

```
In [1]: import os
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

%matplotlib inline
sns.set()

import warnings
warnings.filterwarnings('ignore')
```

```
In [2]: os.getcwd()
```

```
Out[2]: 'C:\\\\Users\\\\bkohl\\\\OneDrive\\\\Desktop\\\\Priyanka\\\\2. Stats & ML\\\\July -23\\\\Lecture notes\\\\20-July-23 - Decision Tree\\\\21st July 2023'
```

```
In [3]: credit_dt = pd.read_csv('German Credit Dataset.csv')
```

```
In [4]: credit_dt.head()
```

```
Out[4]: checking_balance  months_loan_duration  credit_history          purpose  amount  savings_balance
0           < 0 DM                  6      critical  furniture/appliances    1169       unk
1        1 - 200 DM                 48        good  furniture/appliances    5951  < 10
2        unknown                  12      critical        education    2096  < 10
3           < 0 DM                 42        good  furniture/appliances    7882  < 10
4           < 0 DM                 24        poor            car    4870  < 10
```



- DM - Deutsche Mark(currency of West Germany)
- By analyzing this dataset we should be able to know if the person is a credit defaulter or not. The "default" is a dependent variable and others are independent variables

```
In [5]: credit_dt.columns
```

```
Out[5]: 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 [6]: credit_dt.shape
```

```
Out[6]: (1000, 17)
```

```
In [7]: credit_dt.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
```

Data Cleaning

In [8]: `credit_df = credit_dt.copy()`

In [9]: `credit_df.head()`

Out[9]:

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_bal
0	< 0 DM	6	critical	furniture/appliances	1169	unk
1	1 - 200 DM	48	good	furniture/appliances	5951	< 10
2	unknown	12	critical	education	2096	< 10
3	< 0 DM	42	good	furniture/appliances	7882	< 10
4	< 0 DM	24	poor	car	4870	< 10

In [10]: `# checking_balance, savings_balance, employment_duration`

In [11]: `credit_df['checking_balance'].value_counts().index`

Out[11]: `Index(['unknown', '< 0 DM', '1 - 200 DM', '> 200 DM'], dtype='object')`

In [12]:

```
data_correction = ['checking_balance', 'savings_balance', 'employment_duration']
def count_size():
    dict = {}
    for i in data_correction:
        x = credit_df[i].value_counts().index.size
        dict.update({i:x})
    return(dict)

# x = count_size()
# print(x, type(x))
```

```
In [13]: for i in data_correction:  
    x = count_size()  
    if x[i]==4:  
        credit_df[i]=np.where(credit_df[i]== credit_df[i].value_counts().index[0],  
                               np.where(credit_df[i]== credit_df[i].value_counts().index[1],  
                                       np.where(credit_df[i]== credit_df[i].value_counts().index[2],  
                                               np.where(credit_df[i]== credit_df[i].value_counts().index[3],  
                                                       credit_df[i]== credit_df[i].value_counts().index[4]))))  
    elif x[i]==5:  
        credit_df[i]=np.where(credit_df[i]== credit_df[i].value_counts().index[0],  
                               np.where(credit_df[i]== credit_df[i].value_counts().index[1],  
                                       np.where(credit_df[i]== credit_df[i].value_counts().index[2],  
                                               np.where(credit_df[i]== credit_df[i].value_counts().index[3],  
                                                       np.where(credit_df[i]== credit_df[i].value_counts().index[4],  
                                                               credit_df[i]== credit_df[i].value_counts().index[5])))
```

```
In [14]: credit_df.head()
```

```
Out[14]:
```

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

```
In [15]: for i in data_correction:  
    print("*****Original*****\n")  
    print('-----', i, '-----', '\n', credit_dt[i].value_counts(), '\n')  
    print("*****Corrected*****\n")  
    print('-----', i, '-----', '\n', credit_df[i].value_counts(), '\n')
```

*****Original*****

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

*****Corrected*****

```
----- checking_balance -----
0      394
1      274
2      269
3      63
Name: checking_balance, dtype: int64
```

*****Original*****

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

*****Corrected*****

```
----- savings_balance -----
0      603
1      183
2      103
3      63
4      48
Name: savings_balance, dtype: int64
```

*****Original*****

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

*****Corrected*****

```
----- employment_duration -----
0      339
1      253
2      174
3      172
4      62
Name: employment_duration, dtype: int64
```

Looking at above data we can conclude that the data replacement is correctly done.

```
In [16]: # purpose
```

```
In [17]: credit_df['purpose'].value_counts()
```

```
Out[17]: furniture/appliances    473  
car                  337  
business             97  
education            59  
renovations          22  
car0                 12  
Name: purpose, dtype: int64
```

```
In [18]: # Here we are replacing the "furniture/appliances" to "appliances"
```

```
In [19]: credit_df['purpose'] = np.where(credit_df['purpose'] == 'furniture/appliances', 'appliances', credit_df['purpose'])
```

```
In [20]: credit_df['purpose'].value_counts()
```

```
Out[20]: appliances    473  
car                  337  
business             97  
education            59  
renovations          22  
car0                 12  
Name: purpose, dtype: int64
```

```
In [21]: # Changes are reflecting.
```

```
In [22]: print(list(credit_dt.select_dtypes(include = 'object')))  
print(len(list(credit_dt.select_dtypes(include = 'object'))))
```

```
['checking_balance', 'credit_history', 'purpose', 'savings_balance', 'employment_duration', 'other_credit', 'housing', 'job', 'phone', 'default']  
10
```

```
In [23]: print(list(credit_df.select_dtypes(include = 'object')))  
print(len(list(credit_df.select_dtypes(include = 'object'))))
```

```
['credit_history', 'purpose', 'other_credit', 'housing', 'job', 'phone', 'default']  
7
```

```
In [24]: # before changes  
credit_dt.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
```

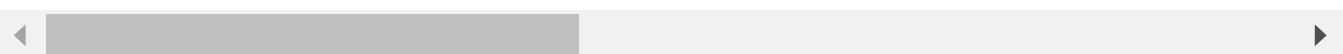
In [25]: `#after changes
credit_df.info()`

```
<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   int32  
 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   int32  
 6   employment_duration  1000 non-null   int32  
 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: int32(3), int64(7), object(7)
memory usage: 121.2+ KB
```

After the above changes in data actual numeric variables got converted into numeric from object such as variables 'checking_balance', 'savings_balance' and 'employment_duration'

In [26]: `credit_df.head()`

	checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	er
0	1	6	critical	appliances	1169		1
1	2	48	good	appliances	5951		0
2	0	12	critical	education	2096		0
3	1	42	good	appliances	7882		0
4	1	24	poor	car	4870		0



Checking Missing values

In [27]: `credit_df.isna().sum()`

Out[27]:

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

No missing values are present.

