💰 💷 💵 💳 Credit Card 💳 💵 💷 💰

🎯 Outline of the Project 🎯

1. 🗂️**Importing Libraries: - To perform Data Manipulation,Visualization & Model Building.**

2. ⏳ **Loading Dataset: - Load the dataset into a suitable data structure using pandas.**

3. 🔎**Exploration of Dataset: - Generate basic informations about the data.**

4. 🤌 Data Cleaning: - garbage cleaning, removing duplicate, and hadling missing values and treating outliers

5. 📊 Exploatory Data Analysis: - To identify trends, patterns, and relationships among the variabels.

6. ⚙️ Data Preprocessing: - To transform data for creating more accurate & robust model.

7. 🎯 Model building:- To build predictive models, using various algorithms.

8. ⚡ Model evaluation: - To analyze the Model performance using metrics.

9. 📝 Conclusion: - Conclude the project by summarizing the key findings.

🗂️ Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

sns.set()
%matplotlib inline

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

⏳ Loading Dataset

In [2]:

```
dataset = 'German Credit Dataset.csv'
df = pd.read_csv(dataset)
df.head()
```

Out[2]:

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

🔎 Exoloring Dataset

In [3]:

```
df.shape
```

Out[3]:

```
(1000, 17)
```

In [4]:

```
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   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 [5]:

```
object_cols_df = df.select_dtypes(include='object')
object_col = list(object_cols_df.columns)
object_col
```

Out[5]:

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

In [6]:

```
numeric_cols_df = df.select_dtypes(include='number')
num_col = list(numeric_cols_df.columns)
num_col
```

Out[6]:

```
['months_loan_duration',
 'amount',
 'percent_of_income',
 'years_at_residence',
 'age',
 'existing_loans_count',
 'dependents']
```

🩹 Cleaning of dataset

Checking and Removing Duplicates

In [7]:

```
# Remove duplicates

def drop_dup(df):
    if df.duplicated().any() == True:
        print('The total duplicate row before removing duplicate:', df.duplicated().sum(
        df.drop_duplicates(inplace=True , keep = 'last')  # Remove duplicates
        df = df.reset_index(drop=True) #Reset the index
        print('The total duplicate row after removing duplicate:', df.duplicated().sum()
    else:
        return 'No duplicate entries'
drop_dup(df)
```

Out[7]:

```
'No duplicate entries'
```

Checking null values

In [8]:

```
df.isnull().sum()
```

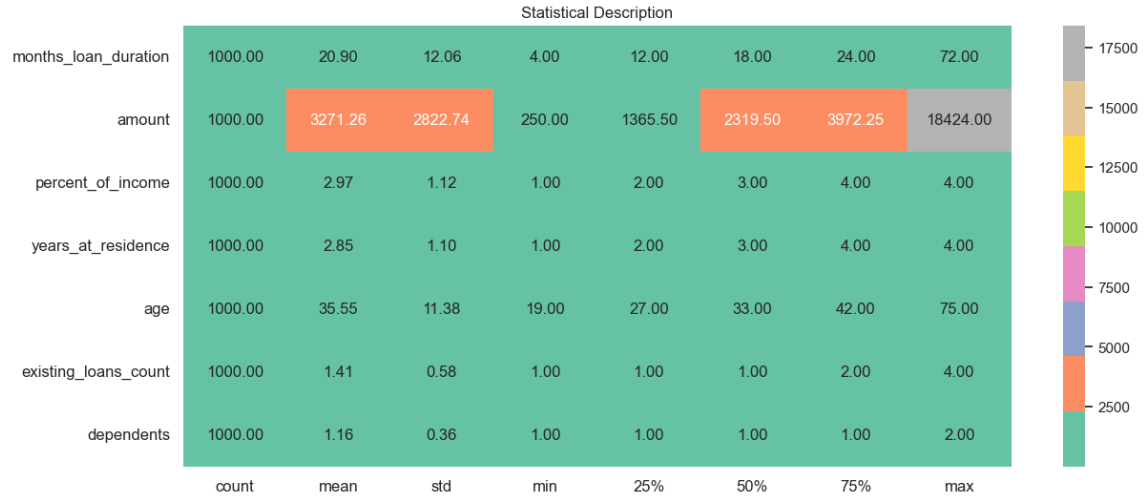
Out[8]:

```
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
```

📊 Descriptive Statistics

In [9]:

```
desc=df.describe().T
def descriptive_stats(df):


    plt.figure(figsize=(14,6))
    sns.heatmap(df, annot=True, cmap='Set2', fmt=".2f")
    plt.xticks(size = 12)
    plt.yticks(size = 12, rotation = 0)
    plt.title('Statistical Description')
    plt.show()


descriptive_stats(desc)
```

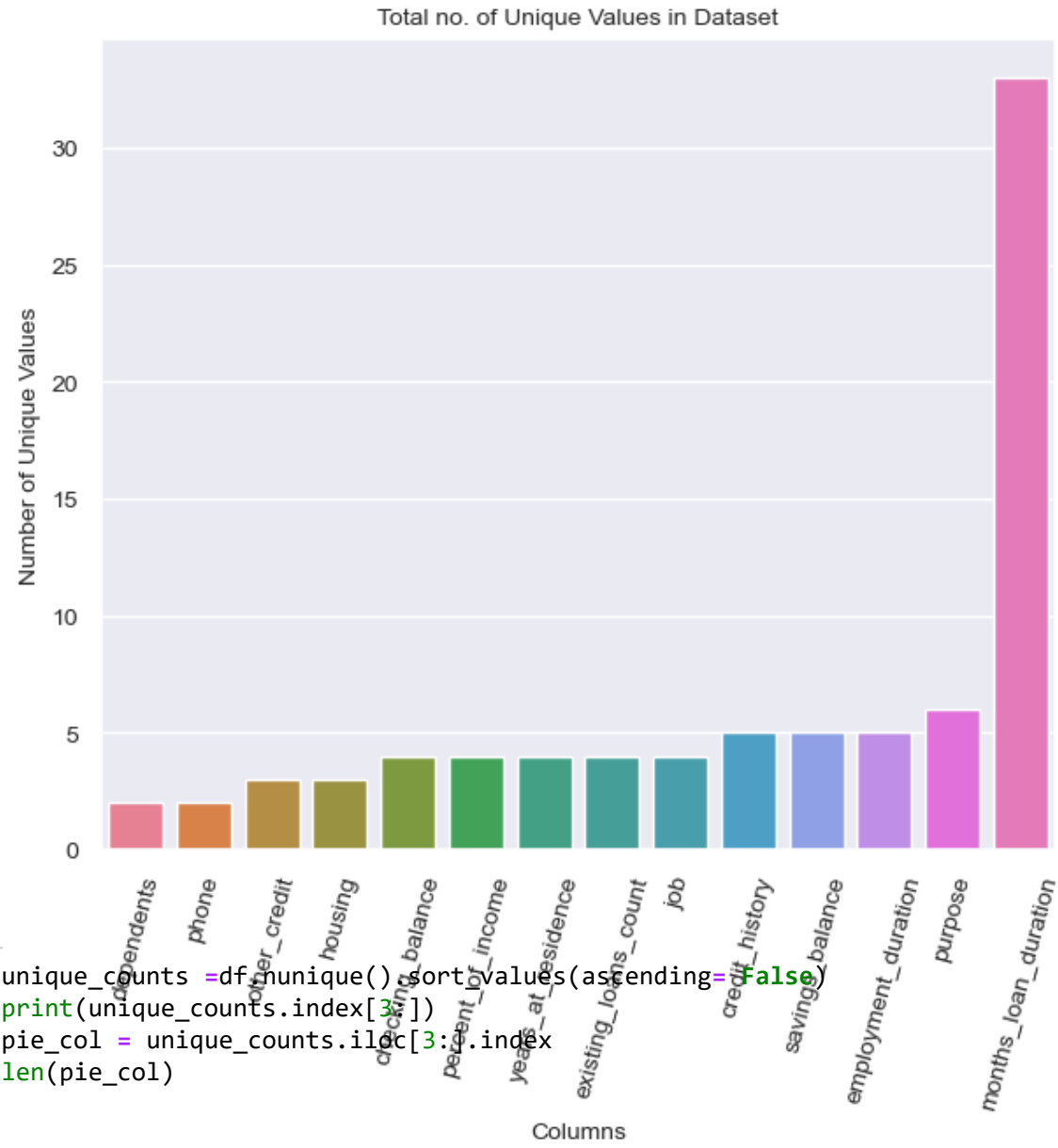


Checking Unique values of different features

In [10]:

```
# Count the number of unique values in each column
def check_unquie_count(df):
    unique_counts = df.nunique().sort_values()

    print('=='*30)
    print('  '*10, 'Total no. of Unique Values')
    print('=='*30)
    print(unique_counts)
    print('=='*30)
# Create a bar plot or count plot of unique values
    #plt.style.use('dark_background')
    plt.figure(figsize=(7, 6))
    sns.barplot(x=unique_counts.index[:-2], y=unique_counts.iloc[:-2],palette='husl' )

    plt.xticks(rotation=75, fontsize= 10)
    plt.yticks( fontsize= 10 )
    plt.xlabel('Columns',fontsize=10)
    plt.ylabel('Number of Unique Values', fontsize=10)
    plt.title('Total no. of Unique Values in Dataset', fontsize=10)

