HW 3 Supervised Learning (Total Points - 5)

You have to submit two files for this part of the HW

1. FirstNameLastName_Hw3.ipynb (colab notebook)
2. FirstNameLastName_Hw3.pdf pdf file**

For Task1: You are also provided HW3_EDA_Template. You can folllow this template to complete Task1.

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')
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 326.6/326.6 kB 2.8 MB/s eta 0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 10.8/10.8 MB 16.7 MB/s eta 0:00:00
Mounted at /content/drive
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call dr
```

```
import feature_engine
import sklearn
```

```
print(feature_engine.__version__)
print(sklearn.__version__)
```

```
1.6.1
1.3.0
```

```
# Import packages for data manipulation and mathematical operations
import pandas as pd  # For data manipulation using dataframes
import numpy as np  # For mathematical operations

# Import packages for data visualization
import matplotlib.pyplot as plt  # For basic plots
import seaborn as sns  # For more advanced plots
import scipy.stats as stats  # For statistical tests and transformations
# To display plots inline in Jupyter Notebooks
%matplotlib inline

# Import packages for saving and loading machine learning models
import joblib  # To save and load ML models

# Import packages for file and system operations
from pathlib import Path  # For OS-agnostic file paths
import sys  # For system-specific parameters and functions

# Import packages for data splitting and model evaluation
from sklearn.model_selection import train_test_split  # For data splitting
from sklearn.model_selection import StratifiedKFold  # For stratified cross-validatio
from sklearn.model_selection import KFold  # For simple cross-validation

# Import packages for data preprocessing
from feature_engine.encoding import OneHotEncoder  # For one-hot encoding categorical
from feature_engine.encoding import RareLabelEncoder  # For encoding rare labels
from sklearn.preprocessing import StandardScaler  # For standardizing numerical varia

# Import packages for building pipelines
from sklearn.pipeline import Pipeline  # For creating pipelines

# Import packages for hyperparameter tuning
from sklearn.model_selection import GridSearchCV  # For grid search cross-validation

# Import packages for machine learning algorithms
from sklearn.neighbors import KNeighborsClassifier  # For K-Nearest Neighbors classif

# Import packages for fetching datasets
from sklearn.datasets import fetch_openml  # To fetch datasets from OpenML

# Import packages for feature transformations
from feature_engine.transformation import LogTransformer  # For logarithmic transform
from feature_engine.wrappers import SklearnTransformerWrapper  # To use scikit-learn
from sklearn.model_selection import KFold
from sklearn.model_selection import ShuffleSplit
```

Specify Project Folder Location

```
base_folder = Path('/content/drive/MyDrive/Applied_ML/Class_3/Assignment') # CHANGE T
# You need the else block only if you are NOT using COLAB
```

```
# CHANGE TO LOCATION BASED ON YOUR GOOGLE DRIVE
save_model_folder = base_folder/'Model' # CHANGE TO LOCATION WHERE YOU ARE PLANNING T
custom_function_folder = base_folder/'Custom_functions' # CHANGE TO LOCATION WHERE YO
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_3/Assignment/Custom_functions']
```

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

Task: Classification on the 'credit-g' dataset (10 points)

The goal is to classify people described by a set of attributes as good or bad credit risks.

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)

```
from sklearn.datasets import fetch_openml
```

```
# 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:1002: Future
  warn(
```

```
X.head()
```

	checking_status	duration	credit_history	purpose	credit_amount	savi
0	<0	6.0	critical/other existing credit	radio/tv	1169.0	no k

Task1 : EDA (9 Points)

Perfrom the initial EDA analysis and draw your conclusion. Based on EDA identify the preprocessing steps that we need to do. You can use the code from preprocessing notebook in Lecture 4.

Check Data (1.5 Points)

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

Check few rows

```
X.head(10)
```

	checking_status	duration	credit_history	purpose	credit_amount	savi
1	0<=X<200	48.0	existing paid	radio/tv	5951.0	
6	no checking	24.0	existing paid	furniture/equipment	2835.0	

Check column names

```
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

```
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 unique 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
```