Checking Duplicates

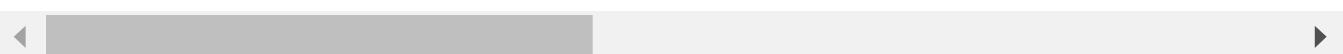
In [28]: `credit_df.duplicated().sum()`

Out[28]: 0

In [29]: `credit_df[credit_df.duplicated()]`

Out[29]:

checking_balance	months_loan_duration	credit_history	purpose	amount	savings_balance	empl
------------------	----------------------	----------------	---------	--------	-----------------	------



There are no duplicate values.

Unique Variables

In [30]:

```
dict={}
for i in credit_df.columns:
```

```

        x = len(credit_df[i].unique())
        dict[i]=x
        #print(f'{i}:\n{x}\n\n')
cnt = pd.DataFrame(dict.items(), columns = ['features','unique_count']).sort_values
cnt

```

Out[30]:

	features	unique_count
0	default	2
1	dependents	2
2	phone	2
3	housing	3
4	other_credit	3
5	job	4
6	existing_loans_count	4
7	checking_balance	4
8	percent_of_income	4
9	years_at_residence	4
10	employment_duration	5
11	savings_balance	5
12	credit_history	5
13	purpose	6
14	months_loan_duration	33
15	age	53
16	amount	921

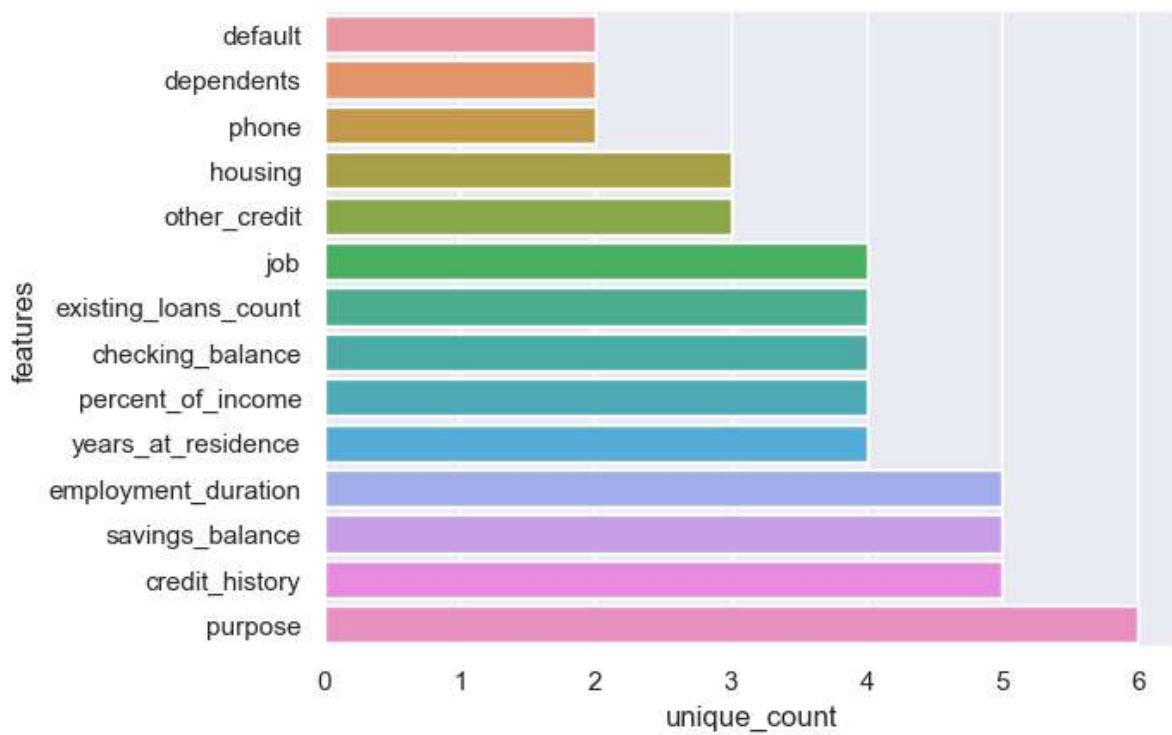
In [31]:

```

for i in credit_df.columns:

    if i not in ['months_loan_duration','age','amount']:
        sns.barplot(y= cnt['features'][:14], x = cnt['unique_count'][:14])
        #plt.xticks(rotation = 90)

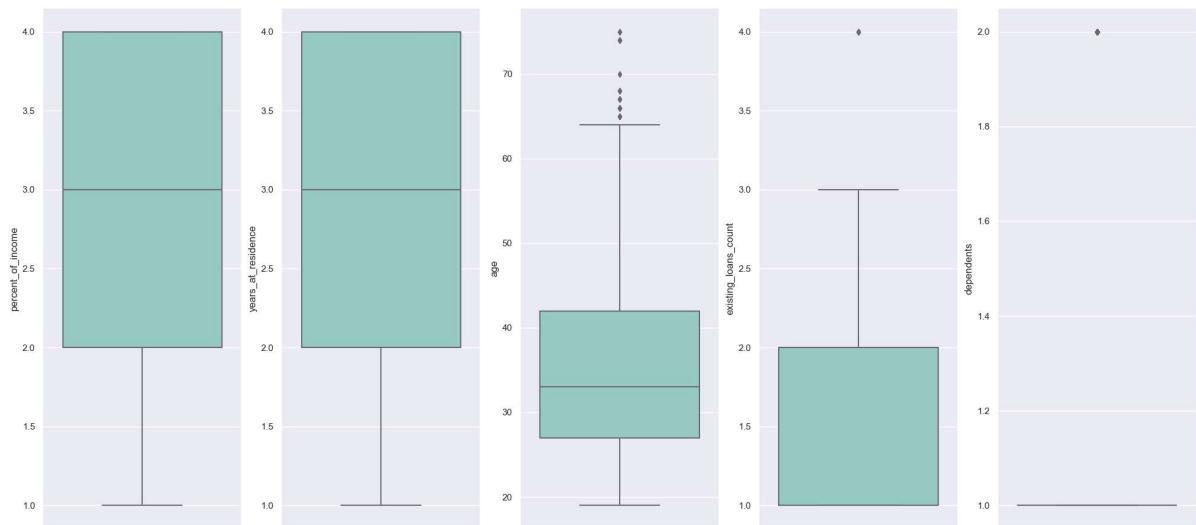
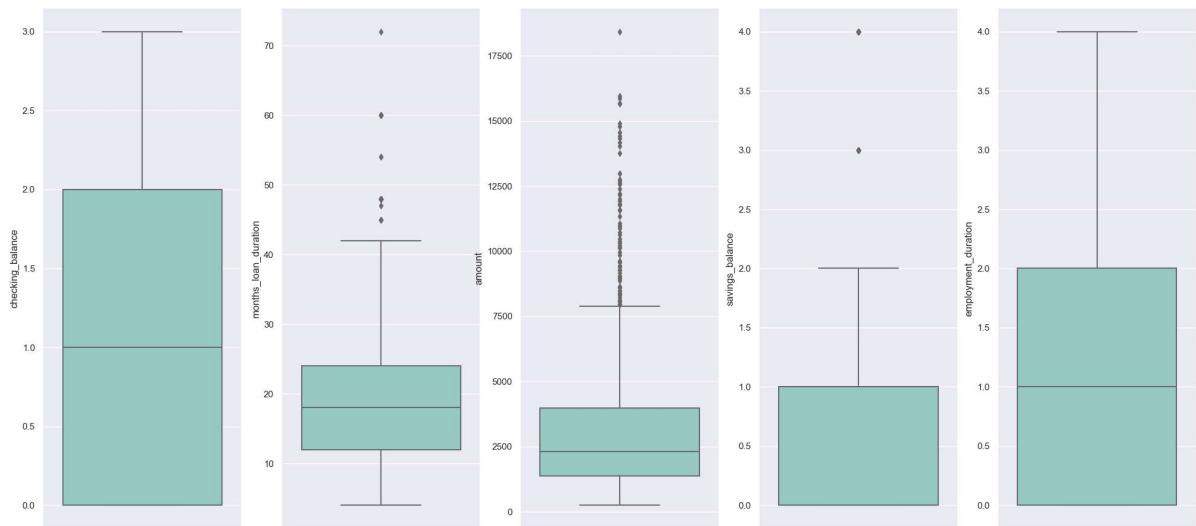
```



