64
 6   Value_savings_stocks               1000 non-null   int64
 7   Duration_of_current_employment     1000 non-null   int64
 8   Instalment_percent                 1000 non-null   int64
 9   Marital_status_gender              1000 non-null   int64
 10  Guarantors                         1000 non-null   int64
 11  Duration_in_current_address        1000 non-null   int64
 12  Property                           1000 non-null   int64
 13  Age                                1000 non-null   int64
 14  Concurrent_credits                 1000 non-null   int64
 15  Housing                            1000 non-null   int64
 16  No_of_credits_at_this_bank         1000 non-null   int64
 17  Occupation                         1000 non-null   int64
 18  No_of_dependents                   1000 non-null   int64
 19  Telephone                          1000 non-null   int64
```

```
In [3]: gc.head(20)
```

Out[3]:

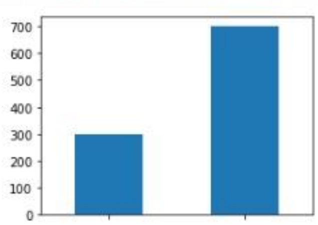
	Creditability	Account_status	Duration_of_credit	Payment_status_of_previous_credit	Purpose	Credit_amount	Value_savings_stocks	Duration_of_current_em
0	1	1	18	4	2	1049	1	
1	1	1	9	4	0	2799	1	
2	1	2	12	2	9	841	2	
3	1	1	12	4	0	2122	1	
4	1	1	12	4	0	2171	1	
5	1	1	10	4	0	2241	1	
6	1	1	8	4	0	3398	1	
7	1	1	6	4	0	1361	1	
8	1	4	18	4	3	1098	1	
9	1	2	24	2	3	3758	3	
10	1	1	11	4	0	3905	1	
11	1	1	30	4	1	6187	2	
12	1	1	6	4	3	1957	1	
13	1	2	48	3	10	7582	2	
14	1	1	18	2	3	1936	5	
15	1	1	6	2	3	2647	3	
16	1	1	11	4	0	3939	1	
17	1	2	18	2	3	3213	3	

2. Frequency distribution for creditability.

Frequency Distribution

```
In [4]: # Frequency Distribution - value_counts()

        #gc['Creditability'].value_counts()
        gc.Creditability.value_counts()

Out[4]: 1    700
        0    300
        Name: Creditability, dtype: int64
```

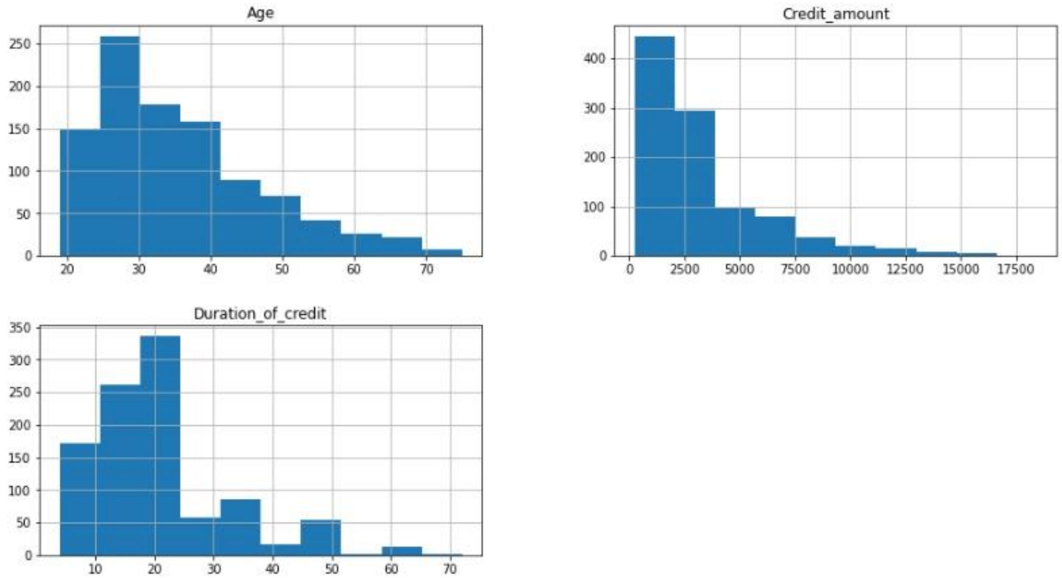
```
In [5]: # Creating Bar chart as the Target variable (Creditability) is Categorical

        GroupedData = gc.groupby('Creditability').size()
        print(GroupedData)
        GroupedData.plot(kind='bar', figsize=(4,3))

        Creditability
        0    300
        1    700
        dtype: int64

Out[5]: <AxesSubplot:xlabel='Creditability'>
```



Interpretation:
Out of 1000 records 700 are Good Credit risk and 300 are Bad Credit risk

3. Histogram for Age, Credit amount and Duration of credit.

```
In [8]: # Plotting histograms of multiple columns together

        gc.hist(['Age', 'Credit_amount','Duration_of_credit'], figsize=(15,8))
        plt.show()
```



Interpretation:

- Majority of the people are from the age group of 25 to 30
- Majority of the people have taken the credit amount between 0 to 5000
- Majority duration of the credit is 15 to 25 months

4. Contingency table and Chi-Square Test.

Contingency table for Creditability, Guarantors and Foreign workers.
Chi-Square test to check the association between Creditability and Guarantors.

Contingency Table

```
In [7]: # Contingency Table - crosstable()

        CT = pd.crosstab(index=gc.Creditability, columns=[gc.Foreign_worker,gc.Guarantors])
        print(CT)
```

```
Foreign_worker    1           2
Guarantors        1   2   3   1  2  3
Creditability
0               270  18   8   2  0  2
1               611  19  37  24  4  5
```

```
In [10]: # checking whether there is relation between creditability and guarantors using chi_square test of independence
         # H0 = There is no relation(Independent)
         # H1 = There is relation(Dependent)
         from scipy.stats import chi2_contingency
         CT = pd.crosstab(gc.Creditability, gc['Guarantors'])
         print(CT)
         stat, p, dof, expected = chi2_contingency(CT)

         alpha = 0.05
         print("p value is " + str(p))
         if p <= alpha:
             print('Dependent (reject H0)')
         else:
             print('Independent (H0 holds true)')
```

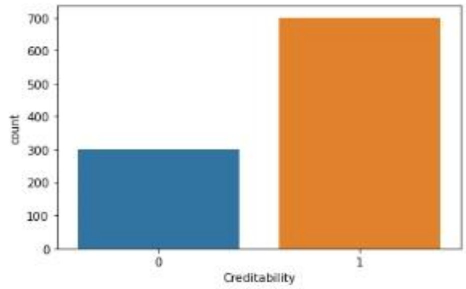
```
Guarantors       1   2   3
Creditability
0              272  18  10
1              635  23  42
p value is 0.036055954027247206
Dependent (reject H0)
```

Interpretation:

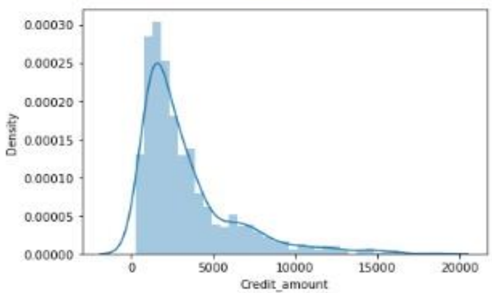
- Foreign workers have Good Credit risk.
- There is association between Guarantors and Creditability.

5. Explanatory Data Analysis- Graphical Method

```
In [25]: sns.countplot(gc.Creditability)
Out[25]: <AxesSubplot:xlabel='Creditability', ylabel='count'>
```




