HW4 Part 1 (6 Points): Required Submissions:

1. Submit colab/jupyter notebooks.
2. Pdf version of the notebooks (HWs will not be graded if pdf version is not provided).
3. **The notebooks and pdf files should have the output.**
4. **Name files as follows : FirstNameLastName_HW5_Part1**

Question1 (6 Points) : Classification on the ‘credit-g’ dataset using KNN Classification

Import/Install the packages

```
if 'google.colab' in str(get_ipython()):
    print('Running on Colab')
else:
    print('Not Running on Colab')
```

```
Running on Colab
```

```
if 'google.colab' in str(get_ipython()):
  !pip install --upgrade feature_engine scikit-learn -q
  from google.colab import drive
  drive.mount('/content/drive')
```

Show hidden output

```
!pip show scikit-learn
```

```
Name: scikit-learn
Version: 1.3.1
Summary: A set of python modules for machine learning and data mining
Home-page: http://scikit-learn.org
Author:
Author-email:
License: new BSD
Location: /usr/local/lib/python3.10/dist-packages
Requires: joblib, numpy, scipy, threadpoolctl
Required-by: fastai, feature-engine, imbalanced-learn, librosa, mlxtend, qudida, sklearn-pandas, yellowbrick
```

```
!pip show feature_engine
```

```
Name: feature-engine
Version: 1.6.2
Summary: Feature engineering package with Scikit-learn's fit transform functionality
Home-page: http://github.com/feature-engine/feature_engine
Author: Soledad Galli
Author-email: solegalli@protonmail.com
License: BSD 3 clause
Location: /usr/local/lib/python3.10/dist-packages
Requires: numpy, pandas, scikit-learn, scipy, statsmodels
Required-by:
```

```
"""Importing the required packages"""

# For DataFrames and manipulations
import pandas as pd
import numpy as np

# For data Visualization
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.stats as stats
%matplotlib inline

# save and load models
import joblib

# Pathlib to navigate file system
```

```
from pathlib import Path
import sys

# For splitting the dataset
from sklearn.model_selection import train_test_split

# For categorical variables
from feature_engine.encoding import OneHotEncoder
from feature_engine.encoding import RareLabelEncoder

# For scaling the data
from sklearn.preprocessing import StandardScaler

# creating pipelines
from sklearn.pipeline import Pipeline

# Hyper parameter tuning
from sklearn.model_selection import GridSearchCV

# Using KNN classification for our data
from sklearn.neighbors import KNeighborsClassifier

# draws a confusion matrix
from sklearn.metrics import ConfusionMatrixDisplay

# We will use this to download the Dataset
from sklearn.datasets import fetch_openml

# feature engine log transformation
from feature_engine.transformation import LogTransformer

# feature engine wrapper
from feature_engine.wrappers import SklearnTransformerWrapper
```

Specify Project Folder Location

```
base_folder = Path('/content/drive/MyDrive/Applied_ML/Class_4/Assignment')
```

```
data_folder = base_folder/'Datasets'
save_model_folder = base_folder/'Model'
custom_function_folder = Path('/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Custom_function')
save_model_folder.mkdir(exist_ok=True, parents=True)
```

Import Custom Functions from Python file

```
%load_ext autoreload
%autoreload 2
```

```
sys.path.append(str(custom_function_folder))
```

```
sys.path
```

```
['/content',
 '/env/python',
 '/usr/lib/python310.zip',
 '/usr/lib/python3.10',
 '/usr/lib/python3.10/lib-dynload',
 '',
 '/usr/local/lib/python3.10/dist-packages',
 '/usr/lib/python3/dist-packages',
 '/usr/local/lib/python3.10/dist-packages/IPython/extensions',
 '/root/.ipython',
 '/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Custom_function']
```

```
from  plot_learning_curve import plot_learning_curve
```

```
from eda_plots import diagnostic_plots, plot_target_by_category
```

Download Data

You can download the dataset using the commands below and see it's description at https://www.openml.org/d/31

Attribute description from https://www.openml.org/d/31

1. Status of existing checking account, in Deutsche Mark.
2. Duration in months
3. Credit history (credits taken, paid back duly, delays, critical accounts)
4. Purpose of the credit (car, television,...)
5. Credit amount
6. Status of savings account/bonds, in Deutsche Mark.
7. Present employment, in number of years.
8. Installment rate in percentage of disposable income
9. Personal status (married, single,...) and sex
10. Other debtors / guarantors
11. Present residence since X years
12. Property (e.g. real estate)
13. Age in years
14. Other installment plans (banks, stores)
15. Housing (rent, own,...)
16. Number of existing credits at this bank
17. Job
18. Number of people being liable to provide maintenance for
19. Telephone (yes,no)
20. Foreign worker (yes,no)

```
# Load data from  https://www.openml.org/d/31
X, y = fetch_openml("credit-g", version=1, as_frame=True, return_X_y=True)
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/datasets/_openml.py:1022: FutureWarning: The default value of `parser` will c
  warn(
```

```
X.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitmen
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.

Exploratory data analysis

Check Data

Let's explore about the dataset by checking the shape(number of rows and columns), different column labels, duplicate values etc.

Check few rows

```
# check the top 5 rows
X.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitmen
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.