Checking Outliers

```
In [32]: plt.figure(figsize = (25, 25), dpi = 100)

for x, col in enumerate(credit_df.select_dtypes(include = 'int').columns):
    plt.subplot(2,5,x + 1)
    sns.boxplot(y = credit_df[col], data=credit_df, palette = 'Set3')
```



```
In [33]: credit_df.describe().loc[['min','max', '25%', '75%']].T
```

Out[33]:

	min	max	25%	75%
checking_balance	0.0	3.0	0.0	2.00
months_loan_duration	4.0	72.0	12.0	24.00
amount	250.0	18424.0	1365.5	3972.25
savings_balance	0.0	4.0	0.0	1.00
employment_duration	0.0	4.0	0.0	2.00
percent_of_income	1.0	4.0	2.0	4.00
years_at_residence	1.0	4.0	2.0	4.00
age	19.0	75.0	27.0	42.00
existing_loans_count	1.0	4.0	1.0	2.00
dependents	1.0	2.0	1.0	1.00

```
In [34]: q1 = credit_df.quantile(0.25)
q3 = credit_df.quantile(0.75)
IQR = q3-q1
```

```
upper_limit = q3 + 1.5*IQR
lower_limit = q1 - 1.5*IQR
```

```
In [35]: dict1 = {}
dict2 = {}
quant_list = []

dict1.update(upper_limit)
dict2.update(lower_limit)
quant_list.append(dict1)
quant_list.append(dict2)
```

```
In [36]: limit = pd.DataFrame(quant_list).T
limit.columns = ['upper_limit', 'lower_limit']
limit
```

Out[36]:

	upper_limit	lower_limit
checking_balance	5.000	-3.000
months_loan_duration	42.000	-6.000
amount	7882.375	-2544.625
savings_balance	2.500	-1.500
employment_duration	5.000	-3.000
percent_of_income	7.000	-1.000
years_at_residence	7.000	-1.000
age	64.500	4.500
existing_loans_count	3.500	-0.500
dependents	1.000	1.000

```
In [37]: #months_loan_duration, amount, age
#savings balance, existing_loans count, dependent
```

```
In [38]: #count of values above the 75% percentile
for i in ['months_loan_duration', 'amount', 'age', 'savings_balance', 'existing_loans_count']:
    print(i, ' - ', credit_df[credit_df[i] > credit_df.describe().loc['75%'][i]].shape[0])
months_loan_duration - 230
amount - 250
age - 235
savings_balance - 214
existing_loans_count - 34
dependents - 155
```

```
In [39]: #count of values above the upper limit
for i in ['months_loan_duration', 'amount', 'age', 'savings_balance', 'existing_loans_count']:
    print(i, ' - ', credit_df[credit_df[i] > limit.loc[i][0]].shape[0])
months_loan_duration - 70
amount - 72
age - 23
savings_balance - 111
existing_loans_count - 6
dependents - 155
```

```
In [40]: dict_values = {}
```

```
for i in ['months_loan_duration','amount','age','savings_balance','existing_loans_count']:
    print(i, ':', credit_df[credit_df[i] > limit.loc[i][0]][i].values, '\n')

months_loan_duration : [48 48 60 45 48 48 48 48 54 54 48 48 48 60 48 48 45 48 48 48 60 48 4
8 47 48 48 48
48 48 48 60 48 60 48 48 48 48 48 48 48 48 48 60 48 60 48 48 48 60
72 60 48 48 60 48 48 48 48 48 45 48 48 48 48 60 48 60 48 45 45]

amount : [ 9055  8072 12579  9566 14421  8133  9436 12612 15945 11938  8487 10144
8613  9572 10623 10961 14555  8978 12169 11998 10722  9398  9960 10127
11590 13756 14782 14318 12976 11760  8648  8471 11328 11054  8318  9034
8588  7966 8858 12389 12204 9157 15653  7980  8086 10222 10366  9857
14027 11560 14179 12680 8065  9271  9283  9629 15857 8335 11816 10875
9277 15672 8947 10477 18424 14896 12749 10297  8358 10974  8386  8229]

age : [67 66 66 70 65 74 68 66 75 74 65 75 67 74 65 66 74 66 67 65 68 65 68]

savings_balance : [3 4 3 3 4 4 3 3 3 3 4 3 4 3 3 3 3 4 3 4 4 4 3 3 3 4 3 4 4 3 3 3 4 3 4 4 3 4 4
3 3 4 3 4
4 3 4 4 4 3 3 4 3 4 4 3 3 3 3 4 4 4 3 4 4 3 3 3 4 3 4 4 4 3 3 3 3 4 3 3 3 4 3 3 3 3 3 3 3 3]

existing_loans_count : [4 4 4 4 4 4]

dependents : [2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

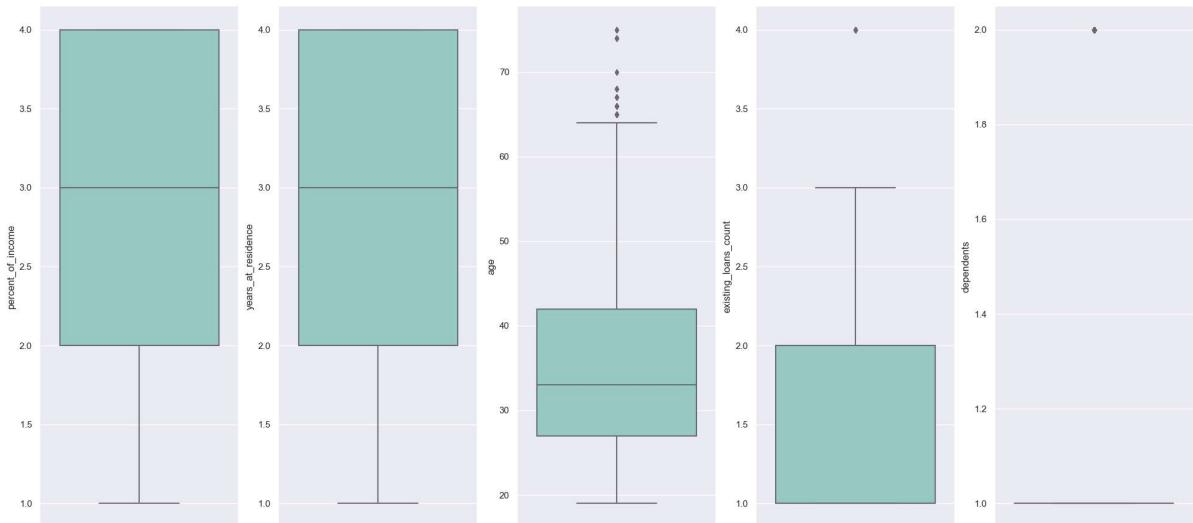
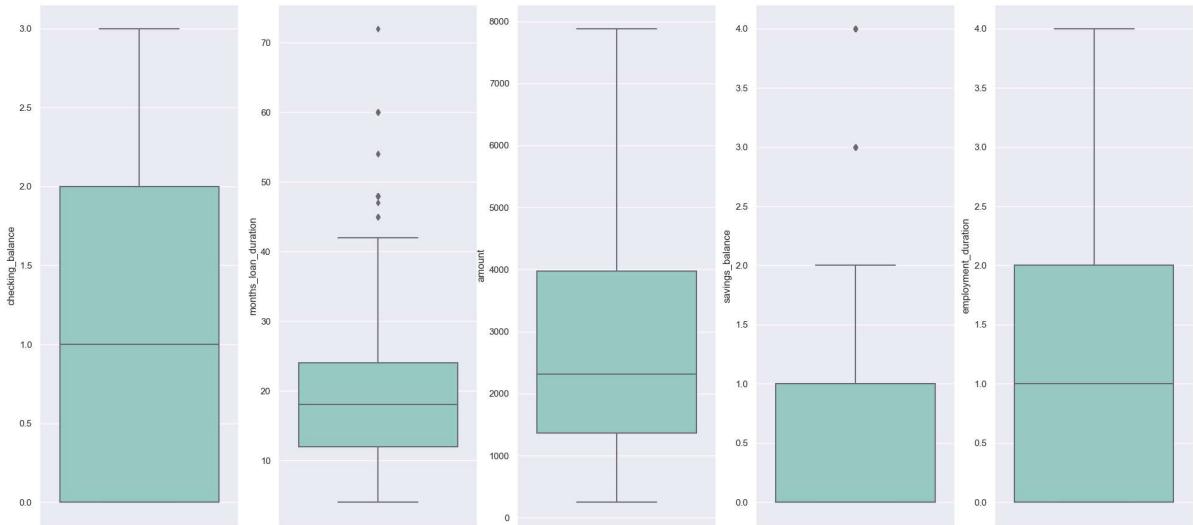
Handling Outliers

```
In [41]: # outliers to be removed : amount, months Loan duration > 50
```

```
In [42]: #['months_loan_duration','amount','age','savings_balance','existing_loans_count','age']
for i in ['amount']:
    credit_df[i].clip(upper = limit.loc[i][0] ,inplace=True)
```

```
In [43]: plt.figure(figsize = (25, 25), dpi = 100)

for x, col in enumerate(credit_df.select_dtypes(include = ['int','float']).columns):
    plt.subplot(2,5,x + 1)
    sns.boxplot(y = credit_df[col], data=credit_df, palette = 'Set3')
```



```
In [44]: #count of values above the upper limit
for i in ['months_loan_duration', 'amount', 'age', 'savings_balance', 'existing_loans_count']:
    print(i, '-' ,credit_df[credit_df[i] > limit.loc[i][0]].shape[0])

months_loan_duration - 70
amount - 0
age - 23
savings_balance - 111
existing_loans_count - 6
dependents - 155
```

Dropping Variable

```
In [45]: credit_df.drop(['phone'], axis =1, inplace = True)
```

Encoding

```
In [46]: # Lst = ['credit_history', 'purpose', 'other_credit', 'housing', 'job']
# for i in Lst:
#     print('*20,i,'*20)
#     print(credit_df[i].value_counts(), '\n')
```

```
In [47]: credit_df1 = credit_df.copy()
```

```
In [48]: credit_df1 = pd.get_dummies(credit_df1, columns=cnt['features'][:14], drop_first =
```

```
In [49]: credit_df1.head()
```

Out[49]:

	months_loan_duration	amount	age	default_yes	dependents_2	phone_yes	housing_own	hou
0	6	1169.0	67	0	0	1	1	1
1	48	5951.0	22	1	0	0	0	1
2	12	2096.0	49	0	1	0	1	1
3	42	7882.0	45	0	1	0	0	0
4	24	4870.0	53	1	1	0	0	0

5 rows × 42 columns

```
In [50]: credit_df1.columns
```

```
Out[50]: Index(['months_loan_duration', 'amount', 'age', 'default_yes', 'dependents_2',
       'phone_yes', 'housing_own', 'housing_rent', 'other_credit_none',
       'other_credit_store', 'job_skilled', 'job_unemployed', 'job_unskilled',
       'existing_loans_count_2', 'existing_loans_count_3',
       'existing_loans_count_4', 'checking_balance_1', 'checking_balance_2',
       'checking_balance_3', 'percent_of_income_2', 'percent_of_income_3',
       'percent_of_income_4', 'years_at_residence_2', 'years_at_residence_3',
       'years_at_residence_4', 'employment_duration_1',
       'employment_duration_2', 'employment_duration_3',
       'employment_duration_4', 'savings_balance_1', 'savings_balance_2',
       'savings_balance_3', 'savings_balance_4', 'credit_history_good',
       'credit_history_perfect', 'credit_history_poor',
       'credit_history_very_good', 'purpose_business', 'purpose_car',
       'purpose_car0', 'purpose_education', 'purpose_renovations'],
      dtype='object')
```