Check summary statistics using describe function

```
X.describe().T
```

	count	mean	std	min	25%	50%	75%	m
duration	1000.0	20.903	12.058814	4.0	12.0	18.0	24.00	7
credit_amount	1000.0	3271.258	2822.736876	250.0	1365.5	2319.5	3972.25	1842
installment_commitment	1000.0	2.973	1.118715	1.0	2.0	3.0	4.00	
residence_since	1000.0	2.845	1.103718	1.0	2.0	3.0	4.00	
age	1000.0	35.546	11.375469	19.0	27.0	33.0	42.00	7
existing_credits	1000.0	1.407	0.577654	1.0	1.0	1.0	2.00	
num_dependents	1000.0	1.155	0.362086	1.0	1.0	1.0	1.00	

Check for duplicate rows

```
X.duplicated().any()
```

```
False
```

Quantifying Missing Data (1.5 Points)

```
X.isna().any()
```

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

Identify numerical, categorical and discrete variables (1.5 Points)

```
categorical = [z for z in X.columns if X[z].dtype == 'category']
discrete = [var for var in X.columns if X[var].dtype != 'category'
            and len(X[var].unique()) < 20]
continous = [ var for var in X.columns if X[var].dtype != 'category'
             and var not in discrete]

#for v in categorical,discrete:
  #df_merged[v] = df_merged[v].astype('object')

print(categorical)
print(discrete)
print(continous)
```

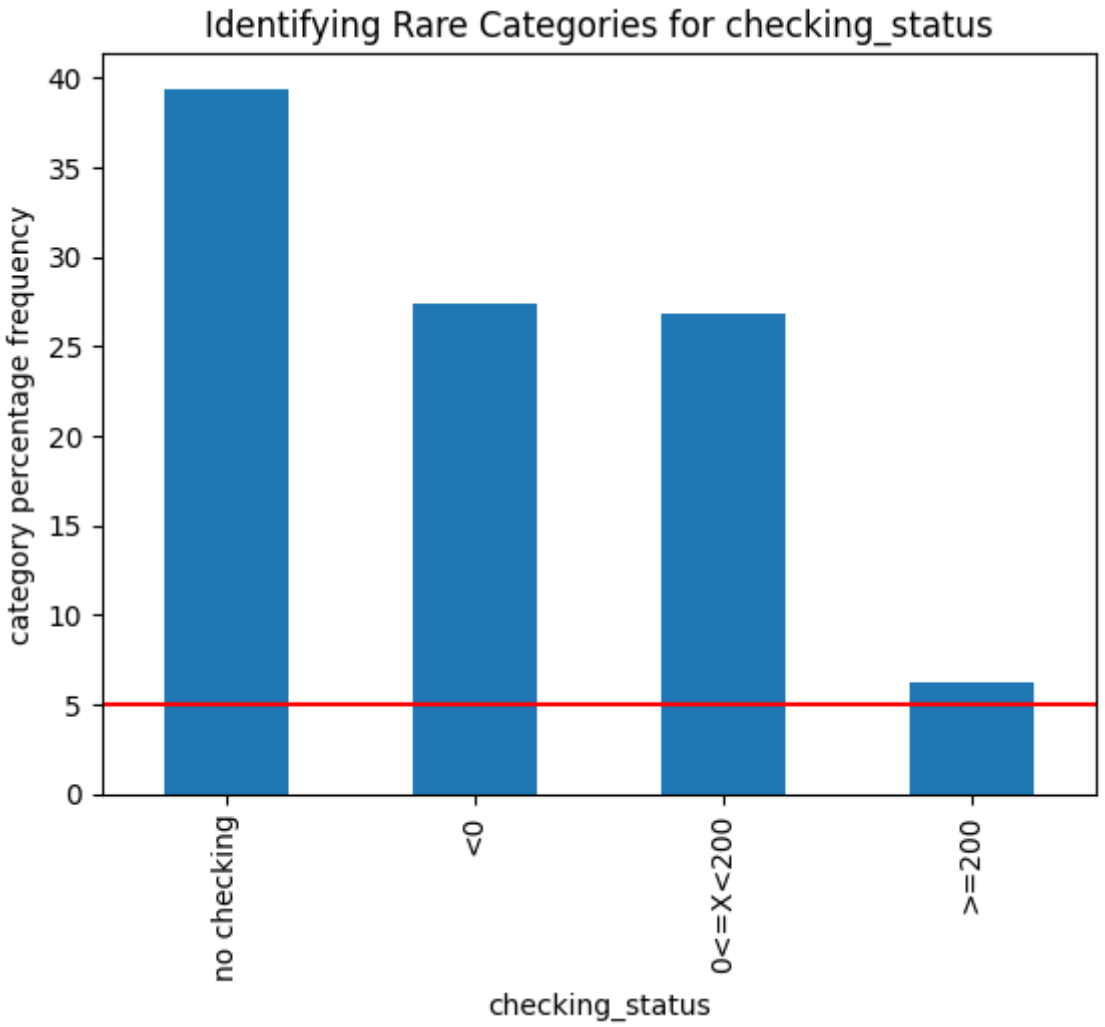
```
['checking_status', 'credit_history', 'purpose', 'savings_status', 'employment',
['installment_commitment', 'residence_since', 'existing_credits', 'num_dependent
['duration', 'credit_amount', 'age']
```