Check column names

```
# Let's check the columns of the data
X.columns
```

```
Index(['checking_status', 'duration', 'credit_history', 'purpose',
       'credit_amount', 'savings_status', 'employment',
       'installment_commitment', 'personal_status', 'other_parties',
       'residence_since', 'property_magnitude', 'age', 'other_payment_plans',
       'housing', 'existing_credits', 'job', 'num_dependents', 'own_telephone',
       'foreign_worker'],
      dtype='object')
```

Check data types of columns

```
# check the data type for the columns
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 20 columns):
 #   Column                  Non-Null Count  Dtype   
---  ------                  --------------  -----   
 0   checking_status         1000 non-null   category
 1   duration                1000 non-null   float64 
 2   credit_history          1000 non-null   category
 3   purpose                 1000 non-null   category
 4   credit_amount           1000 non-null   float64 
 5   savings_status          1000 non-null   category
 6   employment              1000 non-null   category
 7   installment_commitment  1000 non-null   float64 
 8   personal_status         1000 non-null   category
 9   other_parties           1000 non-null   category
 10  residence_since         1000 non-null   float64 
 11  property_magnitude      1000 non-null   category
 12  age                     1000 non-null   float64 
 13  other_payment_plans     1000 non-null   category
 14  housing                 1000 non-null   category
 15  existing_credits        1000 non-null   float64 
 16  job                     1000 non-null   category
 17  num_dependents          1000 non-null   float64 
 18  own_telephone           1000 non-null   category
 19  foreign_worker          1000 non-null   category
dtypes: category(13), float64(7)
memory usage: 69.9 KB
```

Check for unnecessary columns

Columns such as RowNumber, CustomerId, and Surname are unnecessary columns.

We don't need these columns for our analysis as these columns doesn't contain any pattern.

So, we can drop these columns.

Check for unique values

Now, let's see total number of unique values in each column.

```
# Identify Columns that Contain a Single value
# we will use nunique() function to get number of unique values for each column
# We can delete the columns which have single values
X.nunique()
```

```
checking_status             4
duration                   33
credit_history              5
purpose                    10
credit_amount             921
savings_status              5
employment                  5
installment_commitment      4
personal_status             4
other_parties               3
residence_since             4
property_magnitude          4
age                        53
other_payment_plans         3
housing                     3
existing_credits            4
job                         4
num_dependents              2
own_telephone               2
foreign_worker              2
dtype: int64
```

As we can see, all the columns has more than one unique value.

So, all these are valid and useful columns.

Check summary statistics

```
# We will use describe function and then take the transpose for better visualization
X.describe().T
```

	count	mean	std	min	25%	50%	75%	max
duration	1000.0	20.903	12.058814	4.0	12.0	18.0	24.00	72.0
credit_amount	1000.0	3271.258	2822.736876	250.0	1365.5	2319.5	3972.25	18424.0
installment_commitment	1000.0	2.973	1.118715	1.0	2.0	3.0	4.00	4.0
residence_since	1000.0	2.845	1.103718	1.0	2.0	3.0	4.00	4.0
age	1000.0	35.546	11.375469	19.0	27.0	33.0	42.00	75.0
existing_credits	1000.0	1.407	0.577654	1.0	1.0	1.0	2.00	4.0
num_dependents	1000.0	1.155	0.362086	1.0	1.0	1.0	1.00	2.0

Check for duplicate rows

```
# To check the duplicates of the data
dups = X.duplicated()
# report if there are any duplicates
print(dups.any())
```

```
False
```

From the given results, we can check that there are no duplicates in our data.

Quantifying Missing Data

Now, let's check is there any missing values in our dataframe.

```
# check missing values in data
X.isna()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitment	per
0	False	False	False	False	False	False	False	False	
1	False	False	False	False	False	False	False	False	
2	False	False	False	False	False	False	False	False	
3	False	False	False	False	False	False	False	False	
4	False	False	False	False	False	False	False	False	
...	...	...	...	...	...	...	...	...	
995	False	False	False	False	False	False	False	False	
996	False	False	False	False	False	False	False	False	
997	False	False	False	False	False	False	False	False	
998	False	False	False	False	False	False	False	False	
999	False	False	False	False	False	False	False	False	

1000 rows × 20 columns

```
# calculate % of mssing values for each column
X.isna().mean()*100
```

```
checking_status           0.0
duration                  0.0
credit_history            0.0
purpose                   0.0
credit_amount             0.0
savings_status            0.0
employment                0.0
installment_commitment    0.0
personal_status           0.0
other_parties             0.0
residence_since           0.0
property_magnitude        0.0
age                       0.0
other_payment_plans       0.0
housing                   0.0
existing_credits          0.0
job                       0.0
num_dependents            0.0
own_telephone             0.0
foreign_worker            0.0
dtype: float64
```

There are no missing values in the dataset.

Identify numerical, categorical and discrete variables

Since EDA steps can be different depending on type of variables. Let us first create list of different type of variables.