```
In [26]: # distplot() function - Plot Numerical continuous feature

         sns.distplot(gc.Credit_amount)
Out[26]: <AxesSubplot:xlabel='Credit_amount', ylabel='Density'>
```

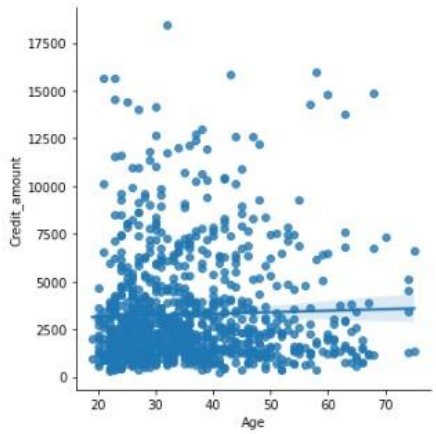


Interpretation:

- Out of 1000 records, 700 records have Good Credit Risk and 300 records have Bad Credit Risk
- Credit amount has positively skewed distribution and majority of the people have taken the credit amount between 0 to 5000

6. Scatter plot to check association between Age and Credit Amount

```
In [27]: #Association between Age vs. Credit_amount
         np.corrcoef(gc.Credit_amount,gc.Age)

Out[27]: array([[1.        , 0.03227268],
                [0.03227268, 1.        ]])

In [28]: # lmplot() - Plot Numeric vs. Numeric features
         # Age vs. Credit_amount

         sns.lmplot(x = 'Age', y = 'Credit_amount', data = gc)

Out[28]: <seaborn.axisgrid.FacetGrid at 0x201de186430>
```

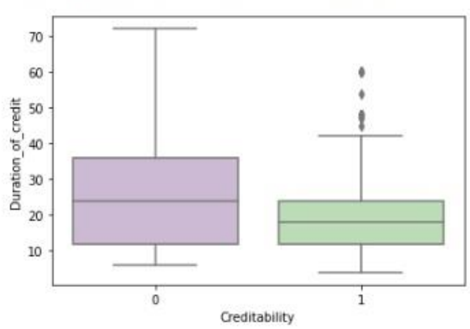


Interpretation:

- There is weak association between them.

7. Box plot between Creditability and Duration of credit.

```
In [29]: # boxplot() function - Plot Numeric vs. Categorical features
         # Creditability vs Duration of credit
         sns.boxplot(x = 'Creditability', y = 'Duration_of_credit', data = gc, palette="PRGn")

Out[29]: <AxesSubplot:xlabel='Creditability', ylabel='Duration_of_credit'>
```



Interpretation

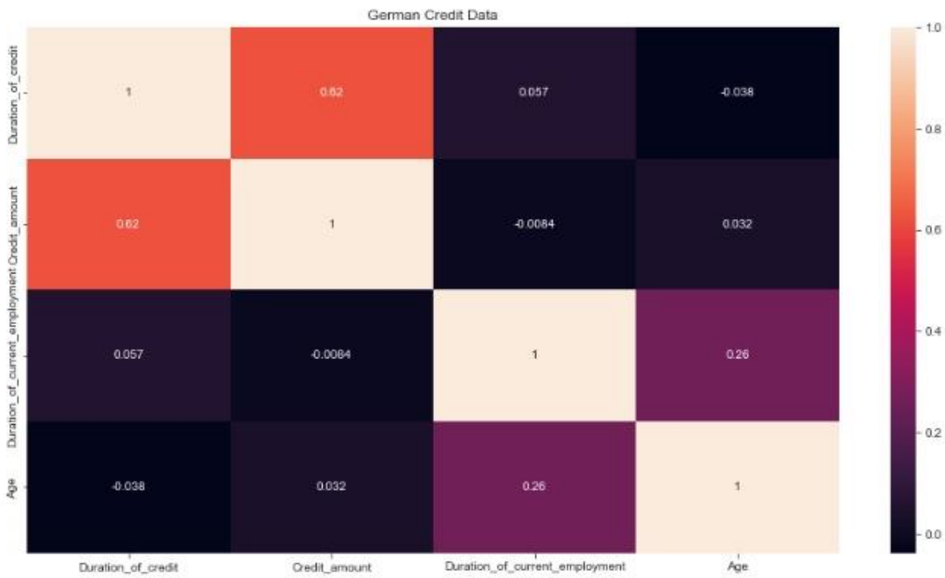
Interpretation:

- Maximum duration of repaying for Good Credit risk is 40 months and average is 15 months.
- Outlier values are observed in Good Credit risk

- More than 40 months has Bad credit risk

8. Heat Map for Age, Duration of credit, Duration of credit employment and Credit Amount.

```
In [39]: # 5. heatmap() - Measures association between muliptle features

         plt.figure(figsize=(15,8))
         sns.set_style('ticks')

         sns.heatmap(gc.corr(), annot=True)

         plt.title('German Credit Data')
         plt.show()
```



Interpretation:

- Moderate positive correlation between Duration of credit and Credit Amount.
- Weak Positive correlation between Age and Duration of current employment and Age

9. Pair Plot for the entire data set

10. Explanatory Data analysis – Descriptive Analysis

In [31]:
```
# Describes only numeric values

gc.describe()
```

Out[31]:

	Duration_of_credit	Credit_amount	Duration_of_current_employment	Age
count	1000.000000	1000.00000	1000.000000	1000.00000
mean	20.903000	3271.24800	3.384000	35.54200
std	12.058814	2822.75176	1.208306	11.35267
min	4.000000	250.00000	1.000000	19.00000
25%	12.000000	1365.50000	3.000000	27.00000
50%	18.000000	2319.50000	3.000000	33.00000
75%	24.000000	3972.25000	5.000000	42.00000
max	72.000000	18424.00000	5.000000	75.00000

In [33]:
```
#gc.describe().T

# EDA for entire data

gc.describe(include='all')
```

Out[33]:

	Creditability	Account_status	Duration_of_credit	Payment_status_of_previous_credit	Purpose	Credit_amount	Value_savings_stocks	Duration_of_current
count	1000.0	1000.0	1000.000000	1000.0	1000.0	1000.00000	1000.0	
unique	2.0	4.0	NaN	5.0	10.0	NaN	5.0	
top	1.0	4.0	NaN	2.0	3.0	NaN	1.0	
freq	700.0	394.0	NaN	530.0	280.0	NaN	603.0	
mean	NaN	NaN	20.903000	NaN	NaN	3271.24800	NaN	
std	NaN	NaN	12.058814	NaN	NaN	2822.75176	NaN	
min	NaN	NaN	4.000000	NaN	NaN	250.00000	NaN	
25%	NaN	NaN	12.000000	NaN	NaN	1365.50000	NaN	
50%	NaN	NaN	18.000000	NaN	NaN	2319.50000	NaN	
75%	NaN	NaN	24.000000	NaN	NaN	3972.25000	NaN	
max	NaN	NaN	72.000000	NaN	NaN	18424.00000	NaN	

11. Correlation:

In [34]:
```
# Diagnostic Analysis - corr() - Create correlation matrix - Karl Pearson's Coefficient of Correlation

gc.corr()
```

Out[34]:

	Duration_of_credit	Credit_amount	Duration_of_current_employment	Age
Duration_of_credit	1.000000	0.624988	0.057381	-0.037550
Credit_amount	0.624988	1.000000	-0.008376	0.032273
Duration_of_current_employment	0.057381	-0.008376	1.000000	0.259116
Age	-0.037550	0.032273	0.259116	1.000000

Interpretation:

- Moderate positive correlation between Duration of credit and Credit Amount.
- Weak Positive correlation between Age and Duration of current employment and Age

12. Data Preprocessing:

12.1. Missing and Null Values

```
In [20]: # Check for NULL values in the dataset

         gc.isna().sum()

Out[20]: Creditability                       0
         Account_status                      0
         Duration_of_credit                  0
         Payment_status_of_previous_credit   0
         Purpose                             0
         Credit_amount                       0
         Value_savings_stocks                0
         Duration_of_current_employment      0
         Instalment_percent                  0
         Marital_status_gender               0
         Guarantors                          0
         Duration_in_current_address         0
         Property                            0
         Age                                 0
         Concurrent_credits                  0
         Housing                             0
         No_of_credits_at_this_bank          0
         Occupation                          0
         No_of_dependents                    0
         Telephone                           0
         Foreign_worker                      0
         dtype: int64
```

```
gc.isnull().sum()                                       # Count of missing values
#round(gc.isnull().sum() / gc.isnull().count() * 100, 2)    # Percentage of missing values

Out[19]: Creditability                       0
         Account_status                      0
         Duration_of_credit                  0
         Payment_status_of_previous_credit   0
         Purpose                             0
         Credit_amount                       0
         Value_savings_stocks                0
         Duration_of_current_employment      0
         Instalment_percent                  0
         Marital_status_gender               0
         Guarantors                          0
         Duration_in_current_address         0
         Property                            0
         Age                                 0
         Concurrent_credits                  0
         Housing                             0
         No_of_credits_at_this_bank          0
         Occupation                          0
         No_of_dependents                    0
         Telephone                           0
         Foreign_worker                      0
         dtype: int64
```

Interpretation:
There are no missing and null values.

12.2. **One Hot Encoding**.