# Display the plot
    plt.show()

check_unquie_count(df.iloc[:,0:-1])
```

```
============================================================
                    Total no. of Unique Values
============================================================
dependents                2
phone                     2
other_credit              3
housing                   3
checking_balance          4
percent_of_income         4
years_at_residence        4
existing_loans_count      4
job                       4
credit_history            5
savings_balance           5
employment_duration       5
purpose                   6
months_loan_duration     33
age                      53
amount                  921
dtype: int64
============================================================
```




```
unique_counts =df.nunique().sort_values(ascending=False)
print(unique_counts.index[3:])
pie_col = unique_counts.iloc[3:].index
len(pie_col)
```

```
Index(['purpose', 'credit_history', 'savings_balance', 'employment_duratio
n',
       'checking_balance', 'existing_loans_count', 'job', 'years_at_reside
nce',
       'percent_of_income', 'housing', 'other_credit', 'dependents', 'phon
e',
       'default'],
      dtype='object')
```

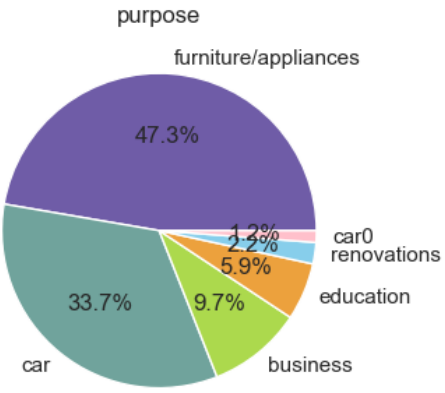
Out[11]:

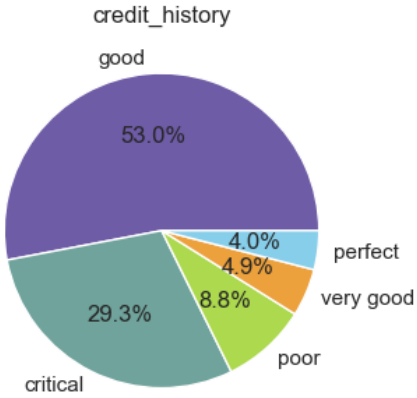
14

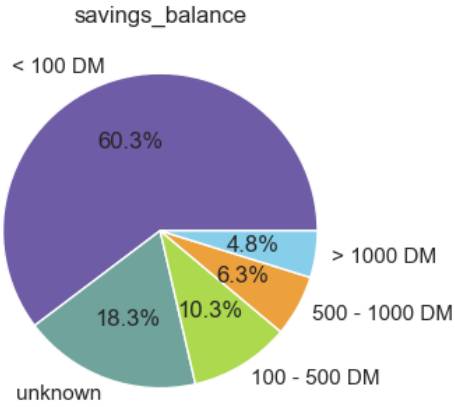
percentage count of categorical columns

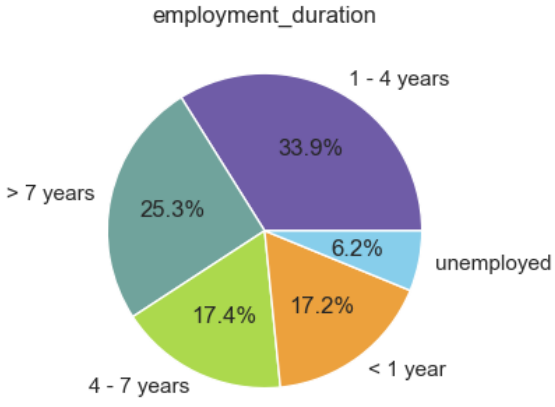
In [12]:

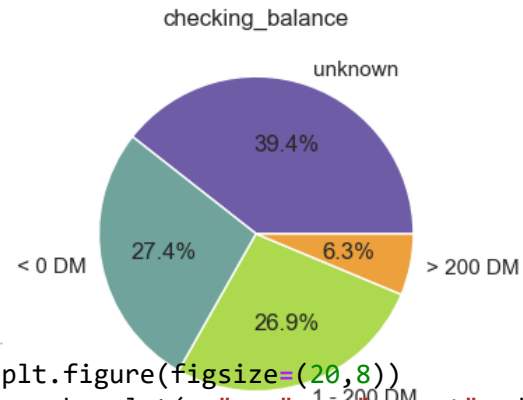
```
plt.figure(figsize=(10,30))
for i, col in enumerate(pie_col):
    colors =['#6F5CA7','#70A39C', '#ACD94D', '#ECA13D','skyblue','pink']
    plt.subplot(7,2,i+1)
    df[col].value_counts().plot(kind='pie', autopct='%1.1f%%', colors =colors)
    plt.title(f'{col}')
    plt.ylabel('')
```

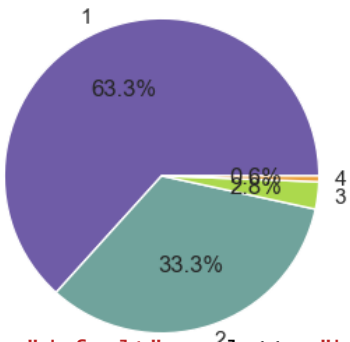








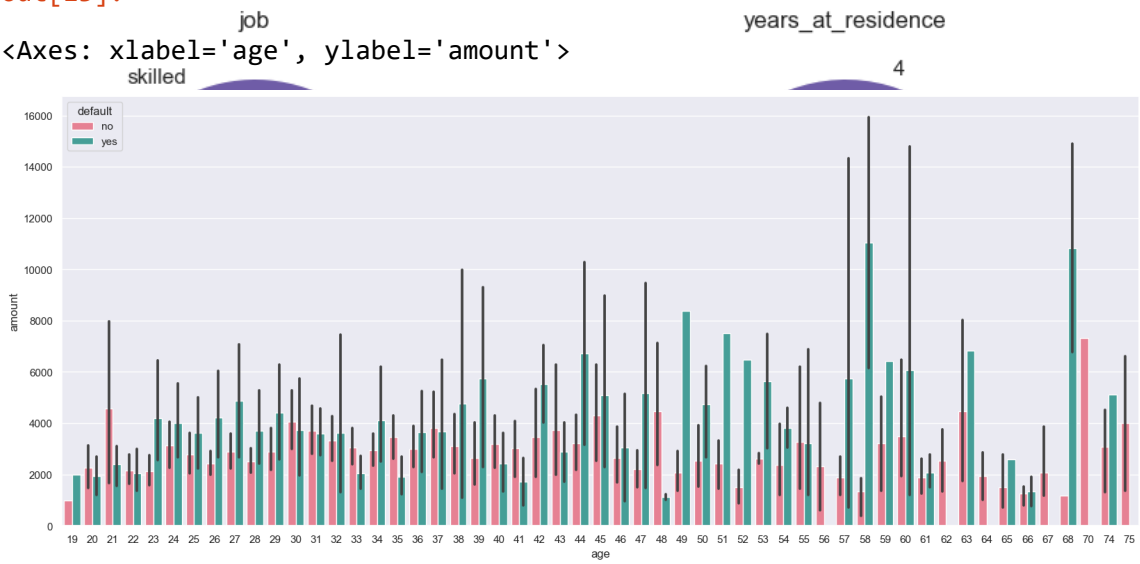


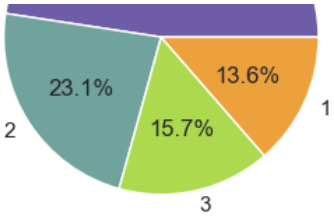


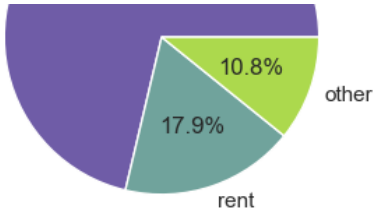

```
plt.figure(figsize=(20,8))
sns.barplot(x="age", y="amount", data=df, hue="default", palette="husl")
```

Out[13]:

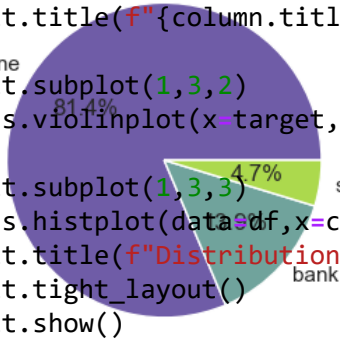
```
<Axes: xlabel='age', ylabel='amount'>
```

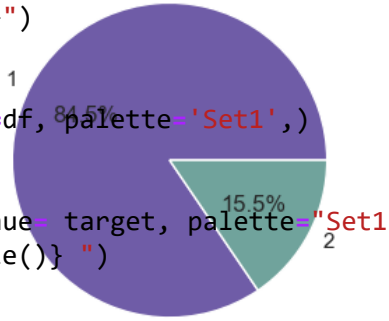


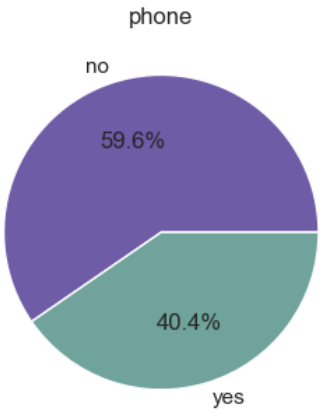


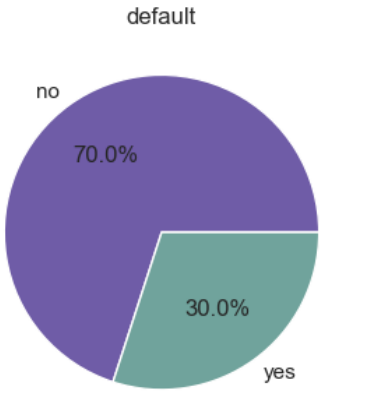