```
# Create a list of categorical variables
# Since the dtype of categorical variable is Object we can compare the values with 'O'
categorical = [var for var in X.columns if X[var].dtype.name == 'category']

# Create a list of discrete variables
# we do not want to consider Exited as this is target variable
discrete = [
    var for var in X.columns if X[var].dtype.name != 'category'
    and len(X[var].unique()) < 20
]

# Create a list of continuous Variables
continuous = [
    var for var in X.columns if X[var].dtype.name != 'category'
    if var not in discrete
]
```

```
# check continous Variables
continuous
```

```
['duration', 'credit_amount', 'age']
```

```
# check categorical variables
categorical
```

```
['checking_status',
 'credit_history',
 'purpose',
 'savings_status',
 'employment',
 'personal_status',
 'other_parties',
 'property_magnitude',
 'other_payment_plans',
 'housing',
 'job',
 'own_telephone',
 'foreign_worker']
```

```
# check discrete variables
discrete
```

```
['installment_commitment',
 'residence_since',
 'existing_credits',
 'num_dependents']
```

Check unique values for variables

```
# Check number of unique values for discrete variables
total_unique_values= X[discrete].nunique()
for key, value in total_unique_values.items():
    if value > 0:
        print(key,":",value)
```

```
installment_commitment : 4
residence_since : 4
existing_credits : 4
num_dependents : 2
```

```
# check values for discrete variables
for var in discrete:
    print(var, X[var].unique(), '\n')
```

```
installment_commitment [4. 2. 3. 1.]

residence_since [4. 2. 3. 1.]

existing_credits [2. 1. 3. 4.]

num_dependents [1. 2.]
```

```
# Check number of unique values for continuous variables
total_unique_values= X[continuous].nunique()
for key,value in total_unique_values.items():
    if value >0:
        print(key,":",value)
```

```
duration : 33
credit_amount : 921
age : 53
```

```
# check values for continuous variables
# we will check the first 20 values
for var in continuous:
    print(var, X[var].unique()[0:20], '\n')
```

```
duration [ 6. 48. 12. 42. 24. 36. 30. 15.  9. 10.  7. 60. 18. 45. 11. 27.  8. 54.
 20. 14.]

credit_amount [ 1169.  5951.  2096.  7882.  4870.  9055.  2835.  6948.  3059.  5234.
  1295.  4308.  1567.  1199.  1403.  1282.  2424.  8072. 12579.  3430.]

age [67. 22. 49. 45. 53. 35. 61. 28. 25. 24. 60. 32. 44. 31. 48. 26. 36. 39.
```

```
       42. 34.]
```

```
# Check number of unique values for categorical variables
total_unique_values= X[categorical].nunique()
for key,value in total_unique_values.items():
    if value >0:
        print(key,":",value)
```

```
checking_status : 4
credit_history : 5
purpose : 10
savings_status : 5
employment : 5
personal_status : 4
other_parties : 3
property_magnitude : 4
other_payment_plans : 3
housing : 3
job : 4
own_telephone : 2
foreign_worker : 2
```

```
# check values for categorical variables
for var in categorical:
    print(var, X[var].unique(), '\n')
```

```
checking_status ['<0', '0<=X<200', 'no checking', '>=200']
Categories (4, object): ['0<=X<200', '<0', '>=200', 'no checking']

credit_history ['critical/other existing credit', 'existing paid', 'delayed previously', 'no credits/all paid', 'all paid']
Categories (5, object): ['all paid', 'critical/other existing credit', 'delayed previously',
                         'existing paid', 'no credits/all paid']

purpose ['radio/tv', 'education', 'furniture/equipment', 'new car', 'used car', 'business', 'domestic appliance', 'repairs',
Categories (10, object): ['business', 'domestic appliance', 'education', 'furniture/equipment',
                          ..., 'radio/tv', 'repairs', 'retraining', 'used car']

savings_status ['no known savings', '<100', '500<=X<1000', '>=1000', '100<=X<500']
Categories (5, object): ['100<=X<500', '500<=X<1000', '<100', '>=1000', 'no known savings']

employment ['>=7', '1<=X<4', '4<=X<7', 'unemployed', '<1']
Categories (5, object): ['1<=X<4', '4<=X<7', '<1', '>=7', 'unemployed']

personal_status ['male single', 'female div/dep/mar', 'male div/sep', 'male mar/wid']
Categories (4, object): ['female div/dep/mar', 'male div/sep', 'male mar/wid', 'male single']

other_parties ['none', 'guarantor', 'co applicant']
Categories (3, object): ['co applicant', 'guarantor', 'none']