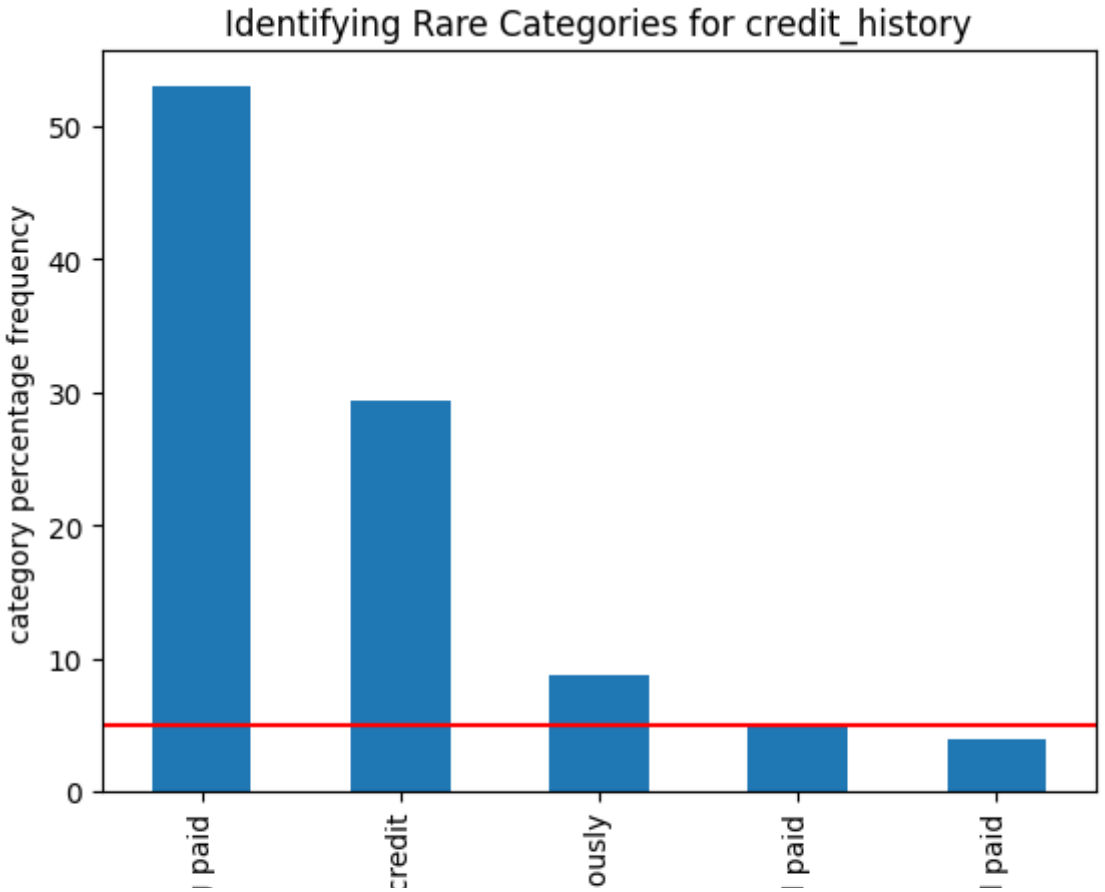
Check Variable Distributions (2 Points)