```
def num_dist(column, target= 'default'):
    plt.figure(figsize=(12,4))
    plt.subplot(1,3,1)
    sns.boxplot(x=target,y=column, data=df, palette="Set2")
    plt.title(f"{column.title()} vs {target}")
    plt.subplot(1,3,2)
    sns.violinplot(x=target, y=column, data=df, palette='Set1',)

    plt.subplot(1,3,3)
    sns.histplot(data=df,x=column,kde=True,hue= target, palette="Set1")
    plt.title(f"Distribution of {column.title()} ")
    plt.tight_layout()
    plt.show()
```



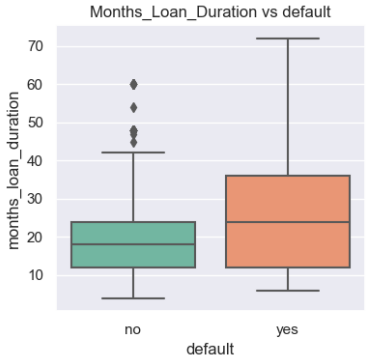


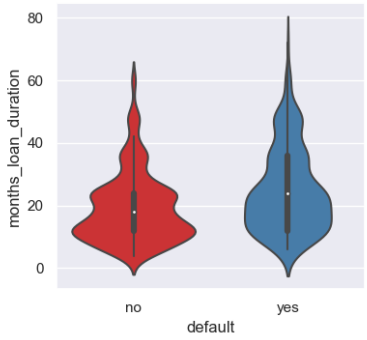


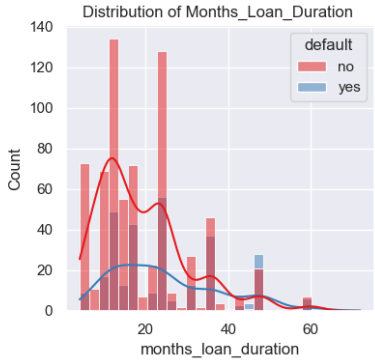


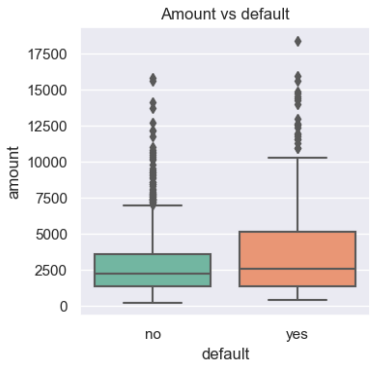
In [15]:

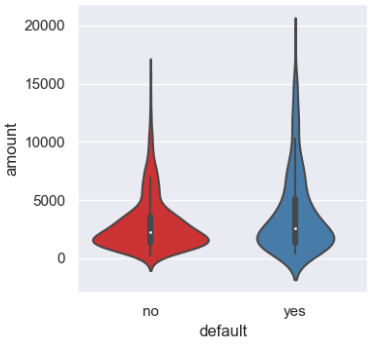
```
for col in num_col:
    num_dist(col,target ='default')
```

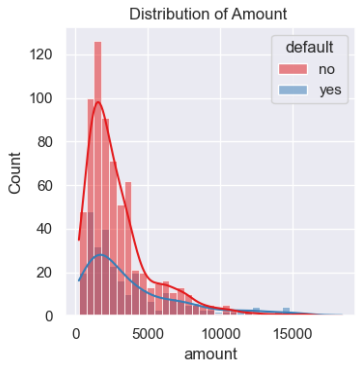


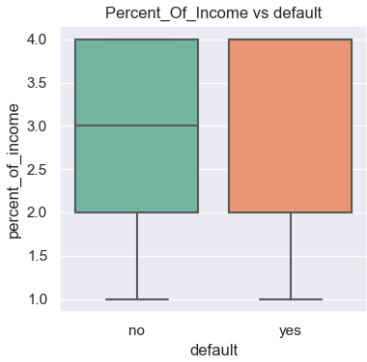


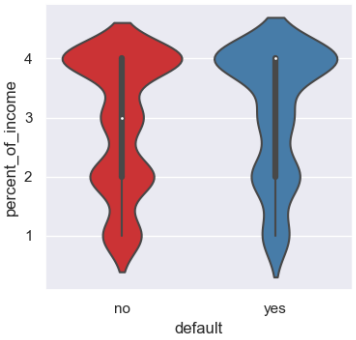


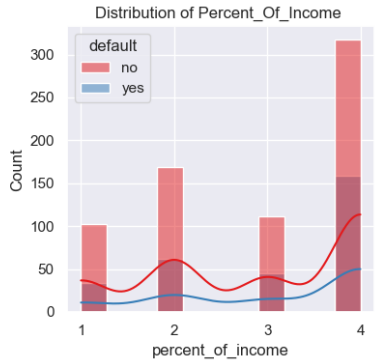


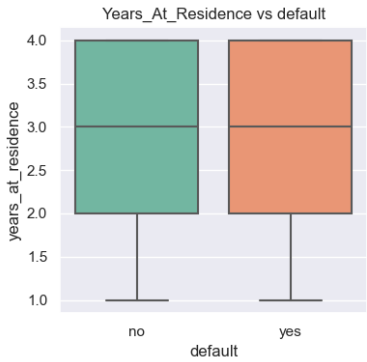


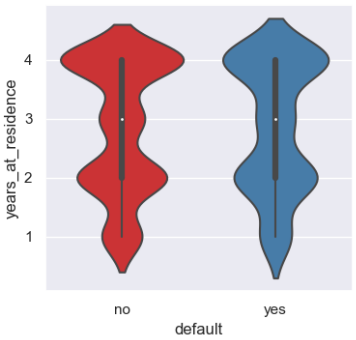


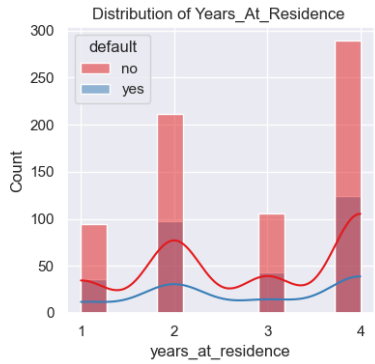


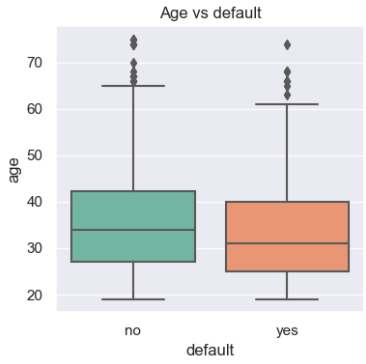


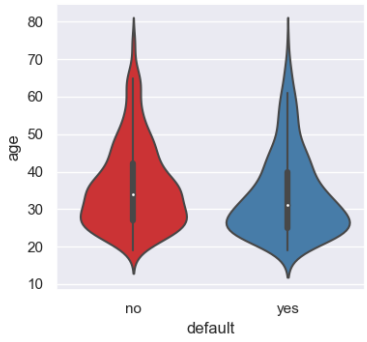


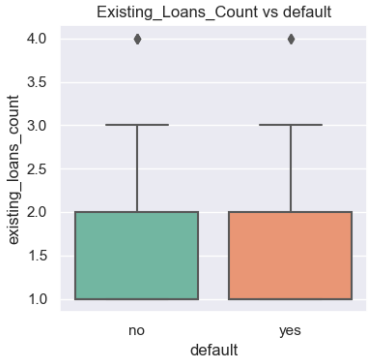


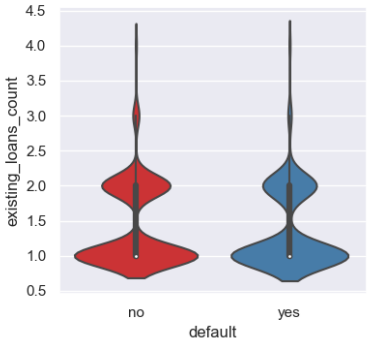


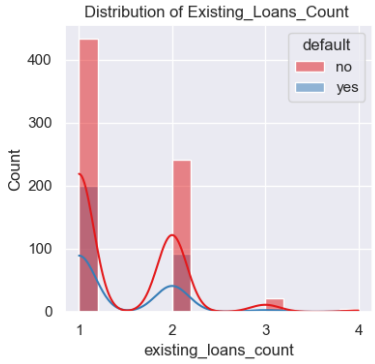


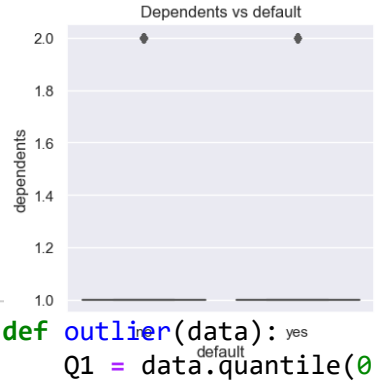


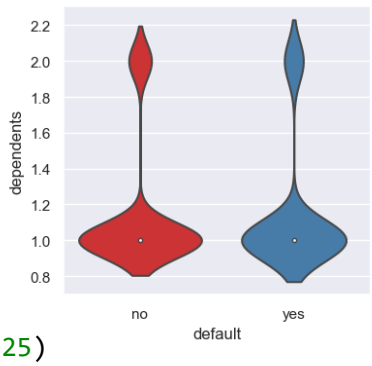


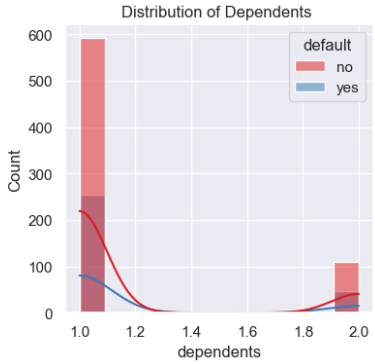




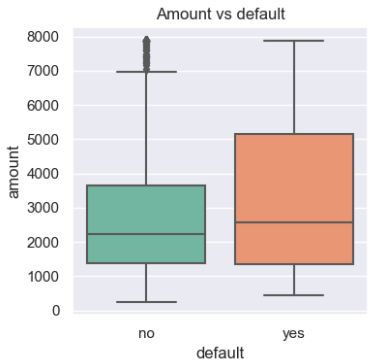


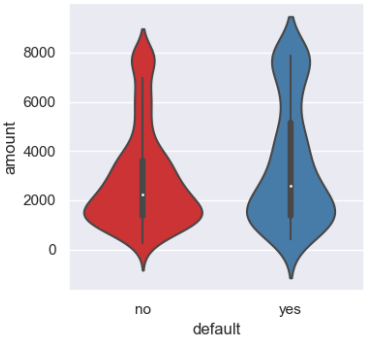


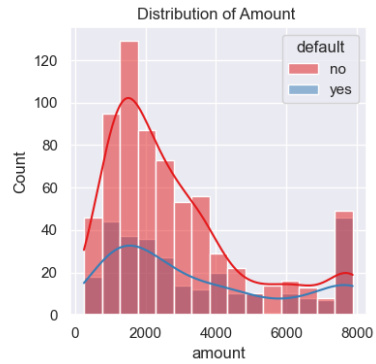