```
In [15]: gc1= pd.get_dummies(gc, columns=['Creditability', 'Account_status',
                'Payment_status_of_previous_credit', 'Purpose',
                'Value_savings_stocks', 'Duration_of_current_employment',
                'Instalment_percent', 'Marital_status_gender', 'Guarantors',
                'Duration_in_current_address', 'Property',  'Concurrent_credits',
                'Housing', 'No_of_credits_at_this_bank', 'Occupation',
                'No_of_dependents', 'Telephone', 'Foreign_worker'],drop_first=True)
         gc1.head()
```

Out[15]:

	Duration_of_credit	Credit_amount	Age	Creditability_1	Account_status_2	Account_status_3	Account_status_4	Payment_sta
0	18	1049	21	1	0	0	0	
1	9	2799	36	1	0	0	0	
2	12	841	23	1	1	0	0	
3	12	2122	39	1	0	0	0	
4	12	2171	38	1	0	0	0	

5 rows × 55 columns

12.3. **Standardization**

```
In [16]: # Standard Scaling (Standardization)
         gc2=gc[['Duration_of_credit','Credit_amount','Age','Duration_of_current_employment']]
         standardizer = StandardScaler()
         gc2 = pd.DataFrame(standardizer.fit_transform(gc2))
         gc2.columns=[  'Duration_of_credit',
                        'Credit_amount',
                        'Age',
                        'Duration_of_current_employment']
         gc2.head()
```

Out[16]:

	Duration_of_credit	Credit_amount	Age	Duration_of_current_employment
0	-0.240857	-0.787657	-1.281573	-1.145978
1	-0.987573	-0.167384	0.040363	-0.317959
2	-0.738668	-0.861381	-1.105315	0.510060
3	-0.738668	-0.407341	0.304750	-0.317959
4	-0.738668	-0.389974	0.216621	-0.317959

13. Simple Linear Regression

To predict the Credit amount with the help of Age

In [43]:
```
import statsmodels.api as sm

X = sm.add_constant(gc['Age'])
X.head()
```

Out[43]:

	const	Age
0	1.0	21
1	1.0	36
2	1.0	23
3	1.0	39
4	1.0	38

In [44]:
```
# BoxOfficeCollection (Y) - Outcome/ dependent Variable(Y)

Y = gc['Credit_amount']
Y.head()
```

Out[44]:
```
0    1049
1    2799
2     841
3    2122
4    2171
Name: Credit_amount, dtype: int64
```

In [45]:
```
from sklearn.model_selection import train_test_split

trainX, testX, trainY, testY = train_test_split(X , Y, train_size = 0.8, random_state = 100)  # Split the dataset into training (
print("Input attributes X - Train dataset :", trainX.shape)
print("Input attributes X - Test dataset  :", testX.shape)
print("Output attribute y - Train dataset :", trainY.shape)
print("Output attribute y - Test dataset  :", testY.shape)
```

```
Input attributes X - Train dataset : (800, 2)
Input attributes X - Test dataset  : (200, 2)
Output attribute y - Train dataset : (800,)
Output attribute y - Test dataset  : (200,)
```

In [46]:
```
#  Build the model on training dataset  - yhat = a + bx - OLS - Ordinary Least Squares Principle - statsmodels package

import statsmodels.api as sm

slm = sm.OLS(trainY, trainX).fit()
```

In [47]:
```
# Print estimated parameters and interpret the results

print(slm.params)
```

```
const    3067.965162
Age         9.225568
dtype: float64
```

Interpretation

- Estimated equation of straight line = a + bx
- Credit_amount = 3067.965162 + 9.225568 x

In [48]:
```
#  Model Diagnostics

slm.summary2()
```

Out[48]:

Model:	OLS	Adj. R-squared:	-0.000
Dependent Variable:	Credit_amount	AIC:	15073.6256
Date:	2022-12-29 21:13	BIC:	15082.9948
No. Observations:	800	Log-Likelihood:	-7534.8
Df Model:	1	F-statistic:	0.9754
Df Residuals:	798	Prob (F-statistic):	0.324
R-squared:	0.001	Scale:	8.9009e+06

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	3067.9652	346.7410	8.8480	0.0000	2387.3329	3748.5974
Age	9.2256	9.3413	0.9876	0.3236	-9.1108	27.5620

Omnibus:	294.057	Durbin-Watson:	2.018
Prob(Omnibus):	0.000	Jarque-Bera (JB):	882.767
Skew:	1.849	Prob(JB):	0.000
Kurtosis:	6.580	Condition No.:	122

Interpretation

- Null Hypothesis : H0: $\boldsymbol{\beta}1 = 0$ (There lies no linear relationship between Credit_amount & Age)
- Alternate Hypothesis : H0: $\boldsymbol{\beta}1 \neq 0$
- Independent Variable : Age
- Dependent Variable : Credit_amount
- R Squared : 0.001 (In 0.1% of the cases Credit_amount is explained by Age
- Since p(F-test) = 0.324 > 0.05, do not reject Ho
- Interpretation - There lies no linear relationship between Credit_amount & Age.

14. Multiple Linear Regression.

To predict the Credit amount with the help of Age, Duration of Current Employment, Duration of Credit

In [52]:
```
from sklearn.model_selection import train_test_split

X = gc2[['Duration_of_credit','Age','Duration_of_current_employment']]   # Input/ independent features
y = gc['Credit_amount']                                                 # Dependent feature
```

In [53]:
```
# Split the data into X_train, X_test, y_train, y_test with train_size=0.7/ test_size = 0.30

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
```

```
Dimensions of USAhousing dataset including Address feature  : (1000, 21)
Input attributes X - Train dataset : (700, 3)
Input attributes X - Test dataset  : (300, 3)
Output attribute y - Train dataset : (700,)
Output attribute y - Test dataset  : (300,)
```

In [54]:
```
mlr = LinearRegression().fit(Xtrain,ytrain)

# Model the equation - Evaluate the model by checking out it's coefficients and interpret them
print("Intercept is                     :",mlr.intercept_)       # intercept_ gives Y-intercept value
print("Coefficients of the input features \n",mlr.coef_)
```

```
Intercept is                     : 3359.705074584242
Coefficients of the input features 
 [1905.8935267   177.98063702 -217.19343076]
```

Interpretation -

- Credit_amount = 105.4 + 158.1(Duration_of_credit)+ 15.6(Age) + -179.8(Length_of_current_employment)

- Holding all other features fixed, a 1 unit increase in Duration_of_credit is associated with an increase in Credit_amount by 158.12. *Holding all other features fixed,a* 1 *unit increase in Ageis associated with an increase of Credit amount by* 158.12. Holding all other features fixed, a 1 unit increase in Age is associated with an increase of Credit amount by 15.6
- Holding all other features fixed, a 1 unit increase in Length_of_current_employment is associated with an decrease of $-179.8 in Credit_amount

In [56]:
```
# Predict the data on test dataset - Validate the model developed using train dataset

predict_test = mlr.predict(Xtest)                          # Predict for test dataset
df = pd.DataFrame({'Actual': ytest, 'Predicted': predict_test})   # Display Actual against Predicted values
df.head()
```

Out[56]:

	Actual	Predicted
249	5248	3294.433394
353	3499	1918.329010
537	1455	677.365977
424	1829	2299.684811
564	4272	4717.593586

In [57]:
```
print("Accuracy of the model - R square                = ",mlr.score(Xtest,ytest))   # Check R2 value for the model