property_magnitude ['real estate', 'life insurance', 'no known property', 'car']
Categories (4, object): ['car', 'life insurance', 'no known property', 'real estate']

other_payment_plans ['none', 'bank', 'stores']
Categories (3, object): ['bank', 'none', 'stores']

housing ['own', 'for free', 'rent']
Categories (3, object): ['for free', 'own', 'rent']

job ['skilled', 'unskilled resident', 'high qualif/self emp/mgmt', 'unemp/unskilled non res']
Categories (4, object): ['high qualif/self emp/mgmt', 'skilled', 'unemp/unskilled non res', 'unskilled resident']

own_telephone ['yes', 'none']
Categories (2, object): ['none', 'yes']

foreign_worker ['yes', 'no']
Categories (2, object): ['no', 'yes']
```

Check Variable Distributions

Categorical Varibles

Frequency distribution of categorical variables and rare categories

```
def check_rare(var):
  cat_freq = 100 * X[var].value_counts(normalize=True)
  fig = cat_freq.sort_values(ascending=False).plot.bar()
  fig.axhline(y=5, color='red')
  fig.set_ylabel('category percentage frequency')
  fig.set_xlabel(var)
  fig.set_title('Identifying Rare Categories')
  plt.show()
```

```
for var in categorical:
  check_rare(var)
```

From the above graph, we can see that we need to do rare label emcoding for following variables: `'credit_history', 'purpose', 'savings_status', 'personal_status', 'other_parties', other_payment_plans', 'job'`

Check distribution of target variable

```
print(f"{100 * y.value_counts(normalize=True)} ")
```

```
good    70.0
bad     30.0
```

```
Name: class, dtype: float64
```

From the above analysis, we can observe that 70% of the the data is classified as good credit risk and 30% of the data is classified as bad credit risk. **We can see that the dataset is imbalanced** i.e. we have far more observation from one class or label. We will see how to address this issue later in the course.

Distribution of continuous and discrete variables

We can use histograms, Q-Q plots, and Boxplots to check the distribution of continuous variables.

We created this function in last lecture. We have added this function in python file eda_plots. We have imorted the function and will use it now.

Distribution of continuous variables

```
for var in continuous:
    diagnostic_plots(X, var)
```

All the three continuous variables are skewed. None of these variables have zero or negative values. We can use any of the following transformations - logartithmic, yeo-johnson or boxplot for these variabes.

Distribution of discrete variables

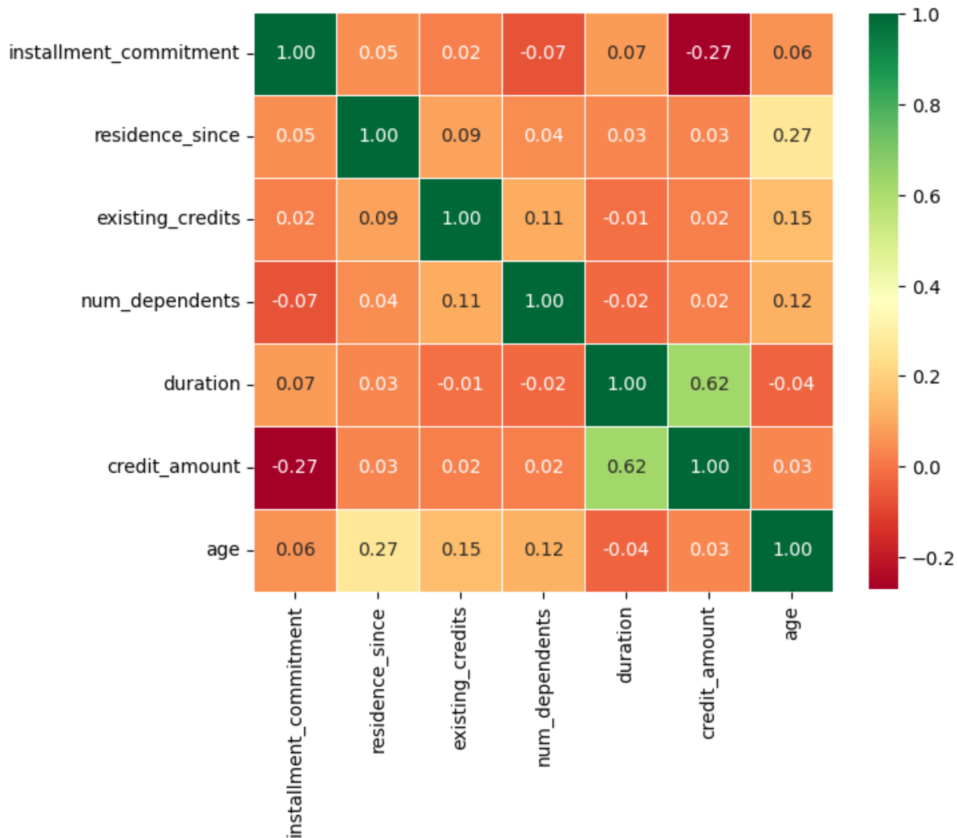
```
# histograms for discrete variables
for var in discrete:
    plt.figure(figsize=(12, 4))
    plt.subplot(1, 1, 1)
    sns.histplot(X[var], bins=30)
    plt.title('Histogram')
```

From the above graphs, it seems that these variables were continuos varibles. These have allredy been have been discretized uisng equal width method.

Visualizing Relationships between variables

Correlation Matrix