Categorical Varibles

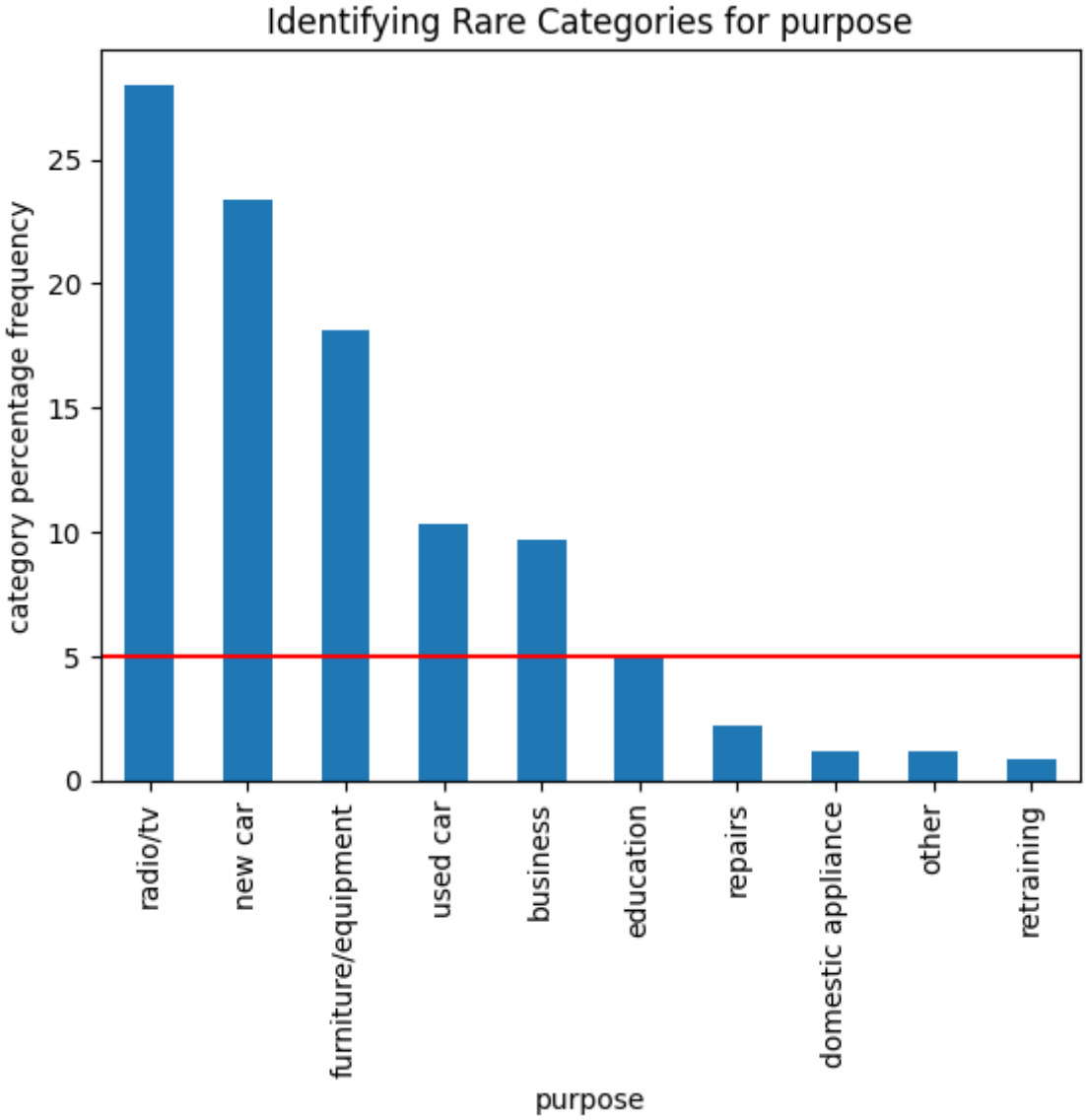
Frequency distribution of categorical variables and rare categories