```
def outlier(data):
    Q1 = data.quantile(0.25)
    Q3 = data.quantile(0.75)
    IQR = Q3-Q1
    upper_bound = Q3 + 1.5*IQR
    lower_bound = Q1 - 1.5*IQR
    return data.clip(upper_bound, lower_bound)

df['amount'] = outlier(df['amount'])
num_dist('amount',target ='default')
```





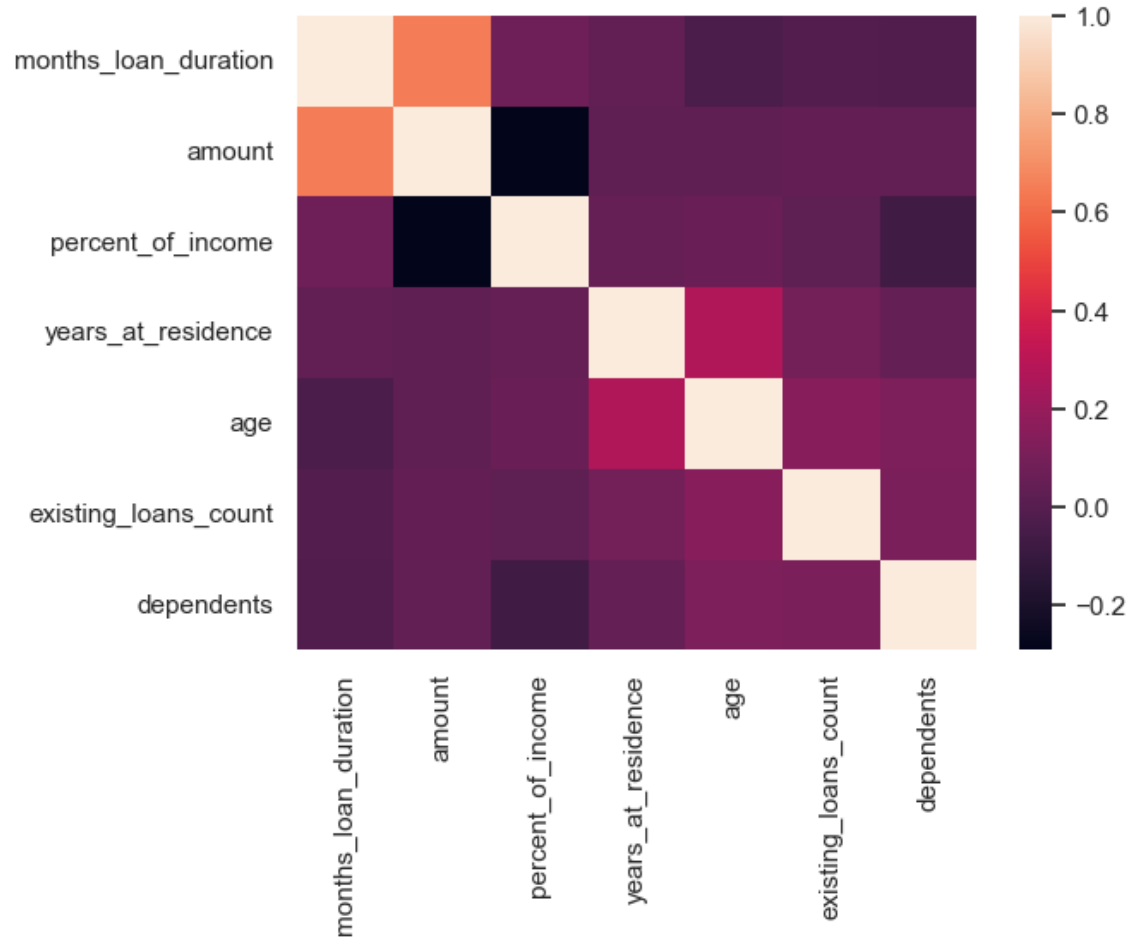


In [17]:

```
sns.heatmap(df.select_dtypes('number').corr())
```

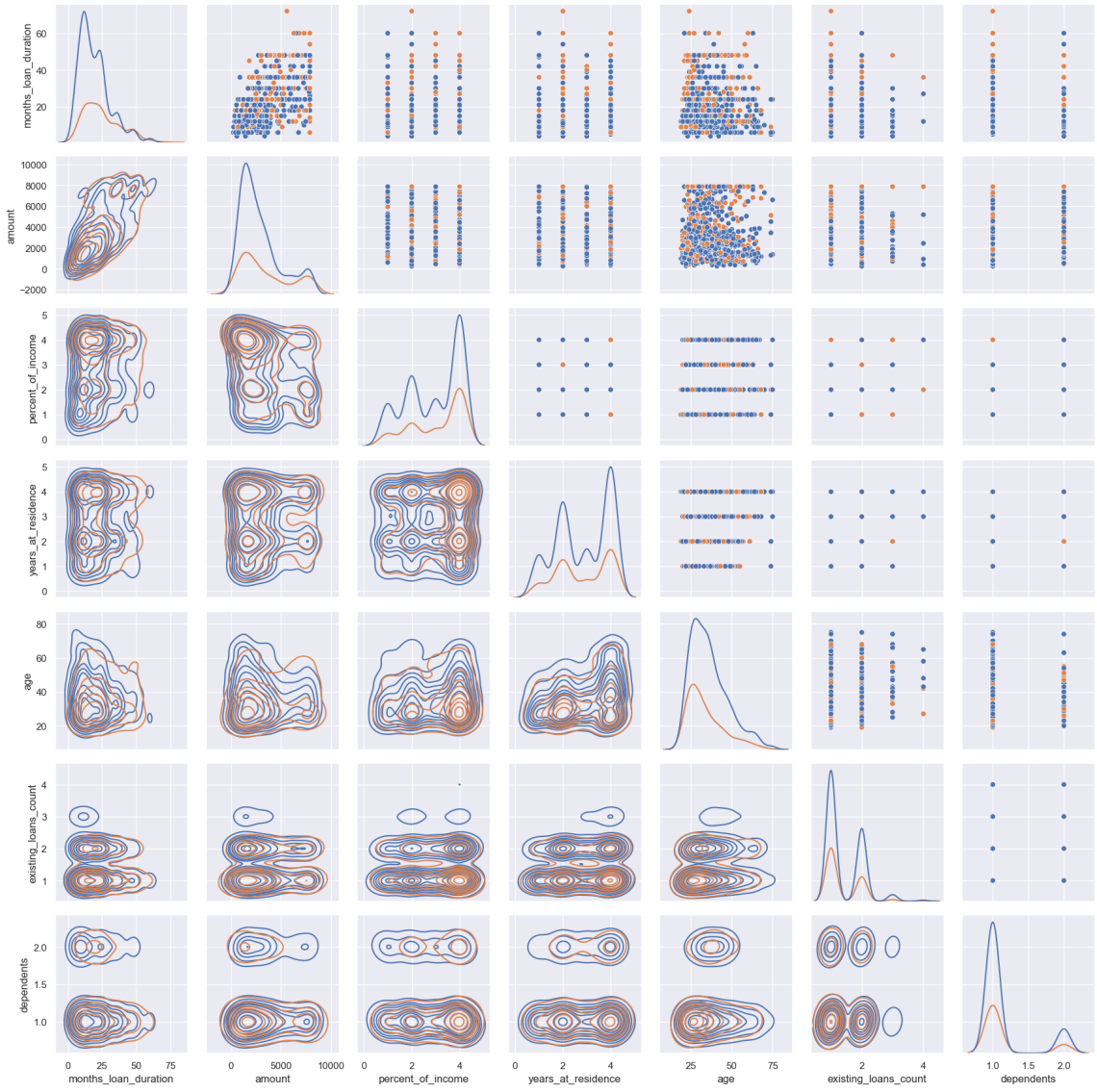
Out[17]:

<Axes: >



In [18]:

```
g = sns.PairGrid(df, hue='default' , diag_sharey=False)
g.map_upper(sns.scatterplot)
g.map_lower(sns.kdeplot)
g.map_diag(sns.kdeplot)
plt.show()
```

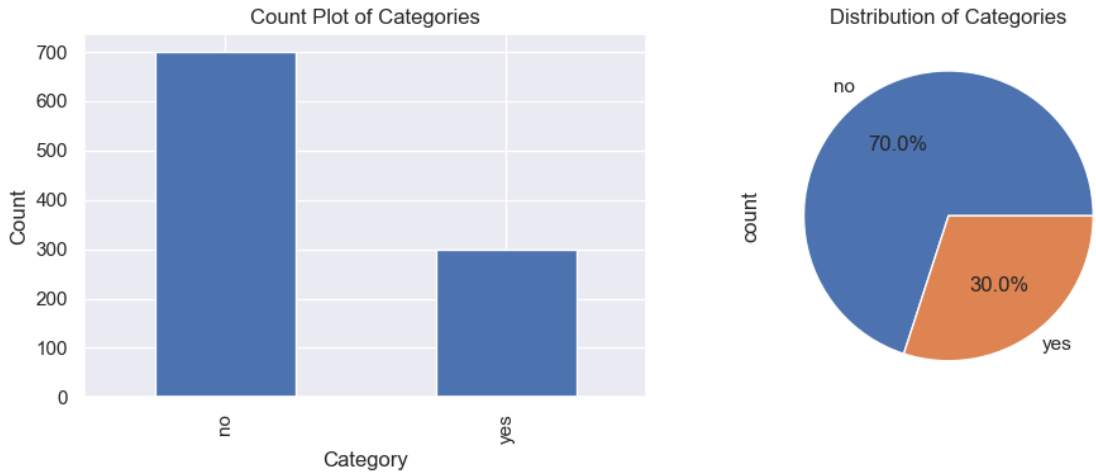


📊 Checking Imbalance between the different classes

In [19]:

```
target = 'default'
value_counts =df[target].value_counts()
print(value_counts)


fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# Plot the bar plot on the first subplot (axes[0])
value_counts.plot(kind='bar', ax=axes[0],)
axes[0].set_xlabel('Category')
axes[0].set_ylabel('Count')
axes[0].set_title('Count Plot of Categories')


value_counts.plot(
    kind='pie',
    ax=axes[1],
    #explode=[0.1, 0.1,0,0, 0, 0],     # Explode the second category to emphasize it
    autopct='%1.1f%%',         # Display percentage with one decimal place
    #shadow=True,               # Add a shadow effect to the chart
    #colors=['cyan', 'gray', 'orange', 'green']  # Custom colors for the pie chart
)
axes[1].set_title('Distribution of Categories')


plt.tight_layout()

plt.show()
```

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



** Imbalancy in dataset found**

- Most of the default status are having quality of **no**.

- A **class-imbalance** in the target feature is observed.
- To overcome this imbalancy, oversampling method is used for modeling.

🛠️🧱⛏️ Model Building Predict Data and Evaluation of Model

Sklearn Libraries

In [20]:

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix, roc
from sklearn.metrics import precision_score, recall_score, f1_score
from sklearn.datasets import make_classification

from sklearn.tree import DecisionTreeClassifier, plot_tree, export_text
from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, GradientBoosting
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
```

Encoding

1. changing column to category

In [21]:

```
# changing column to category
def col_name_unique_value(data):
    for col in data.columns:
        uniq_tot = data[col].unique()

        if  col in data.columns and data[col].nunique() <= 6:
            data[col]=data[col].astype('category')

col_name_unique_value(df)
print(df.dtypes)
```

```
checking_balance        category
months_loan_duration       int64
credit_history          category
purpose                 category
amount                   float64
savings_balance         category
employment_duration     category
percent_of_income       category
years_at_residence      category
age                        int64
other_credit            category
housing                 category
existing_loans_count    category
job                     category
dependents              category
phone                   category
default                 category
dtype: object
```

2. Making list of categorical column

In [22]:

```
all_cols = set(df.columns)
num_cols = set(df._get_numeric_data().columns)
cat_cols = list(all_cols - num_cols)
```

In [23]:

```
cat_cols
```

Out[23]:

```
['housing',
 'employment_duration',
 'existing_loans_count',
 'purpose',
 'percent_of_income',
 'checking_balance',
 'default',
 'dependents',
 'phone',
 'other_credit',
 'years_at_residence',
 'credit_history',
 'job',
 'savings_balance']
```

3. Label Encoding

In [24]:

```
for col in cat_cols:
    if df[col].nunique() <= 2:
        df[col] = df[col].cat.codes
        print(df[col].unique())
```

```
[0 1]
[0 1]
[1 0]
```

4. One-Hot Encoding

In [25]:

```
for col in cat_cols:
    if df[col].nunique() > 2:
        df =pd.get_dummies(df, columns= [col], drop_first = True)
```

In [26]:

```
df.dtypes
```

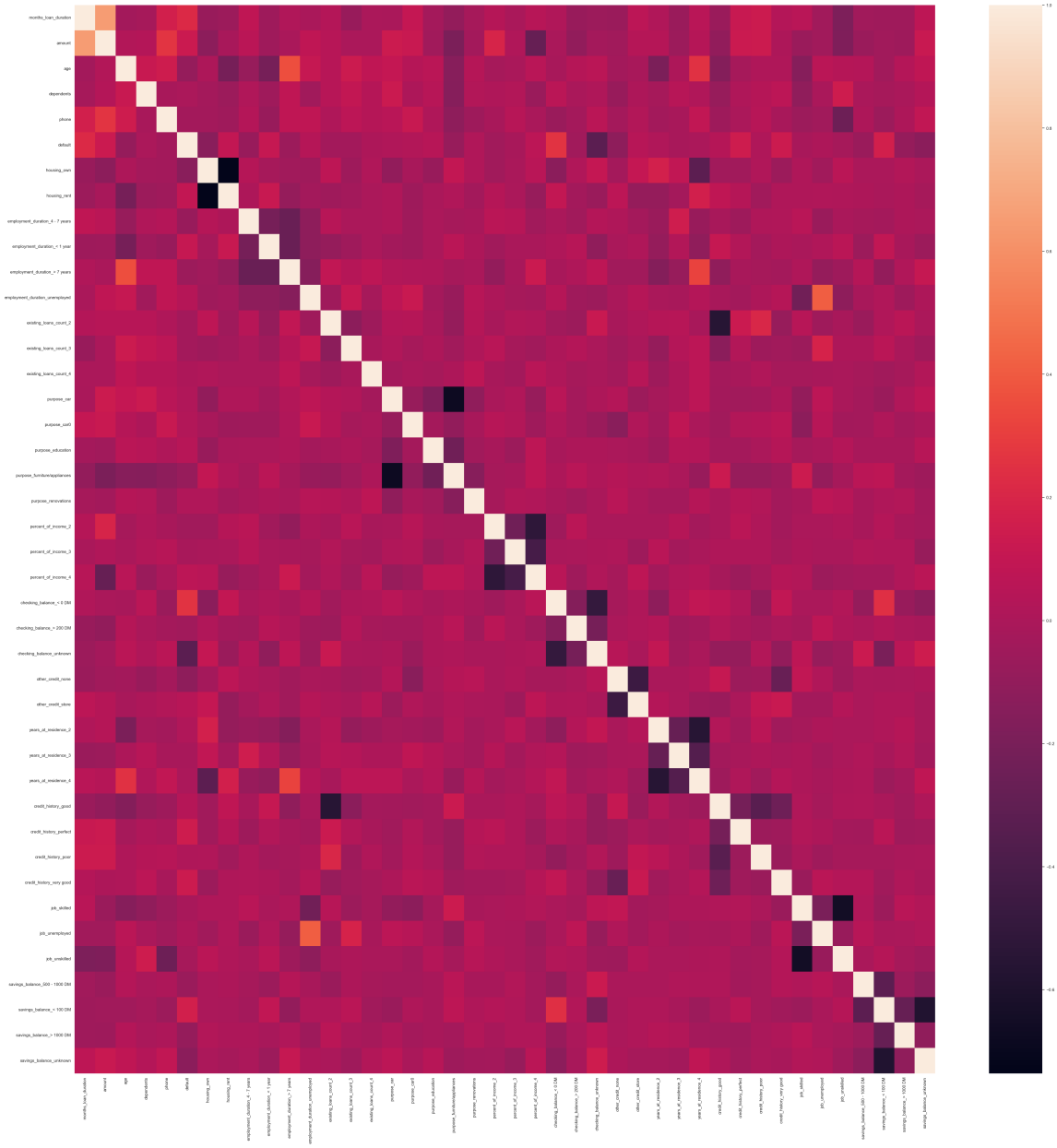
Out[26]:

```
months_loan_duration                int64
amount                            float64
age                                 int64
dependents                          int8
phone                               int8
default                             int8
housing_own                         bool
housing_rent                        bool
employment_duration_4 - 7 years     bool
employment_duration_< 1 year        bool
employment_duration_> 7 years       bool
employment_duration_unemployed      bool
existing_loans_count_2              bool
existing_loans_count_3              bool
existing_loans_count_4              bool
purpose_car                         bool
purpose_car0                        bool
purpose_education                   bool
purpose_furniture/appliances        bool
purpose_renovations                 bool
percent_of_income_2                 bool
percent_of_income_3                 bool
percent_of_income_4                 bool
checking_balance_< 0 DM             bool
checking_balance_> 200 DM           bool
checking_balance_unknown            bool
other_credit_none                   bool
other_credit_store                  bool
years_at_residence_2                bool
years_at_residence_3                bool
years_at_residence_4                bool
credit_history_good                 bool
credit_history_perfect              bool
credit_history_poor                 bool
credit_history_very good            bool
job_skilled                         bool
job_unemployed                      bool
job_unskilled                       bool
savings_balance_500 - 1000 DM       bool
savings_balance_< 100 DM            bool
savings_balance_> 1000 DM           bool
savings_balance_unknown             bool
dtype: object
```