# Evaluate the performance of algorithm/ model developed
print('Maximum error between original & predicted data = ', max_error(ytest, predict_test))
print('Mean Absolute Error (MAE)                       = ', mean_absolute_error(ytest, predict_test))
print('Mean squared error (MSE)                        = ', mean_squared_error(ytest, predict_test))
print('Root mean squared error (RMSE)                  = ', np.sqrt(mean_squared_error(ytest, predict_test)))
```

```
Accuracy of the model - R square                =  0.3218779561916397
Maximum error between original & predicted data =  6616.24228603882
Mean Absolute Error (MAE)                       =  1344.3807742216363
Mean squared error (MSE)                        =  3096274.6376442034
Root mean squared error (RMSE)                  =  1759.6234363193175
```

Interpretation

- From the model developed, R2 value demostrates that in 32 percent of the cases Credit_amount is explained by the input attributes ('Duration_of_credit','Age','Length_of_current_employment')
- This means algorithm is not very accurate but can still make reasonably good predictions because of the huge error in predicting results

14.1. Cross Validation for Multiple Linear Regression

In [58]:
```
# Cross validation

from sklearn.model_selection import cross_val_score

r2 = cross_val_score(mlr,X,y,scoring='r2',cv=5)   # store 5 scores of r2 in the object r2
r2
```

Out[58]:
```
array([0.24556101, 0.50617024, 0.44310278, 0.369878  , 0.30713812])
```

Interpretation

In [59]:
```
# Hyperparameter Tuning Using Grid Search Cross-Validation

len(X.columns)                                                    # number of features in X
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)  # step 1: Create a cross-validation scheme
hyper_params = [{'n_features_to_select': list(range(1, 4))}]     # step 2: Specify range of hyperparameters to tune
mlr = LinearRegression()                                          # step 3: Perform grid search - specify model
mlr.fit(X, y)
rfe = RFE(mlr)

# Call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe, param_grid = hyper_params, scoring= 'r2', cv = 5)
model_cv.fit(X, y)                                # fit the model
print("Best Number of features to select:",model_cv.best_params_)
print("Best R2 score:",model_cv.best_score_)
```

```
Best Number of features to select: {'n_features_to_select': 3}
Best R2 score: 0.3743700288839701
```

Interpretation:

- Max Score: 50%
- Min Score: 24%
- Best Number of features: 3
- Best R square 37%

15. Logistic Linear Regression

To predict creditability with the help of Age, Duration of Credit, Credit Amount and Duration of Current Employment.

```
X = gc[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/ independent features
y = gc1['Creditability_1']                                                               # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
```

```
Dimensions of USAhousing dataset including Address feature  : (1000, 21)
Input attributes X - Train dataset : (700, 4)
Input attributes X - Test dataset  : (300, 4)
Output attribute y - Train dataset : (700,)
Output attribute y - Test dataset  : (300,)
```

```
In [62]: model = LogisticRegression() # Build a Logisitc Regression Model
         logmodel=model.fit(Xtrain,ytrain)
         blr_predict = logmodel.predict(Xtest)              # Predict for test data values
         blr_predict

         print("Classification Report\n",classification_report(ytest,blr_predict))   # Evaluation Metrics
         print("Confusion Matrix\n",confusion_matrix(ytest,blr_predict))
         print("Model Accuracy is:",accuracy_score(ytest,blr_predict))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.50      0.16      0.24        82
           1       0.75      0.94      0.83       218

    accuracy                           0.73       300
   macro avg       0.62      0.55      0.54       300
weighted avg       0.68      0.73      0.67       300

Confusion Matrix
 [[ 13  69]
 [ 13 205]]
Model Accuracy is: 0.7266666666666667
```

Interpretation:

Accuracy: 72%

16. KNN Classification model with Hyperparameter tuning.

To predict creditability with the help of Age, Duration of Credit, Credit Amount and Duration of Current Employment.

kNN Classification Model

In [63]:
```
gc.columns
```

Out[63]:
```
Index(['Creditability', 'Account_status', 'Duration_of_credit',
       'Payment_status_of_previous_credit', 'Purpose', 'Credit_amount',
       'Value_savings_stocks', 'Duration_of_current_employment',
       'Instalment_percent', 'Marital_status_gender', 'Guarantors',
       'Duration_in_current_address', 'Property', 'Age', 'Concurrent_credits',
       'Housing', 'No_of_credits_at_this_bank', 'Occupation',
       'No_of_dependents', 'Telephone', 'Foreign_worker'],
      dtype='object')
```

In [64]:
```
# Datasets       - Explanatory vs. Target features
# Split Datasets - train vs. test datasets


X = gc2[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/ independent features
y = gc1['Creditability_1']                                                  # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
```

```
Dimensions of USAhousing dataset including Address feature  : (1000, 21)
Input attributes X - Train dataset : (700, 4)
Input attributes X - Test dataset  : (300, 4)
Output attribute y - Train dataset : (700,)
Output attribute y - Test dataset  : (300,)
```

In [65]:
```
# KNN algorithm - Build a model that will be used to predict patients

knn       = KNeighborsClassifier(n_neighbors=5)             # Create KNN Object
knn_model = knn.fit(Xtrain, ytrain)            # Train the model

knn_pred = knn_model.predict(Xtest)            # Predict test data set

print("Accuracy score :",roc_auc_score(ytest, knn_pred))              # Check performance of model with ROC Score
print("Model Accuracy is:",accuracy_score(ytest,knn_pred))
print("\nClassification Report\n",classification_report(ytest, knn_pred)) # Check performance with classification report
```

```
Accuracy score : 0.5584582680689193
Model Accuracy is: 0.69

Classification Report
               precision    recall  f1-score   support

           0       0.40      0.27      0.32        82
           1       0.76      0.85      0.80       218

    accuracy                           0.69       300
   macro avg       0.58      0.56      0.56       300
weighted avg       0.66      0.69      0.67       300
```

In [66]:
```
knn_model.get_params()
```

Out[66]:
```
{'algorithm': 'auto',
 'leaf_size': 30,
 'metric': 'minkowski',
 'metric_params': None,
 'n_jobs': None,
 'n_neighbors': 5,
 'p': 2,
 'weights': 'uniform'}
```

Interpretation

```
In [67]: # Use Hyperparameter Tuning to Improve Model Performance - Because the performance of the model is Low

         leaf_size   = list(range(1,50))    # List Hyperparameters that we want to tune
         n_neighbors = list(range(1,30))    # Number of neighbors to use
         p=[1,2]                            # Power parameter for the Minkowski metric. p=1 for manhattan_distance; & p=2 for euclidean_d

         hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)  # Convert to dictionary

         knn_2 = KNeighborsClassifier()                          # Create new KNN object

         clf = GridSearchCV(knn_2, hyperparameters, cv=10)       # Use GridSearch

         best_model = clf.fit(X,y)                               # Fit the model

         # Print The value of best Hyperparameters
         print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
         print('Best p:', best_model.best_estimator_.get_params()['p'])
         print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])

         Best leaf_size: 1
         Best p: 2
         Best n_neighbors: 22

In [68]: knn2_pred = best_model.predict(Xtest)                                       # Predict test data set

         print("Accuracy score :",roc_auc_score(ytest, knn2_pred))                   # Check performance of model with ROC Score
         print("Model Accuracy is:",accuracy_score(ytest,knn2_pred))
         print("\nClassification Report\n",classification_report(ytest, knn2_pred)) # Check performance with classification report

         Accuracy score : 0.5540389348847617
         Model Accuracy is: 0.7333333333333333

         Classification Report
                       precision    recall  f1-score   support

                    0       0.54      0.16      0.25        82
                    1       0.75      0.95      0.84       218

             accuracy                           0.73       300
            macro avg       0.65      0.55      0.54       300
         weighted avg       0.69      0.73      0.68       300
```

Interpretation:

- Accuracy before Hyperparameter tuning 69%
- Tuning the parameters leaf_size, n_neighbours, p and cv.
- Accuracy increased from 69% to 73%

17. Decision Tree Classification mode with Hyperparameter tuning.

To predict creditability with the help of Age, Duration of Credit, Credit Amount and Duration of Current Employment.