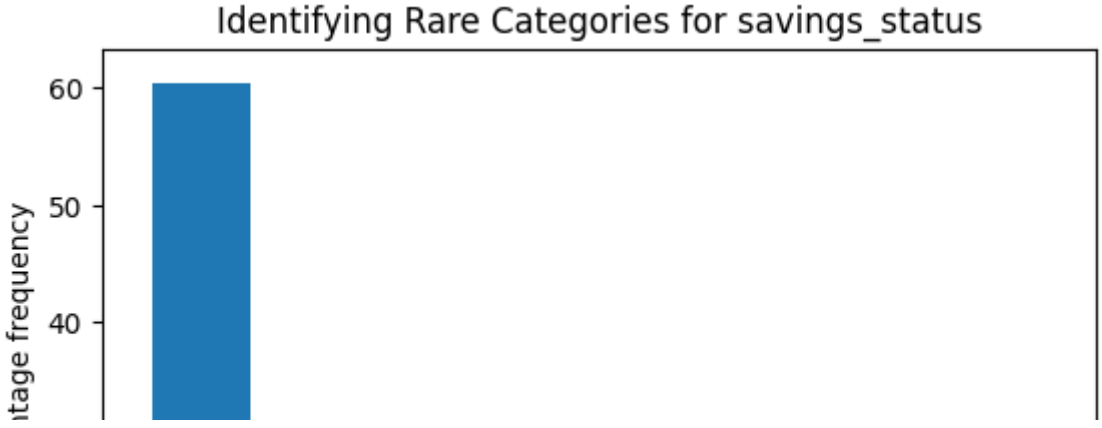
```
#creating a fucntion to check rare

def rare(df,var):
  cat_freq = 100 * df[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(f'Identifying Rare Categories for {var}')
  plt.show()

for var in categorical:
  rare(X,var)
```

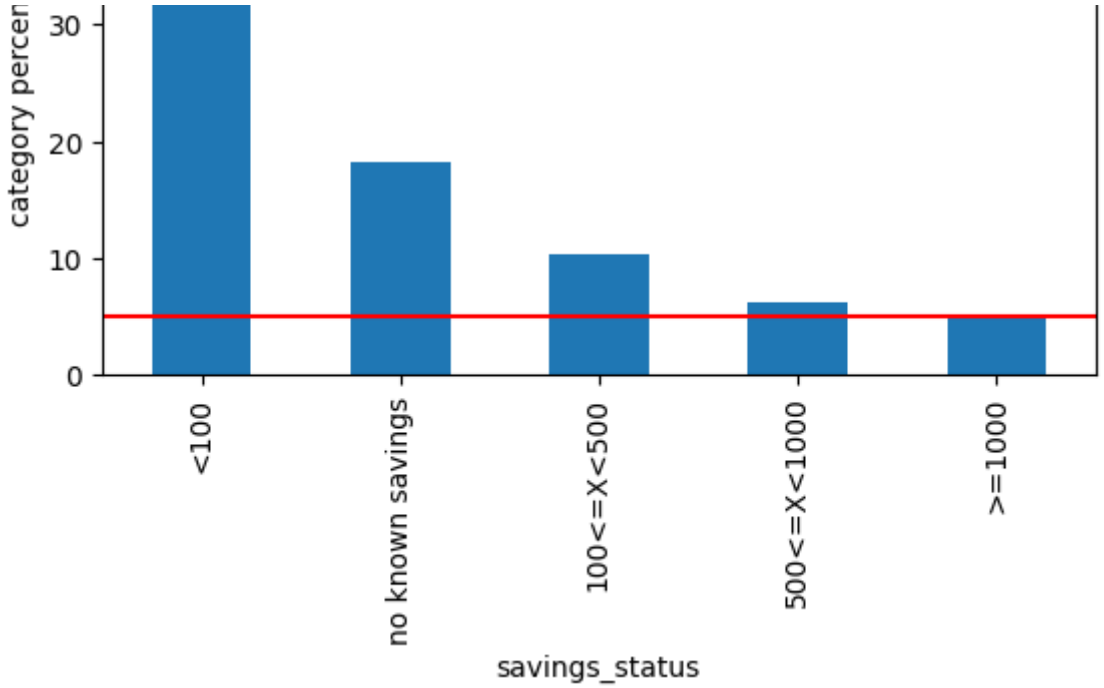

Identifying Rare Categories for checking_status
category percentage frequency
0
5
10
15
20
25
30
35
40
no checking
<0
0<=X<200
>=200
checking_status