Finding correlation

In [27]:

```
plt.figure(figsize=(50,50))
corr = df.corr()
sns.heatmap(corr,)
plt.show()
```



In [28]:

```
df.corr().loc['default'].round(2)
```

Out[28]:

```
months_loan_duration               0.21
amount                             0.13
age                               -0.09
dependents                        -0.00
phone                             -0.04
default                            1.00
housing_own                       -0.13
housing_rent                       0.09
employment_duration_4 - 7 years   -0.08
employment_duration_< 1 year       0.11
employment_duration_> 7 years     -0.06
employment_duration_unemployed     0.04
existing_loans_count_2            -0.04
existing_loans_count_3            -0.03
existing_loans_count_4             0.01
purpose_car                        0.02
purpose_car0                       0.03
purpose_education                  0.05
purpose_furniture/appliances      -0.08
purpose_renovations                0.02
percent_of_income_2               -0.04
percent_of_income_3               -0.01
percent_of_income_4                0.07
checking_balance_< 0 DM            0.26
checking_balance_> 200 DM         -0.04
checking_balance_unknown          -0.32
other_credit_none                 -0.11
other_credit_store                 0.05
years_at_residence_2               0.02
years_at_residence_3              -0.01
years_at_residence_4               0.00
credit_history_good                0.04
credit_history_perfect             0.14
credit_history_poor                0.01
credit_history_very good           0.13
job_skilled                       -0.01
job_unemployed                     0.01
job_unskilled                     -0.02
savings_balance_500 - 1000 DM     -0.07
savings_balance_< 100 DM           0.16
savings_balance_> 1000 DM         -0.09
savings_balance_unknown           -0.13
Name: default, dtype: float64
```

Splitting into target and independent features

In [29]:

```
target = "default"
x = df.drop(columns=target)
y = df[target]
```

Feature Scaling

In [30]:

```
#StandardScaler in dataframe
sc = StandardScaler()
sc_x = pd.DataFrame(sc.fit_transform(x) , columns=x.columns)
sc_x.head()
```

Out[30]:

	months_loan_duration	amount	age	dependents	phone	housing_own	housing
0	-1.236478	-0.860961	2.766456	-0.428290	1.214598	0.634448	-0.4
1	2.248194	1.326550	-1.191404	-0.428290	-0.823318	0.634448	-0.4
2	-0.738668	-0.436908	1.183312	2.334869	-0.823318	0.634448	-0.4
3	1.750384	2.209879	0.831502	2.334869	-0.823318	-1.576173	-0.4
4	0.256953	0.832050	1.535122	2.334869	-0.823318	-1.576173	-0.4

5 rows × 41 columns

Imbalance treatment

In [31]:

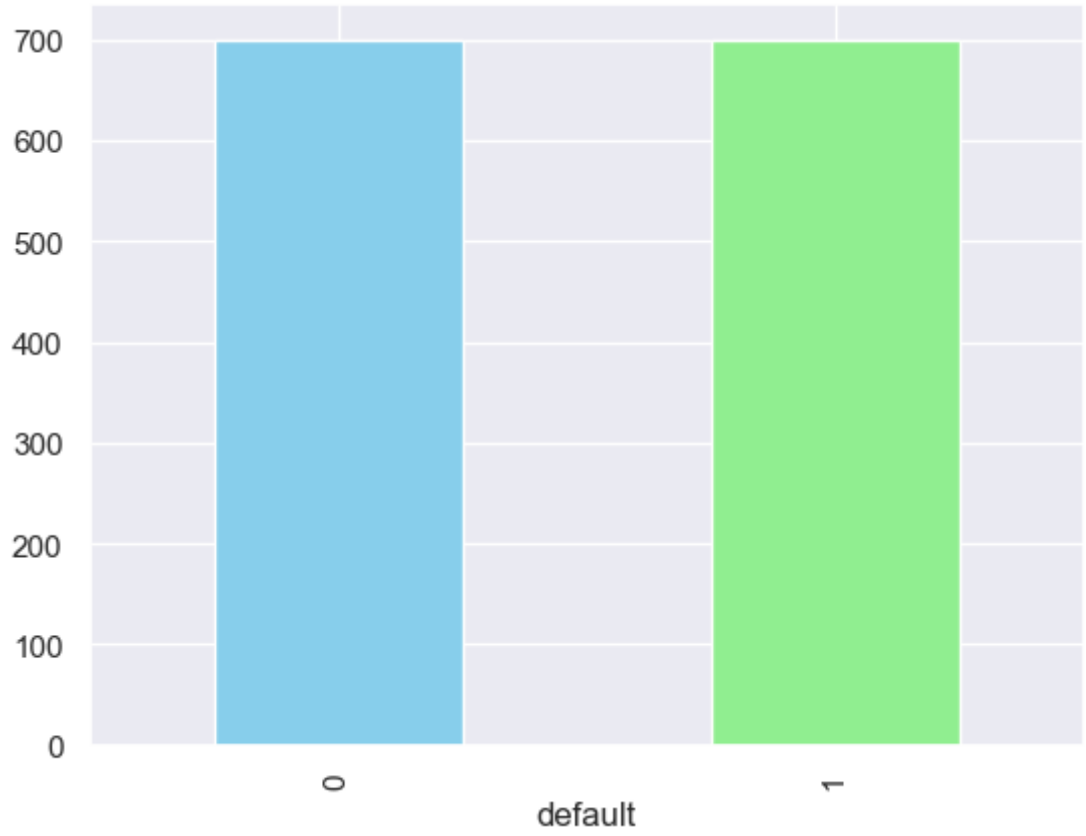
```
from imblearn.over_sampling import RandomOverSampler
over = RandomOverSampler()
x_over, y_over = over.fit_resample(sc_x,y)
```

In [32]:

```
y_over.value_counts().plot(kind='bar', color=['skyblue', 'lightgreen'])
```

Out[32]:

```
<Axes: xlabel='default'>
```



In [33]:

```
x_train,x_test,y_train,y_test = train_test_split(x_over,y_over,test_size=0.25,random_sta
```