```
In [70]: dtree = DecisionTreeClassifier().fit(Xtrain,ytrain)

         dt_predict = dtree.predict(Xtest)
         dt_predict

         print("Classification Report\n",classification_report(ytest,dt_predict))
         print("Confusion Matrix\n",confusion_matrix(ytest,dt_predict))
         print("Model Accuracy is:",accuracy_score(ytest,dt_predict))

         Classification Report
                       precision    recall  f1-score   support

                    0       0.28      0.28      0.28        82
                    1       0.73      0.72      0.73       218

             accuracy                           0.60       300
            macro avg       0.50      0.50      0.50       300
         weighted avg       0.60      0.60      0.60       300

         Confusion Matrix
          [[ 23  59]
          [ 60 158]]
         Model Accuracy is: 0.6033333333333334

In [71]: # hyper parameter tuning
         parameter={
             "criterion":["gini","entropy","log_loss"],
             "splitter":["best","random"],
             "max_depth":[1,2,3,4,5,6,7,8],
             "max_features":["auto","sqrt","log2"]
         }
         gscv=GridSearchCV(dtree,param_grid=parameter,cv=5,scoring="accuracy").fit(Xtrain,ytrain)
         gscv.best_params_

Out[71]: {'criterion': 'entropy',
          'max_depth': 6,
          'max_features': 'auto',
          'splitter': 'random'}

In [72]: dt_predict = gscv.predict(Xtest)
         print("Classification Report\n",classification_report(ytest,dt_predict))
         print("Confusion Matrix\n",confusion_matrix(ytest,dt_predict))
         print("Model Accuracy is:",accuracy_score(ytest,dt_predict))

         Classification Report
                       precision    recall  f1-score   support

                    0       0.35      0.10      0.15        82
                    1       0.73      0.93      0.82       218

             accuracy                           0.70       300
            macro avg       0.54      0.51      0.49       300
         weighted avg       0.63      0.70      0.64       300

         Confusion Matrix
          [[  8  74]
          [ 15 203]]
         Model Accuracy is: 0.7033333333333334
```

Interpretation:

- Accuracy before hyperparameter tuning: 60%.
- Hypermeters tuning done with criterion, splitter, max_dept, max_features.
- After Hyperparameter tuning the accuracy increases to 72%.

18. Random Forest Classification model with Hyperparameter tuning:

To predict creditability with the help of Age, Duration of Credit, Credit Amount and Duration of Current Employment.

```
In [74]: rfc = RandomForestClassifier(n_estimators=100).fit(Xtrain,ytrain)

         rf_predict = rfc.predict(Xtest)
         rf_predict

         print("Classification Report\n",classification_report(ytest,rf_predict))
         print("Confusion Matrix\n",confusion_matrix(ytest,rf_predict))
         print("Model Accuracy is:",accuracy_score(ytest,rf_predict))

         Classification Report
                        precision    recall  f1-score   support

                    0       0.34      0.26      0.29        82
                    1       0.74      0.81      0.78       218

             accuracy                           0.66       300
            macro avg       0.54      0.53      0.53       300
         weighted avg       0.63      0.66      0.64       300

         Confusion Matrix
          [[ 21  61]
          [ 41 177]]
         Model Accuracy is: 0.66
```

```
In [75]: # hyper parameter tuning
         parameter={
             "criterion":["gini","entropy","log_loss"],
             "max_depth":[1,2,3,4,5,6,7,8],
             "max_features":["auto","sqrt","log2"],
             'n_estimators':[50,100,150,200,250]
         }
         gscv=GridSearchCV(rfc,param_grid=parameter,cv=5,scoring="accuracy").fit(Xtrain,ytrain)
         gscv.best_params_

Out[75]: {'criterion': 'gini',
          'max_depth': 5,
          'max_features': 'auto',
          'n_estimators': 100}
```

```
In [76]: rfc_predict = gscv.predict(Xtest)
         print("Classification Report\n",classification_report(ytest,rfc_predict))
         print("Confusion Matrix\n",confusion_matrix(ytest,rfc_predict))
         print("Model Accuracy is:",accuracy_score(ytest,rfc_predict))

         Classification Report
                        precision    recall  f1-score   support

                    0       0.60      0.11      0.19        82
                    1       0.74      0.97      0.84       218

             accuracy                           0.74       300
            macro avg       0.67      0.54      0.51       300
         weighted avg       0.70      0.74      0.66       300

         Confusion Matrix
          [[  9  73]
          [  6 212]]
         Model Accuracy is: 0.7366666666666667
```

Interpretation

- Normal decision tree model gave the accuracy of 68%
- Hyperparameter tuning of Decision tree with criterion,n_estimator,max_depth,max_features increases the accuracy to 73%

Interpretation:

- Normal decision tree model gave the accuracy of 68%
- Hyperparameter tuning of Decision tree with criterion, n_estimator, max_depth, max_features increases the accuracy to 73%

19. AdaBoost Classification Model:

To predict creditability with the help of Age, Duration of Credit, Credit Amount and Duration of Current Employment.

```
In [78]: ab=AdaBoostClassifier(n_estimators=100)                              # Build the model
         ab.fit(Xtrain,ytrain)                    # Fit the model on training dataset X & y attributes

         ab_predict = ab.predict(Xtest)
         ab_predict

         print("Classification Report\n",classification_report(ytest,ab_predict))
         print("Confusion Matrix\n",confusion_matrix(ytest,ab_predict))
         print("Model Accuracy is:",accuracy_score(ytest,ab_predict))
```

```
Classification Report
               precision    recall  f1-score   support

           0       0.47      0.22      0.30        82
           1       0.76      0.91      0.82       218

    accuracy                           0.72       300
   macro avg       0.61      0.56      0.56       300
weighted avg       0.68      0.72      0.68       300

Confusion Matrix
 [[ 18  64]
 [ 20 198]]
Model Accuracy is: 0.72
```

Interpretation:

Accuracy: 72%

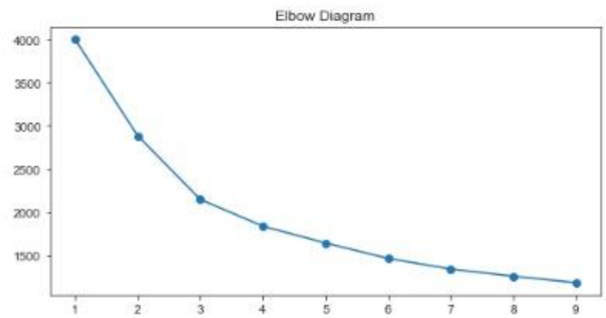
20. K-Means Clustering Model

Clustering the customers of the German bank based on Age, Credit Amount, Duration of Credit, Duration of Current Employment.

In [79]:
```
# Elbow Method - variance explained by the clusters is plotted against the no. of clusters

cluster_error = []
cluster_range = range(1,10)
for i in cluster_range:
    clust = KMeans(i).fit(gc2)
    cluster_error.append(clust.inertia_)  # inertia_ in python - wcss
```

In [80]:
```
# Plot - Elbow method outcome

plt.figure(figsize=(8,4))
plt.plot(cluster_range, cluster_error, marker = "o")
plt.title("Elbow Diagram")
```

Out[80]:
```
Text(0.5, 1.0, 'Elbow Diagram')
```



In [85]:
```

h_clusters = AgglomerativeClustering(n_clusters=3).fit(gc2)
gc["h_clusterid"] = h_clusters.labels_
gc[0:5]
```

Out[85]:

	Creditability	Account_status	Duration_of_credit	Payment_status_of_previous_credit	Purpose	Credit_amount	Value_savings_stocks	Duration_of_current_empl
0	1	1	18	4	2	1049	1	
1	1	1	9	4	0	2799	1	
2	1	2	12	2	9	841	2	
3	1	1	12	4	0	2122	1	
4	1	1	12	4	0	2171	1	

5 rows × 23 columns

In [86]:
```
# Print dimensions of each new cluster created

print("Cluster 0 Dimensions: \t",gc[gc.h_clusterid==0].shape)
print("Cluster 1 Dimensions: \t",gc[gc.h_clusterid==1].shape)
print("Cluster 2 Dimensions: \t",gc[gc.h_clusterid==2].shape)
```