```
# We can check the correlation between every pair of attributes
# The correlation will be generated for the numerical data only
# we will use df.corr() to get correlatons and then use sns.heatmap to print the correlation matrix
corrmat = X[discrete + continuous].corr().round(2)
top_corr_features = corrmat.index
plt.figure(figsize=(7, 7))
sns.heatmap(X[top_corr_features].corr(),annot=True, square=True, fmt='.2f',
            cbar_kws={"shrink": .80}, linewidths=.5, cmap='RdYlGn');
```



- None of the correlations are too high in this dataset.

Relationship between Target variable and categorical variables

```
df = pd.concat([X, y], axis = 1)
```

```
df.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitmen
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.

5 rows × 21 columns

```
df.rename({'class': 'target'}, axis = 1, inplace=True)
```

```
df['target'] = df['target'].map({'good':0, 'bad':1})
```

```
df = df.astype({'target': 'int32'})
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 21 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   checking_status         1000 non-null   category
 1   duration                1000 non-null   float64
 2   credit_history          1000 non-null   category
 3   purpose                 1000 non-null   category
 4   credit_amount           1000 non-null   float64
 5   savings_status          1000 non-null   category
 6   employment              1000 non-null   category
 7   installment_commitment  1000 non-null   float64
 8   personal_status         1000 non-null   category
 9   other_parties           1000 non-null   category
 10  residence_since         1000 non-null   float64
 11  property_magnitude      1000 non-null   category
 12  age                     1000 non-null   float64
 13  other_payment_plans     1000 non-null   category
 14  housing                 1000 non-null   category
 15  existing_credits        1000 non-null   float64
 16  job                     1000 non-null   category
 17  num_dependents          1000 non-null   float64
 18  own_telephone           1000 non-null   category
 19  foreign_worker          1000 non-null   category
 20  target                  1000 non-null   int32
dtypes: category(13), float64(7), int32(1)
memory usage: 73.8 KB
```

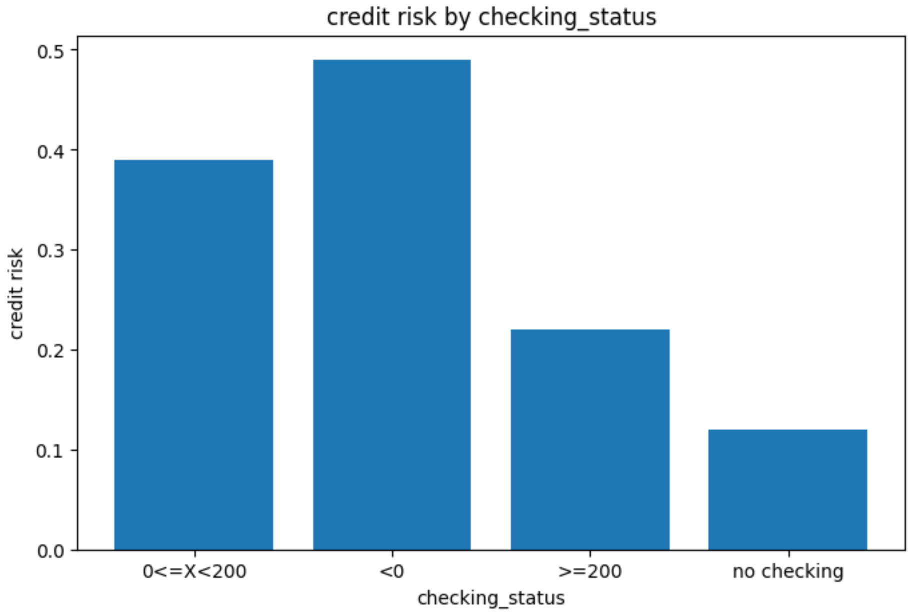
```
df.head()
```

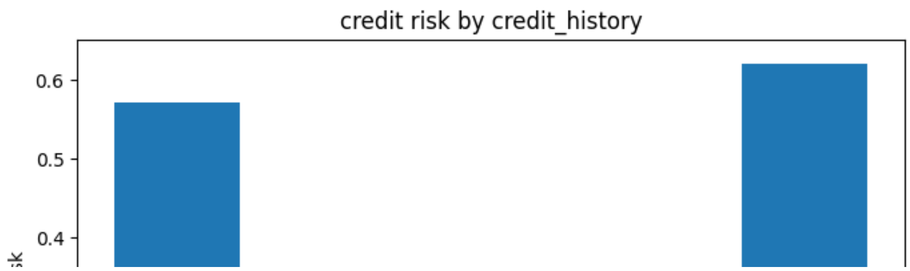
	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitmen
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no known savings	>=7	4.
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	<100	1<=X<4	2.
2	no checking	12.0	critical/other existing credit	education	2096.0	<100	4<=X<7	2.
3	<0	42.0	existing paid	furniture/equipment	7882.0	<100	4<=X<7	2.
4	<0	24.0	delayed previously	new car	4870.0	<100	1<=X<4	3.