```
In [51]: credit_df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 42 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   months_loan_duration    1000 non-null   int64  
 1   amount                  1000 non-null   float64 
 2   age                     1000 non-null   int64  
 3   default_yes             1000 non-null   uint8  
 4   dependents_2             1000 non-null   uint8  
 5   phone_yes               1000 non-null   uint8  
 6   housing_own              1000 non-null   uint8  
 7   housing_rent              1000 non-null   uint8  
 8   other_credit_none        1000 non-null   uint8  
 9   other_credit_store       1000 non-null   uint8  
 10  job_skilled              1000 non-null   uint8  
 11  job_unemployed          1000 non-null   uint8  
 12  job_unskilled            1000 non-null   uint8  
 13  existing_loans_count_2  1000 non-null   uint8  
 14  existing_loans_count_3  1000 non-null   uint8  
 15  existing_loans_count_4  1000 non-null   uint8  
 16  checking_balance_1       1000 non-null   uint8  
 17  checking_balance_2       1000 non-null   uint8  
 18  checking_balance_3       1000 non-null   uint8  
 19  percent_of_income_2      1000 non-null   uint8  
 20  percent_of_income_3      1000 non-null   uint8  
 21  percent_of_income_4      1000 non-null   uint8  
 22  years_at_residence_2     1000 non-null   uint8  
 23  years_at_residence_3     1000 non-null   uint8  
 24  years_at_residence_4     1000 non-null   uint8  
 25  employment_duration_1    1000 non-null   uint8  
 26  employment_duration_2    1000 non-null   uint8  
 27  employment_duration_3    1000 non-null   uint8  
 28  employment_duration_4    1000 non-null   uint8  
 29  savings_balance_1        1000 non-null   uint8  
 30  savings_balance_2        1000 non-null   uint8  
 31  savings_balance_3        1000 non-null   uint8  
 32  savings_balance_4        1000 non-null   uint8  
 33  credit_history_good       1000 non-null   uint8  
 34  credit_history_perfect    1000 non-null   uint8  
 35  credit_history_poor       1000 non-null   uint8  
 36  credit_history_very good  1000 non-null   uint8  
 37  purpose_business          1000 non-null   uint8  
 38  purpose_car               1000 non-null   uint8  
 39  purpose_car0              1000 non-null   uint8  
 40  purpose_education         1000 non-null   uint8  
 41  purpose_renovations       1000 non-null   uint8  
dtypes: float64(1), int64(2), uint8(39)
memory usage: 61.6 KB
```

In [52]: `credit_df1 = credit_df1.rename(columns = {'default_yes':'default'})`

In [53]: `credit_df1.head()`

Out[53]:

	months_loan_duration	amount	age	default	dependents_2	phone_yes	housing_own	housing...
0	6	1169.0	67	0	0	1	1	1
1	48	5951.0	22	1	0	0	1	1
2	12	2096.0	49	0	1	0	1	1
3	42	7882.0	45	0	1	0	0	0
4	24	4870.0	53	1	1	0	0	0

5 rows × 42 columns

In [54]: credit_df1.drop(['phone_yes'], axis=1, inplace=True)

Separate Independent And Dependent Variables

In [55]: x = credit_df1.drop(['default'], axis=1)
y = credit_df1['default']In [56]: print(credit_df1.shape)
print(x.shape)
print(y.shape)
print(y.head())

```
(1000, 41)
(1000, 40)
(1000,)
0    0
1    1
2    0
3    0
4    1
Name: default, dtype: uint8
```

Feature Scaling

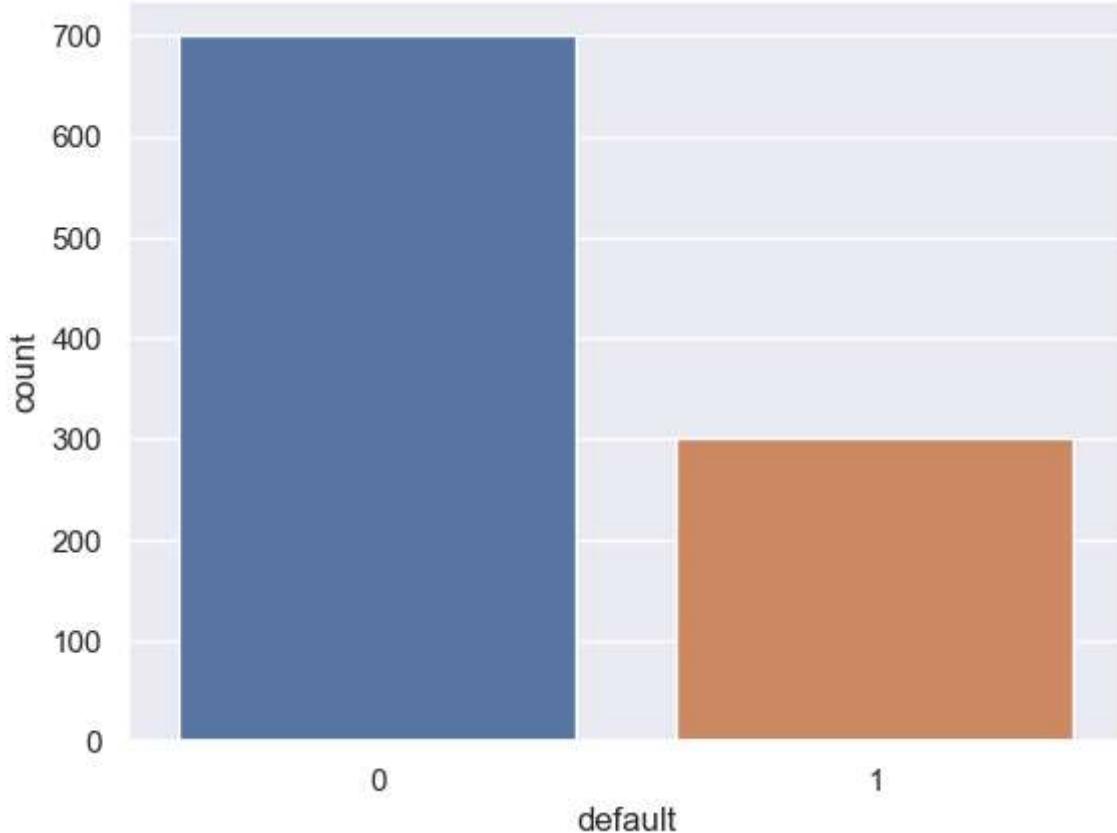
In [57]: from sklearn.preprocessing import StandardScaler
fsc = StandardScaler()
sc_x = fsc.fit_transform(x)
sc_x

Out[57]: array([[-1.23647786, -0.86096108, 2.76645648, ..., -0.11020775,
 -0.2503982 , -0.14998296],
 [2.24819436, 1.32654951, -1.19140394, ..., -0.11020775,
 -0.2503982 , -0.14998296],
 [-0.73866754, -0.43690789, 1.18331231, ..., -0.11020775,
 3.99363901, -0.14998296],
 ...,
 [-0.73866754, -1.02792916, 0.21583532, ..., -0.11020775,
 -0.2503982 , -0.14998296],
 [1.9992892 , -0.55172704, -1.10345149, ..., -0.11020775,
 -0.2503982 , -0.14998296],
 [1.9992892 , 0.69756017, -0.75164167, ..., -0.11020775,
 -0.2503982 , -0.14998296]])

Imbalance Check