```
Cluster 0 Dimensions:    (499, 23)
Cluster 1 Dimensions:    (227, 23)
Cluster 2 Dimensions:    (274, 23)
```

In [87]:
```
accuracy=silhouette_score(gc2,h_clusters.labels_)
print(accuracy)
```

```
0.26004629457346506
```

Interpretation:

- Elbow point is at 3
- Indicates that there might be 3 clusters existing in the dataset
- Cluster 0 (316 instances)
- Cluster 1 (173 instances)
- Cluster 2 (511 instances)

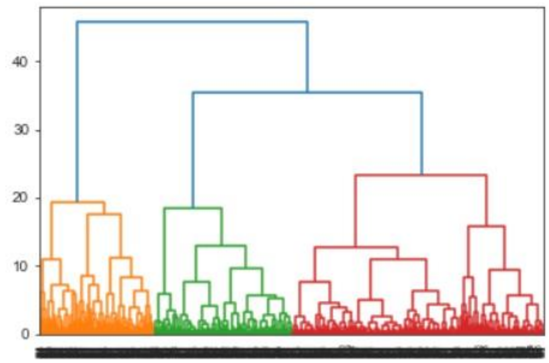
- Accuracy = 26%

21. Hierarchical Clustering

Clustering the customers of the German bank based on Age, Credit Amount, Duration of Credit, Duration of Current Employment.

Hierarchical clustering - Store cluster labels in varible h_clusterid

In [84]:
```
dendrogram = sch.dendrogram(sch.linkage(gc2,method="ward"))
```



In [81]:
```
# Set K=3 & run KMeans algorithm - Create a new column clusterid to capture the cluster number it is assigned to

clust2 = KMeans(n_clusters=3, random_state = 42).fit(gc2)
gc["clusterid"] = clust2.labels_
gc.head()
```

Out[81]:

	Creditability	Account_status	Duration_of_credit	Payment_status_of_previous_credit	Purpose	Credit_amount	Value_savings_stocks	Duration_of_current_empl
0	1	1	18	4	2	1049	1	
1	1	1	9	4	0	2799	1	
2	1	2	12	2	9	841	2	
3	1	1	12	4	0	2122	1	
4	1	1	12	4	0	2171	1	

5 rows × 22 columns

In [82]:
```
accuracy=silhouette_score(gc2,clust2.labels_)
print(accuracy)
```

```
0.30453420003943077
```

In [83]:
```
# Print dimensions of each new cluster created

print("Cluster 0 Dimensions: \t",gc[gc.clusterid==0].shape)
print("Cluster 1 Dimensions: \t",gc[gc.clusterid==1].shape)
print("Cluster 2 Dimensions: \t",gc[gc.clusterid==2].shape)
```

```
Cluster 0 Dimensions:    (173, 22)
Cluster 1 Dimensions:    (511, 22)
Cluster 2 Dimensions:    (316, 22)
```

Interpretation:

- Here with the help of Dendrogram we can say that the optimal number of clusters can be 3.
- Cluster 0 (499 instances)
- Cluster 1 (227 instances)
- Cluster 2 (274 instances)
- Accuracy = 26%

CONCLUSION.

- Out of 1000 records 700 are Good Credit risk and 300 are Bad Credit risk. Therefore, it is an unbalanced data.
- Foreign workers have Good Credit risk.
- There is association between Guarantors and Creditability.
- Majority of the people are from the age group of 25 to 30
- Majority of the people have taken the credit amount between 0 to 5000
- Majority duration of the credit is 15 to 25 months
- There is weak association between Age and Credit Amount.
- Maximum duration of repaying for Good Credit risk is 40 months and average is 15 months.
- More than 40 months has Bad credit risk
- Moderate positive correlation between Duration of credit and Credit Amount.
- Weak Positive correlation between Age and Duration of current employment and Age
- There is no linear relationship between Age and Credit amount
- Age, Duration of current Employment and Duration of credit cannot predict Credit amount accurately as the Accuracy is only 32%
- A predictive model is developed on this data to provide a bank manager guidance for making a decision whether to approve a loan to a prospective applicant based on his/her profiles. Adaboost model gives highest accuracy of 72% without Hyper tuning whereas Decision tree and Random Forest models gives an accuracy of 73% with Hyper tuning
- Segmentation with the help of clustering models didn't gave the desired results and proper clusters as the accuracy is a low as 26%.

SAMPLE CODE

```
import warnings                          # Supressing the warning messages
warnings.filterwarnings('ignore')
import pandas          as pd
import numpy           as np
import scipy           as sp
import matplotlib.pyplot as plt
import seaborn         as sns
import statsmodels.api   as sm

from scipy                          import stats
from scipy.cluster                     import hierarchy as sch
from sklearn.preprocessing               import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.model_selection              import train_test_split, KFold, GridSearchCV
from sklearn.feature_selection            import RFE

from sklearn.linear_model                import LinearRegression, LogisticRegression
from sklearn.neighbors                 import KNeighborsClassifier
from sklearn.tree                     import DecisionTreeClassifier
from sklearn.ensemble                  import RandomForestClassifier
from sklearn.ensemble                  import AdaBoostClassifier
from sklearn.cluster                  import KMeans, AgglomerativeClustering

from sklearn.metrics                   import r2_score, mean_squared_error, max_error,
      mean_absolute_error
from statsmodels.sandbox.regression.predstd import wls_prediction_std
from sklearn.metrics                   import roc_auc_score, classification_report, confusion_matrix,
      accuracy_score, silhouette_score

# Read the dataset - German_Credit

gc = pd.read_excel(r"C:\Users\user\Desktop\coding\ML\ML
      Models\Regression\2022_ICSSR_ExcelWB_DataAnalysis.xls", sheet_name = 'German_Credit')
gc.info()

# Frequency Distribution - value_counts()

#gc['Creditability'].value_counts()
gc.Creditability.value_counts()
# Creating Bar chart as the Target variable (Creditability) is Categorical

GroupedData = gc.groupby('Creditability').size()
print(GroupedData)
GroupedData.plot(kind='bar', figsize=(4,3))
# same as above can be done using value_counts()
groupeddata1=gc.Creditability.value_counts()
groupeddata1.plot(kind='bar', figsize=(4,3))
# Plotting histograms of multiple columns together

gc.hist(['Age', 'Credit_amount','Duration_of_credit'], figsize=(15,8))
```

```
plt.show()
gc.hist(['Age'], figsize=(15,8))
plt.show()
# Contingency Table - crosstable()

CT = pd.crosstab(index=gc.Creditability, columns=[gc.Foreign_worker,gc.Guarantors])
print(CT)
# checking whether there is relation between creditability and guarantors using chi_square test of
    independence
# H0 = There is no relation(Independent)
# H1 = There is relation(Dependent)
from scipy.stats import chi2_contingency
CT = pd.crosstab(gc.Creditability, gc['Guarantors'])
print(CT)
stat, p, dof, expected = chi2_contingency(CT)

alpha = 0.05
print("p value is " + str(p))
if p <= alpha:
   print('Dependent (reject H0)')
else:
   print('Independent (H0 holds true)')
# chi_ square test of independence gives 4 values which are - Chi-square, P-value, DOF and expected
    values
from scipy.stats import chi2_contingency
CT = pd.crosstab(gc.Creditability, gc['Guarantors'])
print(CT)
chi2_contingency(CT)
gc['Creditability']                  = gc['Creditability'].astype('object')
gc['Account_status']                 = gc['Account_status'].astype('object')
gc['Payment_status_of_previous_credit'] = gc['Payment_status_of_previous_credit'].astype('object')
gc['Purpose']                        = gc['Purpose'].astype('object')
gc['Value_savings_stocks']           = gc['Value_savings_stocks'].astype('object')
gc['Marital_status_gender']          = gc['Marital_status_gender'].astype('object')
gc['Guarantors']                     = gc['Guarantors'].astype('object')
gc['Property']                       = gc['Property'].astype('object')
gc['Concurrent_credits']             = gc['Concurrent_credits'].astype('object')
gc['Housing']                        = gc['Housing'].astype('object')
gc['Occupation']                     = gc['Occupation'].astype('object')
gc['Telephone']                      = gc['Telephone'].astype('object')
gc.Foreign_worker                    = gc.Foreign_worker.astype('object')
gc["Instalment_percent"]                   = gc['Instalment_percent'].astype('object')
gc['Duration_in_current_address']                = gc['Duration_in_current_address'].astype('object')
gc['No_of_credits_at_this_bank']                 = gc['No_of_credits_at_this_bank'].astype('object')
gc['No_of_dependents']                     = gc['No_of_dependents'].astype('object')
gc1= pd.get_dummies(gc, columns=['Creditability', 'Account_status',
    'Payment_status_of_previous_credit', 'Purpose',
    'Value_savings_stocks', 'Duration_of_current_employment',
    'Instalment_percent', 'Marital_status_gender', 'Guarantors',
    'Duration_in_current_address', 'Property',  'Concurrent_credits',
    'Housing', 'No_of_credits_at_this_bank', 'Occupation',
```