5 rows × 21 columns

Now, let's plot a bar-plot of each categorical variable w.r.t. churn rate of each category.

```
# Plotting all categorical and discrete features using above function.
for category in categorical + discrete:
  plot_target_by_category(df, 'target',category,'credit risk')
```





Conclusion from EDA

Conclusions:

1. We do not have any missing values or single value columns
2. We have categorical variables in the data frame that should be converted into numerical before we run our model. Hence we need to do encoding of categorical variables.
3. Further, we need to do rare label encoding for following variables: 'credit_history', 'purpose', 'savings_status', 'personal_status', 'other_parties', other_payment_plans', 'job'.
4. From distributions of age, duration and amount, we can see that all three of these variables are skewed. We need to do transformation (like logarithmic, boxplot or yeojohnson) for these variables.
5. From the graphs of discrete variables, it seems that these variables were continuos varibles. These have allredy been have been discretized uisng equal width method.
6. From the relationship between target variables and discrete variables - we can see that for certain variables the relationship is not monotonous. In this case onehot encoding or mean encoding/decision tree encoding might help.
7. Finally we will need to make sure that the continuos variables have same scale. We will need to do feature scaling for continuos variables and discrete variables (if we do mean encoding).

Preprocessing Steps:

Pipeline 1:

1. Rare label encoding for categorical .
2. One hot encoding for categorical + discrete variables
3. Log transformation for continuos variables
4. Scaling for continuous variables.

Pipeline 2: Replace One hot enoding with Decision Tree encoding for categorical and discrete variables.

Complete Pipeline

Split Data

```
# before doing any data cleaning step we need to first split the data into train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=123, stratify =y)
```

```
X_train.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savings_status	employment	installment_commitm
356	no checking	12.0	critical/other existing credit	radio/tv	2331.0	no known savings	>=7	
344	>=200	10.0	existing paid	new car	3949.0	<100	<1	
236	0<=X<200	6.0	existing paid	new car	14555.0	no known savings	unemployed	
699	>=200	15.0	existing paid	education	1905.0	<100	>=7	
424	0<=X<200	12.0	existing paid	furniture/equipment	2762.0	no known savings	>=7	

```
y_train
```

```
356    good
344    good
236     bad
699    good
424     bad
       ...
124     bad
923    good
362    good
217    good
200    good
Name: class, Length: 670, dtype: category
Categories (2, object): ['bad', 'good']
```

```
print(f'Length of X_train: {len(X_train)}')
print(f'Length of X_test: {len(X_test)}')
```

```
Length of X_train: 670
Length of X_test: 330
```

```
print(f'Length of y_train: {len(y_train)}')
print(f'Length of y_test: {len(y_test)}')
```

```
Length of y_train: 670
Length of y_test: 330
```

Pipeline 1

Create a pipeline with following steps:

1. 'rare_label_encoder', variables = var_rare_labels
2. 'One_hot_encoding', variables= categorical+discrete,
3. 'log_transformer', variables = continuous
4. 'scalar',StandardScaler(), variables = continuous
5. 'convert_to_numpy', ConvertToNumpyArray(), all variables
6. KNeighborsClassifier())

```
var_rare_labels= [
 'credit_history',
 'purpose',
 'savings_status',
 'personal_status',
 'other_parties',
 'other_payment_plans',
```

```
 'job',
]
```

```
continuous
```

```
['duration', 'credit_amount', 'age']
```

```
discrete
```

```
['installment_commitment',
 'residence_since',
 'existing_credits',
 'num_dependents']
```

```
categorical
```

```
['checking_status',
 'credit_history',
 'purpose',
 'savings_status',
 'employment',
 'personal_status',
 'other_parties',
 'property_magnitude',
 'other_payment_plans',
 'housing',
 'job',
 'own_telephone',
 'foreign_worker']
```

TASK1 - Create the pipeline using feature engine. (1 Point)

```
from sklearn.base import BaseEstimator, TransformerMixin
class ConvertToNumpyArray(BaseEstimator, TransformerMixin):
    def __init__(self):
        pass

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return np.array(X)
```

```
credit_risk_pipeline_1 = Pipeline([

    ('rare_label_encoder',
      RareLabelEncoder(n_categories=2, variables=var_rare_labels,ignore_format=True)),

    ('one_hot_encoder',
     OneHotEncoder(variables=categorical+discrete,drop_last=True,ignore_format=True)),