```
In [58]: sns.countplot(data=credit_df, x = y)
print('No has the count : ', y.value_counts()[0])
print('Yes has the count : ',y.value_counts()[1])
plt.show()
```

No has the count : 700
 Yes has the count : 300



*This is the imbalance data. ($300 < 700$)**

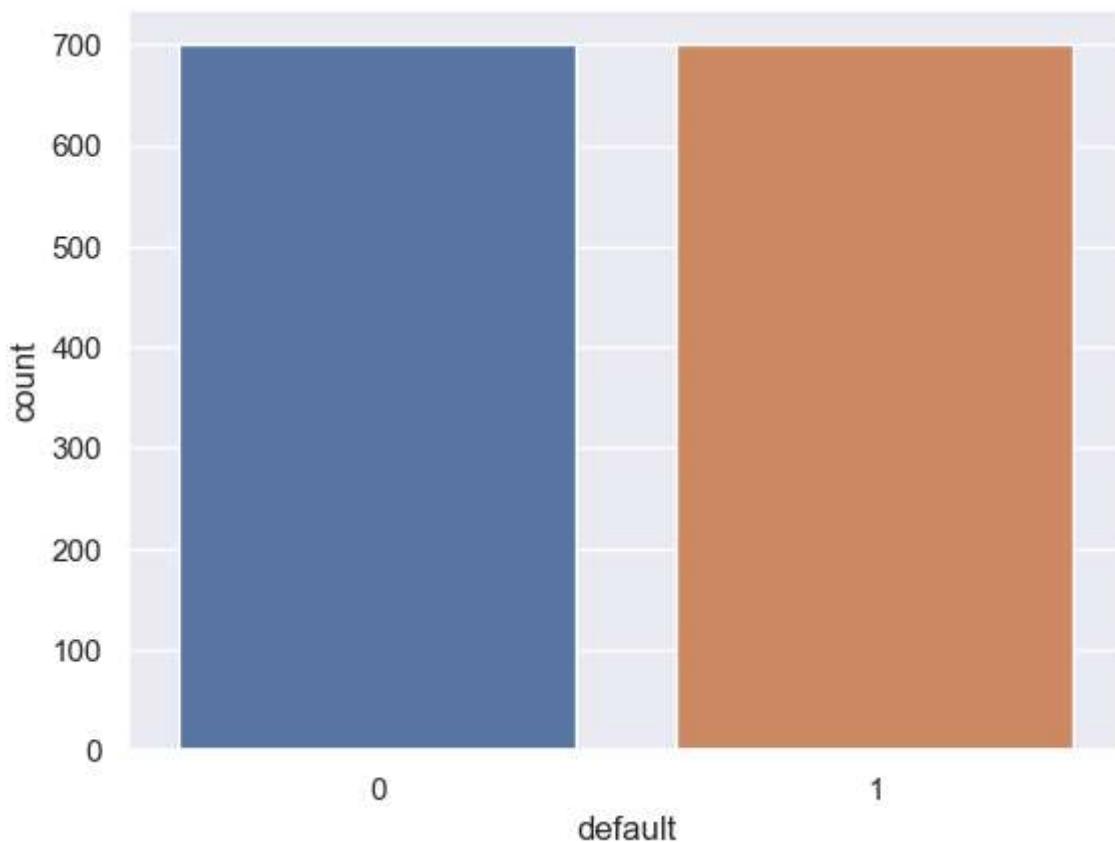
Handle Imbalance Data

```
In [59]: # Oversampling method
import imblearn
```

```
In [60]: from imblearn.over_sampling import RandomOverSampler
over = RandomOverSampler()
x_over, y_over = over.fit_resample(sc_x,y)
```

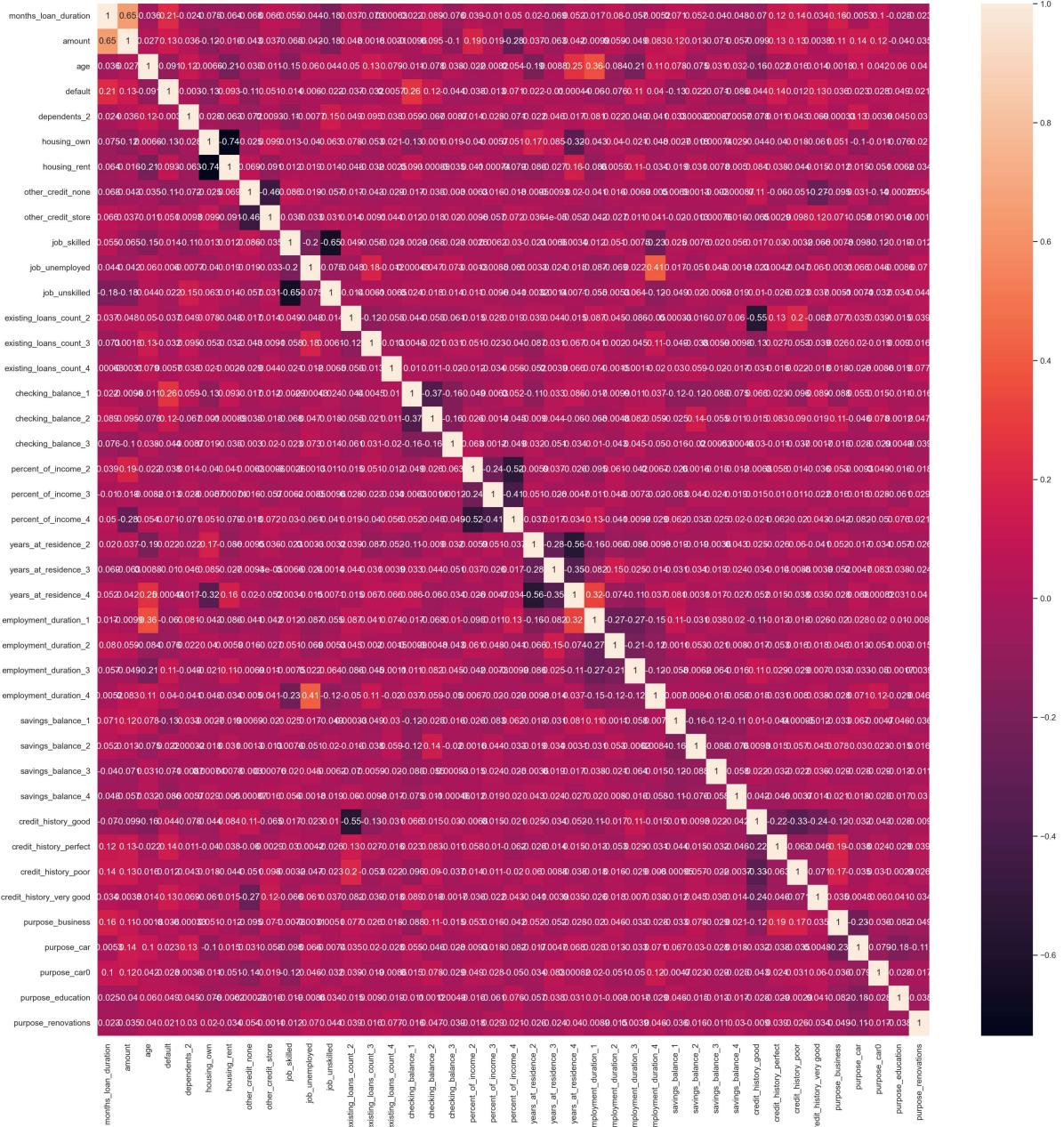
```
In [61]: sns.countplot(data=credit_df, x = y_over)
print('No has the count : ', y_over.value_counts()[0])
print('Yes has the count : ',y_over.value_counts()[1])
plt.show()
```

No has the count : 700
 Yes has the count : 700