```
    'No_of_dependents', 'Telephone', 'Foreign_worker'],drop_first=True)
gc1.head()
# Standard Scaling (Standardization)
gc2=gc[['Duration_of_credit','Credit_amount','Age','Duration_of_current_employment']]
standardizer = StandardScaler()
gc2 = pd.DataFrame(standardizer.fit_transform(gc2))
gc2.columns=[  'Duration_of_credit',
             'Credit_amount',
             'Age',
             'Duration_of_current_employment']
gc2.head()
# Check the number of unique values in each column

gc.nunique()
gc.duplicated().sum()
# Check for missing values in the dataset

gc.isnull().sum()                                  # Count of missing values
#round(gc.isnull().sum() / gc.isnull().count() * 100, 2)    # Percentage of missing values
gc.isna().sum()
# select columns with numerical data types

num_df = gc.select_dtypes(include=['int64', 'float64']).columns
subset = gc[num_df]
# create a histogram plot for each numeric variable

ax = subset.hist()
for axis in ax.flatten():                # disable axis labels to avoid the clutter
   axis.set_xticklabels([])
   axis.set_yticklabels([])
plt.figure(figsize=(15,8))
plt.show()
# countplot() - Plot Categorical feature

sns.countplot(x = "Creditability", data = gc)
sns.countplot(gc.Creditability)
# distplot() function - Plot Numerical continuous feature

sns.distplot(gc.Credit_amount)
#Association between Age vs. Credit_amount
np.corrcoef(gc.Credit_amount,gc.Age)
# lmplot() - Plot Numeric vs. Numeric features
# Age vs. Credit_amount

sns.lmplot(x = 'Age', y = 'Credit_amount', data = gc)
# boxplot() function - Plot Numeric vs. Categorical features
# Creditability vs Duration of credit
sns.boxplot(x = 'Creditability', y = 'Duration_of_credit', data = gc, palette="PRGn")
# Measures of central tendency     - mean(), median(), mode()
# Measures of dispersion           - std(), var()
# Measures of skewness & kurtosis  - skew(), kurtosis()
```

```
# Done for Age
print ('Mean              :', sp.mean(gc.Age))
print ('Median            :', sp.median(gc.Age))
print ('Mode              :', sp.stats.mode(gc.Age))
print ('Standard Deviation :', sp.std(gc.Age))
print ('Variance          :', sp.var(gc.Age))
print ('Skewness          :', sp.stats.skew(gc.Age))
print ('Kurtosis          :', sp.stats.kurtosis(gc.Age))
# Describes only numeric values

gc.describe()
gc.describe().T
#gc.describe().T

# EDA for entire data

gc.describe(include='all')
# Diagnostic Analysis - corr() - Create correlation matrix - Karl Pearson's Coefficient of Correlation (r)

gc.corr()
# scatter() - Graphically plot 2 variables/ features (Age vs. Credit_amount)

plt.figure(figsize=(8,4))                    # Change image size
sns.set_style('darkgrid')                    # Set background

plt.scatter(x = gc.Age,y = gc.Credit_amount,color='crimson') # Create scatterplot

plt.title('Scatter Plot',fontsize = 20)          # Title of the chart
plt.xlabel('Age',fontsize = 12)       # Name x-axis
plt.ylabel('Credit_amount',fontsize = 12)          # Name y-axis
plt.show()
# 3. scatter_matrix() - diagonal - kde  = kernel density estimation - hist = histogram

pd.plotting.scatter_matrix(gc, figsize=(16, 16), diagonal='hist')
plt.show()
gc.Creditability.unique
# 4. Pairplot() - Measures association between multiple features

sns.pairplot(gc)
# 5. heatmap() - Measures association between muliptle features

plt.figure(figsize=(15,8))
sns.set_style('ticks')

sns.heatmap(gc.corr(), annot=True)

plt.title('German Credit Data')
plt.show()
sns.heatmap(gc.corr())
gc.corr().style.background_gradient(cmap='coolwarm')
gc.columns
```

```
import statsmodels.api as sm

X = sm.add_constant(gc['Age'])
X.head()
# BoxOfficeCollection (Y) - Outcome/ dependent Variable(Y)

Y = gc['Credit_amount']
Y.head()
from sklearn.model_selection import train_test_split

trainX, testX, trainY, testY = train_test_split(X , Y, train_size = 0.8, random_state = 100)  # Split the dataset
    into training (80) and validation (20) sub-sets
print("Input attributes X - Train dataset :", trainX.shape)
print("Input attributes X - Test dataset  :", testX.shape)
print("Output attribute y - Train dataset :", trainY.shape)
print("Output attribute y - Test dataset  :", testY.shape)
#  Build the model on training dataset  - yhat = a + bx - OLS - Ordinary Least Squares Principle -
    statsmodels package

import statsmodels.api as sm

slm = sm.OLS(trainY, trainX).fit()
# Print estimated parameters and interpret the results

print(slm.params)
#  Model Diagnostics

slm.summary2()
sns.lmplot(x = 'Age', y = 'Credit_amount', data = gc)
gc.columns
# Selecting final predictors for Machine Learning

gc2.head()
from sklearn.model_selection import train_test_split

X = gc2[['Duration_of_credit','Age','Duration_of_current_employment']]   # Input/ independent features
y = gc['Credit_amount']
# Split the data into X_train, X_test, y_train, y_test with train_size=0.7/ test_size = 0.30

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
mlr = LinearRegression().fit(Xtrain,ytrain)

# Model the equation - Evaluate the model by checking out it's coefficients and interpret them
print("Intercept is                    :",mlr.intercept_)       # intercept_ gives Y-intercept value
print("Coefficients of the input features \n",mlr.coef_)
mlr.get_params()
```

```
# Predict the data on test dataset - Validate the model developed using train dataset

predict_test = mlr.predict(Xtest)                          # Predict for test dataset
df = pd.DataFrame({'Actual': ytest, 'Predicted': predict_test})   # Display Actual against Predicted values
df.head()
print("Accuracy of the model - R square             = ",mlr.score(Xtest,ytest))   # Check R2 value for the
      model

# Evaluate the performance of algorithm/ model developed
print('Maximum error between original & predicted data = ', max_error(ytest, predict_test))
print('Mean Absolute Error (MAE)                   = ', mean_absolute_error(ytest, predict_test))
print('Mean squared error (MSE)                    = ', mean_squared_error(ytest, predict_test))
print('Root mean squared error (RMSE)               = ', np.sqrt(mean_squared_error(ytest, predict_test)))
# Cross validation

from sklearn.model_selection import cross_val_score

r2 = cross_val_score(mlr,X,y,scoring='r2',cv=5)   # store 5 scores of r2 in the object r2
      # Indicator of how good the model is
r2
# Hyperparameter Tuning Using Grid Search Cross-Validation

len(X.columns)                                    # number of features in X
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)  # step 1: Create a cross-validation scheme
hyper_params = [{'n_features_to_select': list(range(1, 4))}]     # step 2: Specify range of hyperparameters
      to tune
mlr = LinearRegression()                                # step 3: Perform grid search - specify model
mlr.fit(X, y)
rfe = RFE(mlr)