    ('log_transformer',
     LogTransformer(variables=continuous) ),

    ('scalar',
      SklearnTransformerWrapper(StandardScaler(),variables=continuous)),

    ('convert_to_numpy', ConvertToNumpyArray()),

    ('knn',
     KNeighborsClassifier())
])
```

TASK2: Hyperparameter Tuning - Round 1 (1 Point)

```
# You will now create the paramtyer grid and do gridsearch. You will only tune n_neighbors of KNeighborsClassifier.
# In the first round use values from 1 to 5 (both included). Use step size of 1.

param_grid_1 = {
    'knn__n_neighbors' : np.arange(1,6,1)
}
```

```
# now we set up the grid search with cross-validation
grid_knn_1 = GridSearchCV(credit_risk_pipeline_1,param_grid=param_grid_1,cv=5,return_train_score=True)
```

```
# fit the grid on training data
grid_knn_1.fit(X_train,y_train)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ RareLabelEncoder
▸ OneHotEncoder
▸ LogTransformer
▸ scalar: SklearnTransformerWrapper
▸ transformer: StandardScaler
▸ StandardScaler
▸ ConvertToNumpyArray
▸ KNeighborsClassifier
```

```
# check the best_parameters from GridSearchCv for your model
print(grid_knn_1.best_params_)
print(grid_knn_1.best_score_)
```

```
{'knn__n_neighbors': 5}
0.7104477611940299
```

```
# Here  save_model_folder is folder where I have saved models. Change that to appropriate location.
# This variable is defined in section Mount Google Drive, Import Data

# specify the file to save the best estimator
file_best_estimator_round1 = save_model_folder / 'knn_round1_best_estimator.pkl'

# specify the file to save complete grid results
file_complete_grid_round1 = save_model_folder / 'knn_round1_complete_grid.pkl'
```

```
# save the best estimator
joblib.dump(grid_knn_1.best_estimator_, file_best_estimator_round1)

# save complete grid results
joblib.dump(grid_knn_1, file_complete_grid_round1)
```

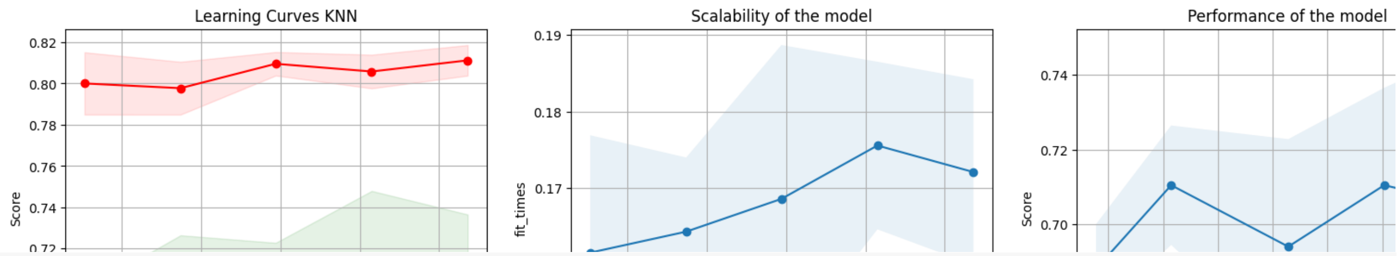
```
['/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Model/knn_round1_complete_grid.pkl']
```

```
# load  the best estimator
loaded_best_estimator_round1 = joblib.load(file_best_estimator_round1)

# load complete grid results
loaded_complete_grid_round1 = joblib.load(file_complete_grid_round1)
```

```
# plot learning curves
# Notice that we are using the best estimator
plot_learning_curve(loaded_best_estimator_round1, 'Learning Curves KNN', X_train, y_train, n_jobs=-1)
```

```
<module 'matplotlib.pyplot' from '/usr/local/lib/python3.10/dist-packages/matplotlib/pyplot.py'>
```




```
# Check the train scores
# to check the train scores we will use loaded best estimator

print(loaded_best_estimator_round1.score(X_train,y_train))

# check the cross validation score
# To check the cross validation score we need to use the loaded complete grid results

print(loaded_complete_grid_round1.best_score_)
```

```
0.8059701492537313
0.7104477611940299
```

- **What are your conclusions from Learning curves?**
- If the model is underfitting then we do not need round 2. This is because if it is underfitting then we need to increase model complexity. Since we are using n_neighbors from 1 to 5 (both included), we cannot increase model complexity further.
- If model is overfitting then specify higher ranges of n_neighbors. Use values of n_neighbors from 6 to 20 in next round
-

Task3: Hyperparameter Tuning - Round 2 (1 Point)

- If round2 is required then repeat all the steps of round 1. **Again only tune n_neighbors**. Use values in the range (6, 20). Use step size of 1.
- Report your conclusions from this round

```
from sklearn.preprocessing import LabelEncoder


labelencoder = LabelEncoder()
y_train_processed = labelencoder.fit_transform(y_train)
y_test_processed = labelencoder.transform(y_test)
```