```
In [62]: plt.figure(figsize =(25,25), dpi = 200)
sns.heatmap(data = credit_df1.corr(), annot =True)

Out[62]: <AxesSubplot:>
```



```
In [63]: # plt.figure(figsize=(50,50))
# sns.pairplot(data = credit_df1, hue = 'default')
```

Splitting Test And Train Dataset

```
In [64]: from sklearn.model_selection import train_test_split
np.random.seed(10)
x_train, x_test, y_train, y_test = train_test_split(x_over, y_over, test_size = 0.2)
print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

(1050, 40) (350, 40) (1050,) (350,)

Model Building

```
In [65]: from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

```
from sklearn.model_selection import cross_val_score
```

1. Logistic Regression

```
In [66]: np.random.seed(10)
lr = LogisticRegression()
lr = lr.fit(x_train, y_train)
```

```
In [67]: y_lr_predict_train = lr.predict(x_train)
y_lr_predict_test = lr.predict(x_test)
```

```
In [68]: r = pd.DataFrame(y_test)
r.insert(1, 'predicted', y_lr_predict_test )
r
```

Out[68]:

	default	predicted
580	1	1
867	0	0
246	0	0
1378	1	1
287	0	1
...
1040	1	0
831	1	1
172	1	1
1255	1	0
1237	1	0

350 rows × 2 columns

Evaluate The Model

```
In [69]: print('*20,'Accuracy Score','*20,'\n')
print("The Accuracy Score for train data is : ", accuracy_score(y_train,y_lr_predict_
print("The Accuracy Score for test data is : ", accuracy_score(y_test,y_lr_predict_)

print('\n\n','*20,'Classification Report','*20,'\
print("The Classification Report for train data is :\n ", classification_report(y_t
print("The Classification Report for test data is :\n ", classification_report(y_te

print('\n\n','*20,'Confusion Matrix','*20,'\
print("The Confusion Matrix for train data is :\n", confusion_matrix(y_train,y_lr_p
print("The Confusion Matrix for test data is :\n", confusion_matrix(y_test,y_lr_p
```

===== Accuracy Score =====

The Accuracy Score for train data is : 0.7304761904761905
The Accuracy Score for test data is : 0.7114285714285714

===== Classification Report =====

The Classification Report for train data is :

	precision	recall	f1-score	support
0	0.74	0.71	0.72	525
1	0.72	0.75	0.74	525
accuracy			0.73	1050
macro avg	0.73	0.73	0.73	1050
weighted avg	0.73	0.73	0.73	1050

The Classification Report for test data is :

	precision	recall	f1-score	support
0	0.72	0.70	0.71	175
1	0.71	0.73	0.72	175
accuracy			0.71	350
macro avg	0.71	0.71	0.71	350
weighted avg	0.71	0.71	0.71	350

===== Confusion Matrix =====

The Confusion Matrix for train data is :

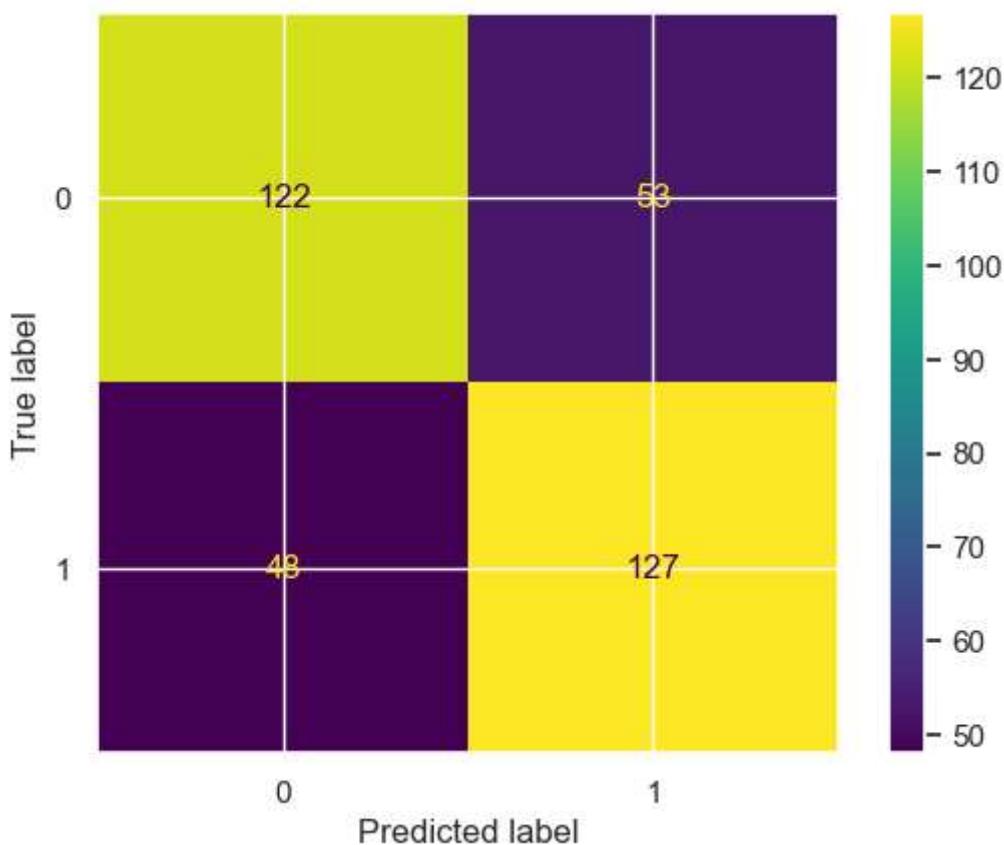
```
[[371 154]
 [129 396]]
```

The Confusion Matrix for test data is :