# Call GridSearchCV()
model_cv = GridSearchCV(estimator = rfe, param_grid = hyper_params, scoring= 'r2', cv = 5)
model_cv.fit(X, y)                          # fit the model
print("Best Number of features to select:",model_cv.best_params_)
print("Best R2 score:",model_cv.best_score_)
gc.corr().style.background_gradient(cmap='coolwarm')
# Datasets      - Explanatory vs. Target features
# Split Datasets - train vs. test datasets



X = gc[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/
      independent features
y = gc1['Creditability_1']                              # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
```

```
model = LogisticRegression() # Build a Logisitc Regression Model
logmodel=model.fit(Xtrain,ytrain)
blr_predict = logmodel.predict(Xtest)             # Predict for test data values
blr_predict

print("Classification Report\n",classification_report(ytest,blr_predict))   # Evaluation Metrics
print("Confusion Matrix\n",confusion_matrix(ytest,blr_predict))
print("Model Accuracy is:",accuracy_score(ytest,blr_predict))
# Datasets       - Explanatory vs. Target features
# Split Datasets - train vs. test datasets


X = gc2[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/
    independent features
y = gc1['Creditability_1']                                  # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
# KNN algorithm - Build a model that will be used to predict patients

knn       = KNeighborsClassifier(n_neighbors=5)           # Create KNN Object
knn_model = knn.fit(Xtrain, ytrain)           # Train the model

knn_pred = knn_model.predict(Xtest)           # Predict test data set

print("Accuracy score :",roc_auc_score(ytest, knn_pred))               # Check performance of model with
    ROC Score
print("Model Accuracy is:",accuracy_score(ytest,knn_pred))
print("\nClassification Report\n",classification_report(ytest, knn_pred)) # Check performance with
    classification report
knn_model.get_params()
# Use Hyperparameter Tuning to Improve Model Performance - Because the performance of the model is
    low

leaf_size   = list(range(1,50))    # List Hyperparameters that we want to tune
n_neighbors = list(range(1,30))    # Number of neighbors to use
p=[1,2]                        # Power parameter for the Minkowski metric. p=1 for manhattan_distance; & p=2
    for euclidean_distance

hyperparameters = dict(leaf_size=leaf_size, n_neighbors=n_neighbors, p=p)  # Convert to dictionary

knn_2 = KNeighborsClassifier()                        # Create new KNN object

clf = GridSearchCV(knn_2, hyperparameters, cv=10)        # Use GridSearch

best_model = clf.fit(X,y)                            # Fit the model
```

```
# Print The value of best Hyperparameters
print('Best leaf_size:', best_model.best_estimator_.get_params()['leaf_size'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
knn2_pred = best_model.predict(Xtest)                    # Predict test data set

print("Accuracy score :",roc_auc_score(ytest, knn2_pred))          # Check performance of model with
    ROC Score
print("Model Accuracy is:",accuracy_score(ytest,knn2_pred))
print("\nClassification Report\n",classification_report(ytest, knn2_pred)) # Check performance with
    classification report
# Datasets      - Explanatory vs. Target features
# Split Datasets - train vs. test datasets


X = gc2[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/
    independent features
y = gc1['Creditability_1']                          # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
dtree = DecisionTreeClassifier().fit(Xtrain,ytrain)

dt_predict = dtree.predict(Xtest)
dt_predict

print("Classification Report\n",classification_report(ytest,dt_predict))
print("Confusion Matrix\n",confusion_matrix(ytest,dt_predict))
print("Model Accuracy is:",accuracy_score(ytest,dt_predict))
# hyper parameter tuning
parameter={
  "criterion":["gini","entropy","log_loss"],
  "splitter":["best","random"],
  "max_depth":[1,2,3,4,5,6,7,8],
  "max_features":["auto","sqrt","log2"]
}
gscv=GridSearchCV(dtree,param_grid=parameter,cv=5,scoring="accuracy").fit(Xtrain,ytrain)
gscv.best_params_
dt_predict = gscv.predict(Xtest)
print("Classification Report\n",classification_report(ytest,dt_predict))
print("Confusion Matrix\n",confusion_matrix(ytest,dt_predict))
print("Model Accuracy is:",accuracy_score(ytest,dt_predict))
# Datasets      - Explanatory vs. Target features
# Split Datasets - train vs. test datasets
```

```
X = gc2[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/
      independent features
y = gc1['Creditability_1']                                    # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
rfc = RandomForestClassifier(n_estimators=100).fit(Xtrain,ytrain)

rf_predict = rfc.predict(Xtest)
rf_predict

print("Classification Report\n",classification_report(ytest,rf_predict))
print("Confusion Matrix\n",confusion_matrix(ytest,rf_predict))
print("Model Accuracy is:",accuracy_score(ytest,rf_predict))
# hyper parameter tuning
parameter={
   "criterion":["gini","entropy","log_loss"],
   "max_depth":[1,2,3,4,5,6,7,8],
   "max_features":["auto","sqrt","log2"],
   'n_estimators':[50,100,150,200,250]
}
gscv=GridSearchCV(rfc,param_grid=parameter,cv=5,scoring="accuracy").fit(Xtrain,ytrain)
gscv.best_params_
rfc_predict = gscv.predict(Xtest)
print("Classification Report\n",classification_report(ytest,rfc_predict))
print("Confusion Matrix\n",confusion_matrix(ytest,rfc_predict))
print("Model Accuracy is:",accuracy_score(ytest,rfc_predict))
# Datasets       - Explanatory vs. Target features
# Split Datasets - train vs. test datasets


X = gc2[['Duration_of_credit','Age','Duration_of_current_employment','Credit_amount']]   # Input/
      independent features
y = gc1['Creditability_1']                                    # Dependent feature

Xtrain, Xtest, ytrain, ytest = train_test_split(X, y, train_size=0.7, random_state=100) # Method 1
print("Dimensions of USAhousing dataset including Address feature  :", gc.shape)
print("Input attributes X - Train dataset :", Xtrain.shape)
print("Input attributes X - Test dataset  :", Xtest.shape)
print("Output attribute y - Train dataset :", ytrain.shape)
print("Output attribute y - Test dataset  :", ytest.shape)
ab=AdaBoostClassifier(n_estimators=100)                    # Build the model
ab.fit(Xtrain,ytrain)                # Fit the model on training dataset X & y attributes

ab_predict = ab.predict(Xtest)
ab_predict
```

```
print("Classification Report\n",classification_report(ytest,ab_predict))
print("Confusion Matrix\n",confusion_matrix(ytest,ab_predict))
print("Model Accuracy is:",accuracy_score(ytest,ab_predict))
# Elbow Method - variance explained by the clusters is plotted against the no. of clusters

cluster_error = []
cluster_range = range(1,10)
for i in cluster_range:
   clust = KMeans(i).fit(gc2)
   cluster_error.append(clust.inertia_)  # inertia_ in python – wcss
# Plot - Elbow method outcome

plt.figure(figsize=(8,4))
plt.plot(cluster_range, cluster_error, marker = "o")
plt.title("Elbow Diagram")
# Set K=3 & run KMeans algorithm - Create a new column clusterid to capture the cluster number it is
     assigned to

clust2 = KMeans(n_clusters=3, random_state = 42).fit(gc2)
gc["clusterid"] = clust2.labels_
gc.head()
accuracy=silhouette_score(gc2,clust2.labels_)
print(accuracy)
# Print dimensions of each new cluster created

print("Cluster 0 Dimensions: \t",gc[gc.clusterid==0].shape)
print("Cluster 1 Dimensions: \t",gc[gc.clusterid==1].shape)
print("Cluster 2 Dimensions: \t",gc[gc.clusterid==2].shape)
dendrogram = sch.dendrogram(sch.linkage(gc2,method="ward"))
h_clusters = AgglomerativeClustering(n_clusters=3).fit(gc2)
gc["h_clusterid"] = h_clusters.labels_
gc[0:5]
# Print dimensions of each new cluster created

print("Cluster 0 Dimensions: \t",gc[gc.h_clusterid==0].shape)
print("Cluster 1 Dimensions: \t",gc[gc.h_clusterid==1].shape)
print("Cluster 2 Dimensions: \t",gc[gc.h_clusterid==2].shape)
accuracy=silhouette_score(gc2,h_clusters.labels_)
print(accuracy)
```