```
param_grid_2 = {'knn__n_neighbors':np.arange(6,21,1)}
grid_knn_2 = GridSearchCV(credit_risk_pipeline_1,param_grid=param_grid_2, cv= 5 , return_train_score=True)

grid_knn_2.fit(X_train,y_train_processed)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
▸ RareLabelEncoder
▸ OneHotEncoder
▸ LogTransformer
▸ scalar: SklearnTransformerWrapper
▸ transformer: StandardScaler
▸ StandardScaler
▸ ConvertToNumpyArray
▸ KNeighborsClassifier
```

```
print(grid_knn_2.best_params_)
print(grid_knn_2.best_score_)
print(grid_knn_2.score(X_train,y_train_processed))
```

```
{'knn__n_neighbors': 8}
0.7283582089552239
0.8014925373134328
```

Task4: Pipeline2 (1 Point)

In this round, we will use a different pipeline. Create a pipeline with following steps:

1. 'rare_label_encoder' for categorical variables
2. 'DecisionTree_Encoder_encoder', variables= categorical+discrete,
3. 'log_transformer', variables = continuous
4. 'scalar',StandardScaler(), variables = continuous
5. 'convert_to_numpy', ConvertToNumpyArray(), all variables
6. KNeighborsClassifier())

Pipeline2

```
from feature_engine.encoding.decision_tree import DecisionTreeEncoder
from seaborn.categorical import variable_type
from feature_engine.discretisation.decision_tree import DecisionTreeClassifier


credit_risk_pipeline_2 = Pipeline([

    ('rare_label_encoder', RareLabelEncoder(n_categories=2,variables = var_rare_labels, ignore_format=True)),
    ('DecisionTree_Encoder_encoder',DecisionTreeEncoder(variables = categorical + discrete, ignore_format= True, regression = Fals
    ('log_transformer',LogTransformer(variables=continuous)),
    ('scalar',SklearnTransformerWrapper(StandardScaler(),variables=continuous)),
    ('convert_to_numpy',ConvertToNumpyArray()),
    ('knn',KNeighborsClassifier())


])
```

```
y_train.info()
```

```
<class 'pandas.core.series.Series'>
Int64Index: 670 entries, 356 to 200
Series name: class
Non-Null Count  Dtype
--------------  -----
670 non-null    category
dtypes: category(1)
memory usage: 6.0 KB
```

Task5: Hyperparameter Tuning - Round 3 (1 Point)

- Repeat all the steps of round 1. **Again only tune n_neighbors**. Use values in the range (1, 10). Use step size of 1.
- Report your conclusion from this round.

```
param_grid_3 ={'knn__n_neighbors':np.arange(1,11,1)}
```

```
grid_knn_3 = GridSearchCV(credit_risk_pipeline_2,param_grid=param_grid_3,cv=5,return_train_score=True)
grid_knn_3.fit(X_train,y_train_processed)
```

```
▸ GridSearchCV
▸ estimator: Pipeline
  ▸ RareLabelEncoder
  ▸ DecisionTreeEncoder
  ▸ LogTransformer
```

```
print(grid_knn_3.best_params_)
print(grid_knn_3.score(X_train,y_train_processed))
```

```
{'knn__n_neighbors': 9}
0.753731343283582
```

```
file_best_param = save_model_folder/"grid_knn_3_best_estimator.pkl"
file_best_grid = save_model_folder/"grid_knn_3_complete_grid.pkl"
```

```
joblib.dump(grid_knn_3.best_params_,file_best_param)
joblib.dump(grid_knn_3,file_best_grid)
```

```
['/content/drive/MyDrive/Applied_ML/Class_4/Assignment/Model/grid_knn_3_complete_grid.pkl']
```

```
loaded_best_param = joblib.load(file_best_param)
loaded_best_grid = joblib.load(file_best_grid)
```

```
print(loaded_best_grid.score(X_train,y_train_processed))
print(loaded_best_grid.best_score_)
```

```
0.753731343283582
0.7149253731343284
```

We will use pipeline 1 (Round 2) as it has better parameter score with values in range etween 6,20 included with step of 1

Task6: Perfromnace on Test Data (1 Point)

```
# check the test scores for final model
# Compare the cross validation score of round1, round2, and round3.
# Whichever round has best cross validation score, use the best estimator from that round to predict the test scores
print(f'Test data accaracy for round 2: { grid_knn_2.score(X_test,y_test_processed)}')
```

```
Test data accaracy for round 2: 0.7181818181818181
```

```
# use the best estimator selected in previous step to plot the confusion matrix
ConfusionMatrixDisplay.from_estimator(grid_knn_2, X_test, y_test_processed,
                                 display_labels=['Bad', 'Good'],
                                 cmap=plt.cm.Blues,
                                 normalize = 'true')
plt.grid(False)
plt.show()
```

Report the conclusion from confusion matrix.

For Good we see 85% accuracy but for Bad we see only 41% accuracy