```
[[122  53]
 [ 48 127]]
```

In [70]: `from sklearn.metrics import ConfusionMatrixDisplay`

```
cm = confusion_matrix(y_test, y_lr_predict_test, labels = lr.classes_ )
display = ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = lr.classes_
display.plot()
plt.show()
```



Understanding Confusion Matrix



- Higher the recall/sensitivity, precision and F1-score better the result.

ROC/AUC

```
In [71]: # roc_auc for test data
lr_roc_auc = roc_auc_score(y_test,y_lr_predict_test )
print("The roc_auc_score for logistic regression is : ", lr_roc_auc)
```

The roc_auc_score for logistic regression is : 0.7114285714285714

```
In [72]: fpr, tpr, thresholds = roc_curve(y_test, y_lr_predict_test)
```

```
In [73]: print("fpr : ", fpr)
print("tpr : ", tpr)
print("Thresholds : ", thresholds)
```

```
fpr :  [0.          0.30285714 1.          ]
tpr :  [0.          0.72571429 1.          ]
Thresholds :  [2 1 0]
```

True Positive Rate(TPR) & False Positive Rate(FPR)



- ROC: Receiver Operating Characteristics
- AUC: Area Under Curve

AUC stands for Area Under the Curve, and the AUC curve represents the area under the ROC curve. It measures the overall performance of the binary classification model. As both TPR and FPR range between 0 to 1, So, the area will always lie between 0 and 1, and A greater value of AUC denotes better model performance. Our main goal is to maximize this area in order to have the highest TPR and lowest FPR at the given threshold

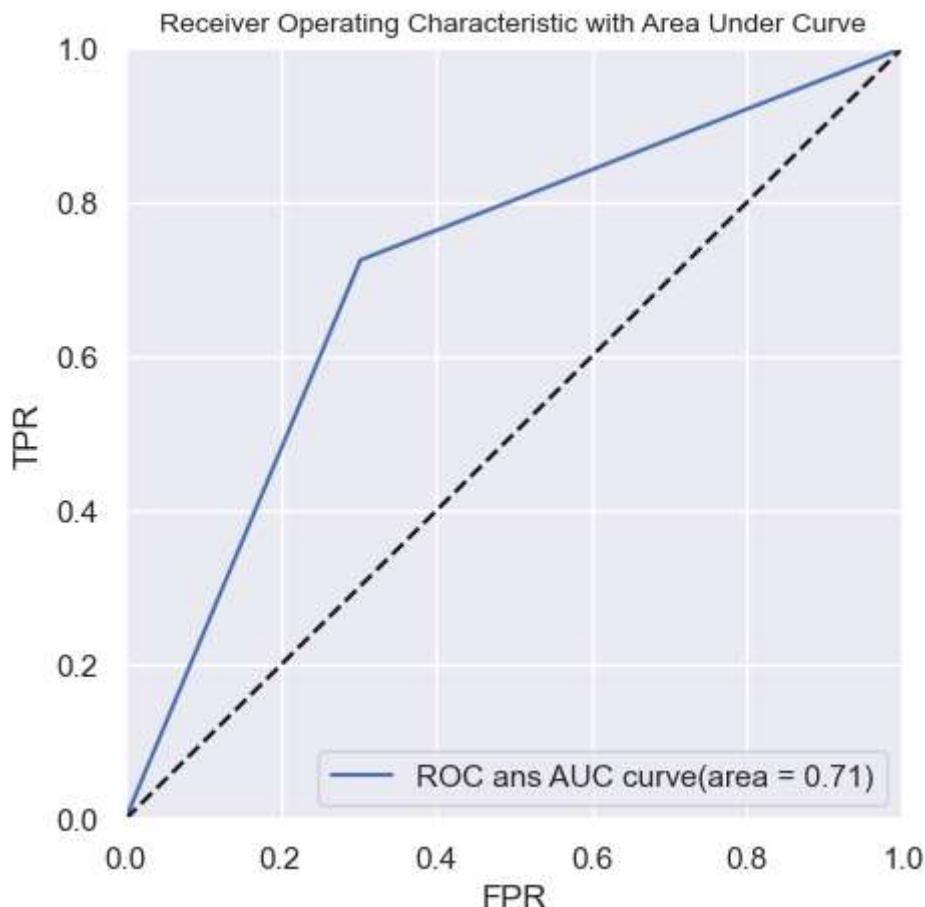
- AUC measures how well a model is able to distinguish between classes.

The classification threshold that returns the upper-left corner of the curve—minimizing the difference between TPR and FPR—is the optimal threshold.

Plotting the ROC-AUC curve

```
In [74]: plt.figure(figsize = (5,5))
plt.plot(fpr, tpr, label = "ROC ans AUC curve(area = %0.2f)" %lr_roc_auc)
plt.plot([0,1],[0,1],'k--')
plt.xlim(0,1)
plt.ylim(0,1)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('Receiver Operating Characteristic with Area Under Curve', fontsize =10)
plt.legend(loc='lower right')

Out[74]: <matplotlib.legend.Legend at 0x1ea7e783be0>
```



Cross Validation

```
In [75]: Training_CV = cross_val_score(lr, x_train, y_train, cv =10)
Testing_CV = cross_val_score(lr, x_test, y_test, cv= 10)
Avg_Training_CV = Training_CV.mean()
Avg_Testing_CV = Testing_CV.mean()

print(f"Training CV: {Training_CV}\n\nTesting CV: {Testing_CV}\n\nAvg_Training_CV:
Training CV: [0.73333333 0.66666667 0.73333333 0.73333333 0.67619048 0.71428571
0.66666667 0.66666667 0.74285714 0.71428571]

Testing CV: [0.82857143 0.65714286 0.62857143 0.71428571 0.71428571 0.54285714
0.74285714 0.82857143 0.68571429 0.77142857]

Avg_Training_CV: 0.7047619047619047

Avg_Testing_CV: 0.7114285714285715
```

```
In [ ]:
```