Model comparision

In [34]:

```
m1 = LogisticRegression()
m2 = DecisionTreeClassifier(max_depth =11, criterion='log_loss' )
m3 = BaggingClassifier(estimator=RandomForestClassifier())
m4 = RandomForestClassifier(max_depth = 10, criterion = 'log_loss', random_state =123 )
m5 = SVC(kernel ='rbf', C =.3,)


models = [('LogisticRegression', m1),
          ('DecisionTreeClassifier', m2),
          ('BaggingClassifier', m3),
         ('RandomForestClassifier',m4),
          ('SupportVectorClassifier',m5)
        ]


from tabulate import tabulate
print(tabulate(models, headers=['Model name', 'Sklearn models']),)
```

```
Model name               Sklearn models
-----------------------  ------------------------------------------------
--------------------------
LogisticRegression       LogisticRegression()
DecisionTreeClassifier   DecisionTreeClassifier(criterion='log_loss', max_
depth=11)
BaggingClassifier        BaggingClassifier(estimator=RandomForestClassifie
r())
RandomForestClassifier   RandomForestClassifier(criterion='log_loss', max_
depth=10, random_state=123)
SupportVectorClassifier  SVC(C=0.3)
```

In [35]:

```
Train_Accuracy = []
Test_Accuracy=[]
Variance =[]
Train_CV=[]
Test_CV=[]
Variance_CV=[]
Precision_tr =[]
Recall_tr =[]
F1_score_tr =[]
Precision_test =[]
Recall_test =[]
F1_score_test =[]
TP=[]
FP=[]
FN=[]
TN=[]


for model_name, model in models:

    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    print('==='*23)
    print(f'           Model name: {model_name}')
    print('==='*23)
    print()

#=================================================
#            Model Accuracy
#-------------------------------------------------
    train_acc = accuracy_score(y_train, y_pred_train)
    test_acc = accuracy_score(y_test, y_pred_test)
#     Train_Accuracy.append(train_acc.round(2)*100)
#     Test_Accuracy.append(test_acc.round(2)*100)
#     variance = abs(train_acc - test_acc).round(2)*100
   # print(f'➽ Training Accuracy, {train_acc.round(2)*100} %' )
   # print(f'➽ TestAccuracy, {test_acc.round(2)*100} %' )
   # print(f'Variance : {variance}')

    #Variance.append(variance)
    #if (variance > 9) or  (train_acc > 99):
        #print(f'ATTENTION : The  {model_name} Model is overfitting')


    print()

#=========================================================
#            Cross Validation
#---------------------------------------------------------
    CV_train_acc =cross_val_score(model, x_train, y_train, cv =10).mean()
    CV_test_acc =cross_val_score(model, x_test, y_test, cv =10).mean()
    Train_CV.append(CV_train_acc.round(2)*100)
    Test_CV.append(CV_test_acc.round(2)*100)
    print(f'➽ Training Accuracy(CrossValidation), {CV_train_acc.round(2)*100} %' )
    print(f'➽ TestAccuracy(CrossValidation), {CV_test_acc.round(2)*100} %' )
    variance_CV = abs(CV_train_acc - CV_test_acc).round(2)*100
    Variance_CV.append(variance_CV)
    print(f'Variance (CrossValidation) : {variance_CV}')
```

```
    if (variance_CV > 9) or (CV_train_acc > 99):
        print(f'ATTENTION : The  {model_name} Model is overfitting')
        print()
#=========================================================
#           Evalutaion Metrics
#---------------------------------------------------------
#     print( '--'*30)
#     print( 'Classification_report:\n')
#     print( '--'*30)
#     print( 'Train:')
#     print( classification_report(y_train, y_pred_train))

#     print( 'Test:')
#     print( '--'*30)
#     print(classification_report(y_test, y_pred_test))
#     print( '--'*30)

    cm_train= confusion_matrix(y_train, y_pred_train)
    cm_test = confusion_matrix(y_test, y_pred_test)
    #print( '--'*30)
    #print( 'confusion_matrix_Train:\n',cm_train,'\nconfusion_matrix_Test :\n',cm_test )
    #print('--'*30)


    TP_test = cm_test[0,0]
    FP_test = cm_test[0,1]
    FN_test = cm_test[1,0]
    TN_test = cm_test[1,1]


    TP.append(TP_test)
    FP.append(FP_test)
    FN.append(FN_test)
    TN.append(TN_test)


model=[i[0] for i in models]
#print(len(Train_Accuracy), len(Test_Accuracy),len(Variance),len(Train_CV),len(Test_CV),
metrics={"Model":model, "Train accuracy(CV)":Train_CV,"Test accuracy(CV)":Test_CV,"Varia
          "TP":TP,"TN":TN,"FP":FP,"FN":FN}

metric_df = pd.DataFrame(metrics)
#print(metric_df.shape)
metric_df
#print(TN)
```

```
========================================================================
           Model name: LogisticRegression
========================================================================


- Training Accuracy(CrossValidation), 71.0 %
- TestAccuracy(CrossValidation), 70.0 %
Variance (CrossValidation) : 1.0
========================================================================
           Model name: DecisionTreeClassifier
========================================================================


- Training Accuracy(CrossValidation), 78.0 %
- TestAccuracy(CrossValidation), 71.0 %
Variance (CrossValidation) : 7.000000000000001
========================================================================
           Model name: BaggingClassifier
========================================================================


- Training Accuracy(CrossValidation), 82.0 %
- TestAccuracy(CrossValidation), 75.0 %
Variance (CrossValidation) : 8.0
========================================================================
           Model name: RandomForestClassifier
========================================================================


- Training Accuracy(CrossValidation), 82.0 %
- TestAccuracy(CrossValidation), 77.0 %
Variance (CrossValidation) : 6.0
========================================================================
           Model name: SupportVectorClassifier
========================================================================


- Training Accuracy(CrossValidation), 74.0 %
- TestAccuracy(CrossValidation), 71.0 %
Variance (CrossValidation) : 3.0
```

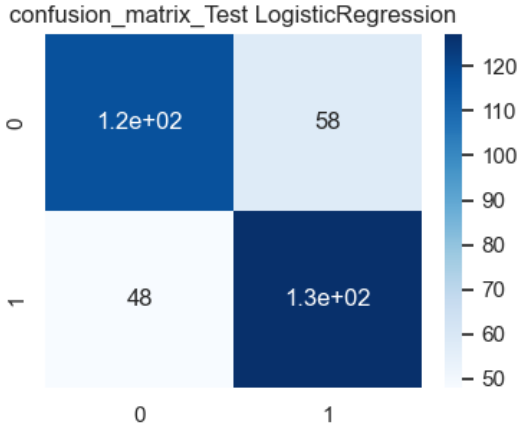
Out[35]:

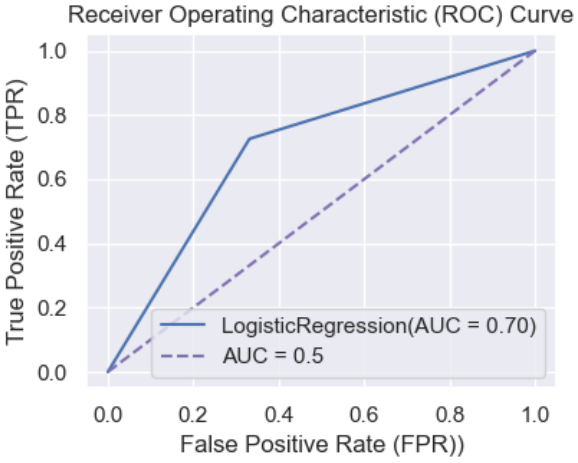
	Model	Train accuracy(CV)	Test accuracy(CV)	Variance_CV	TP	TN	FP	FN
0	LogisticRegression	71.0	70.0	1.0	117	127	58	48
1	DecisionTreeClassifier	78.0	71.0	7.0	123	149	52	26
2	BaggingClassifier	82.0	75.0	8.0	135	158	40	17
3	RandomForestClassifier	82.0	77.0	6.0	135	151	40	24
4	SupportVectorClassifier	74.0	71.0	3.0	112	144	63	31

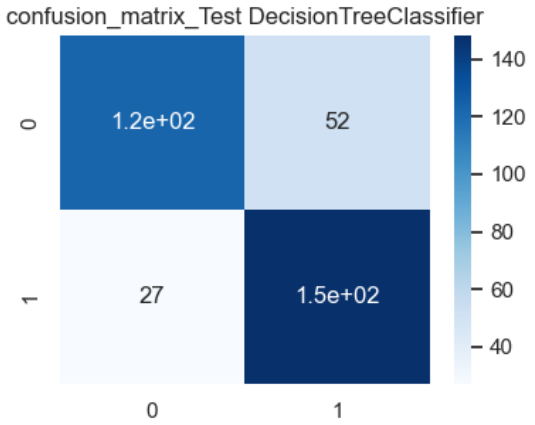
In [36]:

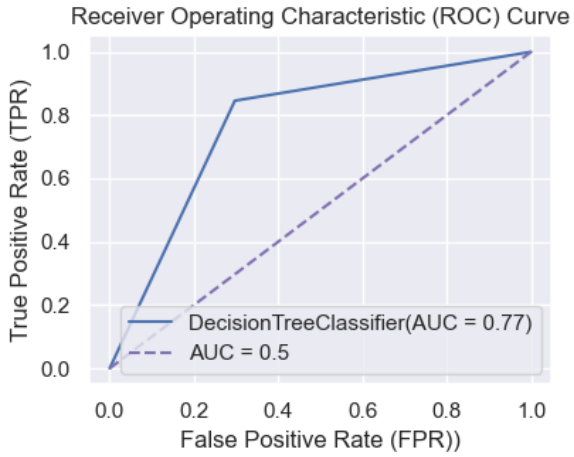
```
for model_name, model in models:
    model.fit(x_train,y_train)
    y_pred_train = model.predict(x_train)
    y_pred_test = model.predict(x_test)
    cm_test = confusion_matrix(y_test, y_pred_test)

    #y_test_binary = (y_test == 1).astype(int)
    #y_pred_test_binary = (y_pred_test == 1).astype(int)
    fpr, tpr, _ = roc_curve(y_test, y_pred_test, pos_label=1)
    roc_auc = auc(fpr, tpr)
    print('--'*15)
    print(model_name)
    print('--'*15)
    print()
    print(f'Test confusion matrix :\n {cm_test}')
    print()
    print(f'Test ROC-AUC : {roc_auc:.2f}')

    plt.figure(figsize = (8,3.5))
    plt.subplot(1,2,1)
    sns.heatmap(cm_test, annot = True, cmap= 'Blues')
    plt.title(f'confusion_matrix_Test {model_name}')

    plt.subplot(1,2,2)
    plt.plot(fpr, tpr, label=f'{model_name}(AUC = {roc_auc:.2f})')

# Plot the diagonal line representing a random classifier (AUC = 0.5)

    plt.plot([0, 1], [0, 1], 'm--', label='AUC = 0.5')

# Set labels and title

    plt.xlabel('False Positive Rate (FPR))')
    plt.ylabel('True Positive Rate (TPR)')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.tight_layout()
# Show the plot
plt.show()
```

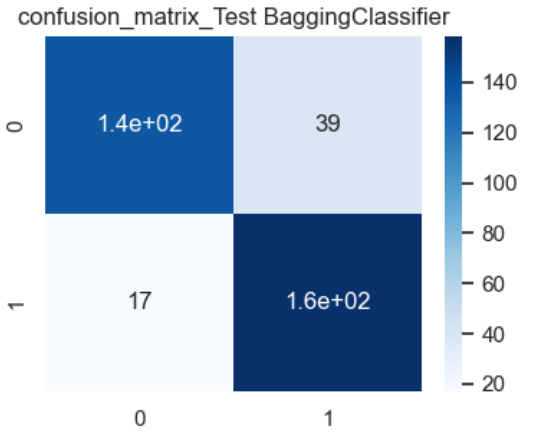
```
------------------------------
LogisticRegression
------------------------------

Test confusion matrix :
 [[117  58]
 [ 48 127]]

Test ROC-AUC : 0.70
------------------------------
DecisionTreeClassifier
------------------------------

Test confusion matrix :
 [[123  52]
 [ 27 148]]

Test ROC-AUC : 0.77
------------------------------
BaggingClassifier
------------------------------

Test confusion matrix :
 [[136  39]
 [ 17 158]]

Test ROC-AUC : 0.84
------------------------------
RandomForestClassifier
------------------------------

Test confusion matrix :
 [[135  40]
 [ 24 151]]

Test ROC-AUC : 0.82
------------------------------
SupportVectorClassifier
------------------------------

Test confusion matrix :
 [[112  63]
 [ 31 144]]

Test ROC-AUC : 0.73
```

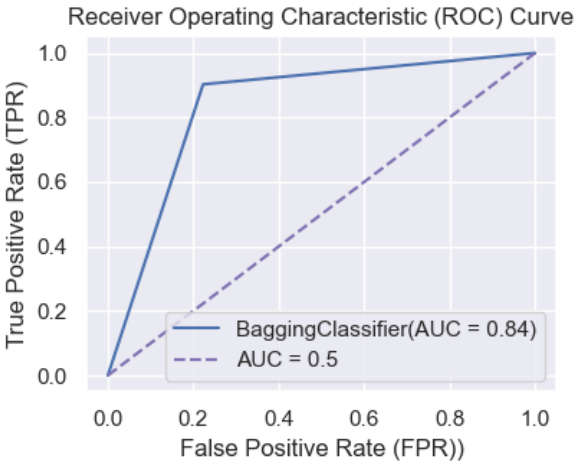


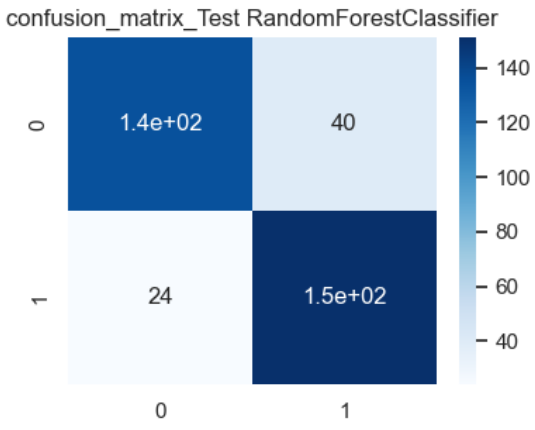


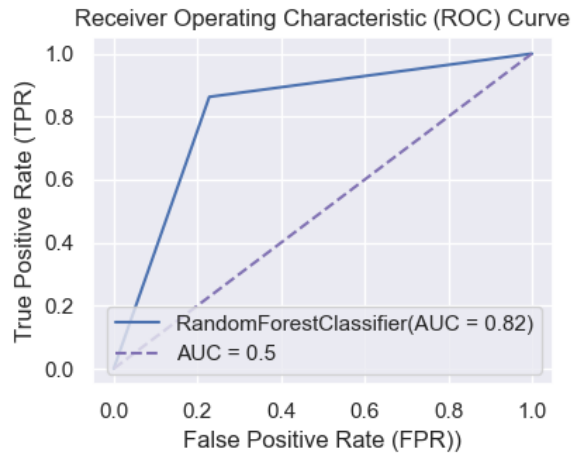

confusion_matrix_Test DecisionTreeClassifier
1.2e+02
52
27
1.5e+02
0
1
140
120
100
80
60
40

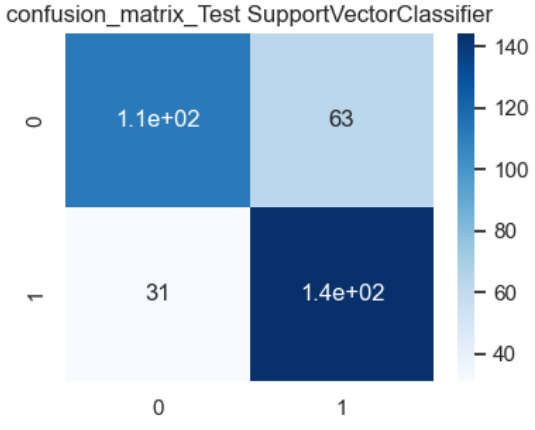

Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
DecisionTreeClassifier(AUC = 0.77)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0

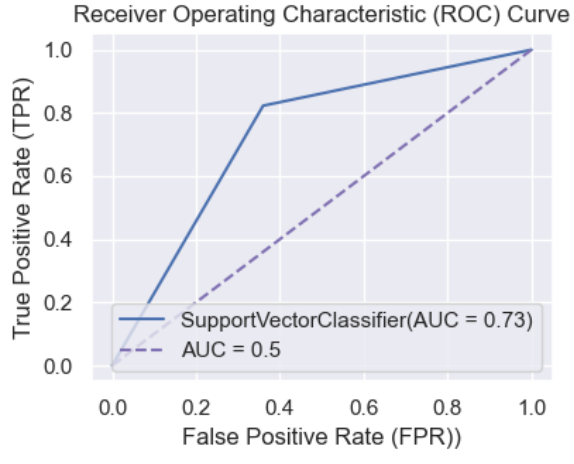

confusion_matrix_Test BaggingClassifier
1.4e+02
39
17
1.6e+02
0
1
140
120
100
80
60
40
20


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
BaggingClassifier(AUC = 0.84)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0


confusion_matrix_Test RandomForestClassifier
1.4e+02
40
24
1.5e+02
0
1
140
120
100
80
60
40


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
RandomForestClassifier(AUC = 0.82)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0


confusion_matrix_Test SupportVectorClassifier
1.1e+02
63
31
1.4e+02
0
1
140
120
100
80
60
40


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (TPR)
False Positive Rate (FPR))
SupportVectorClassifier(AUC = 0.73)
AUC = 0.5
0.0
0.2
0.4
0.6
0.8
1.0

==Conclusion==

* These following findings highlight the relative performance of different classification models on the German credit card dataset

Model	Train accuracy(CV)	Test accuracy(CV)	Variance_CV	Roc
LogisticRegression	71.0	70.0	1.0	0.70
DecisionTreeClassifier	78.0	71.0	7.0	0.77
BaggingClassifier	82.0	75.0	7.0	0.84
RandomForestClassifier	82.0	77.0	5.0	0.82
SupportVectorClassifier	74.0	71.0	3.0	0.73

- The Random Forest Classifier demonstrate stronger predictive capabilities compared to other models, as evidenced by its highest accuracy (77%) .

- Bagging Classifier excels in accurately classifying true instances, demonstrating a strong ability to identify true psotive (TP) and true negative (TN) and ROC-AUC scores (0.84).

- Payal Mohanty
- Project completed 7th August 2023