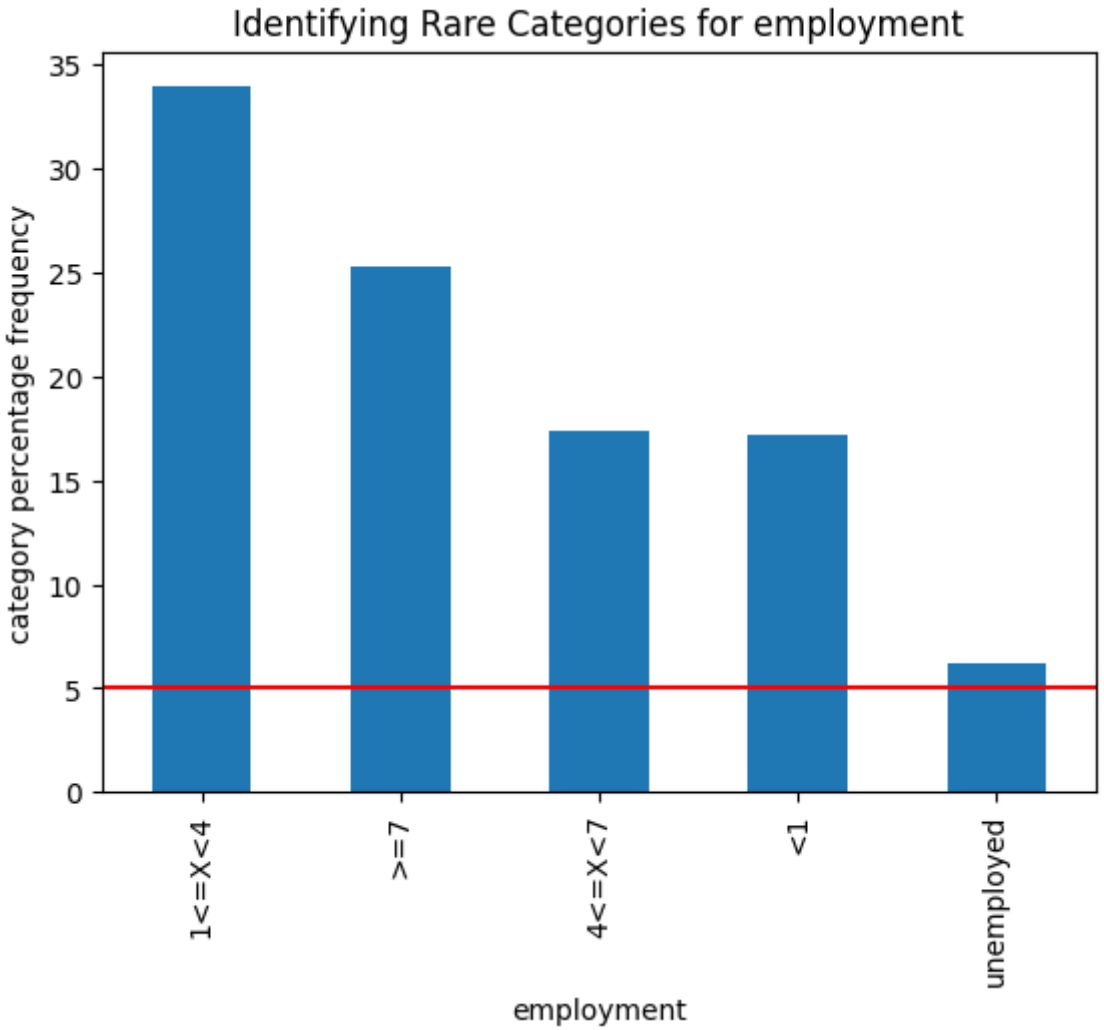

Identifying Rare Categories for credit_history
category percentage frequency
0
10
20
30
40
50
paid
credit
ously
paid
paid

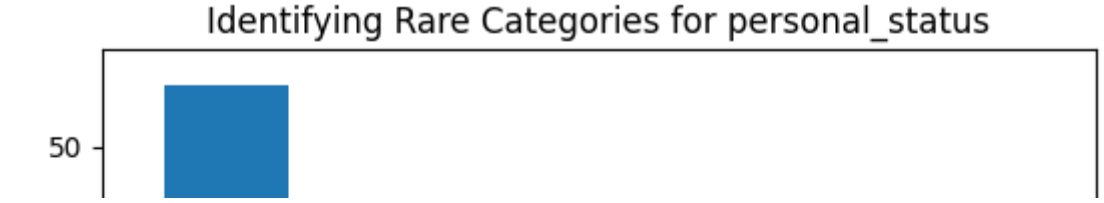

existing
critical/other existing
delayed previ
all
no credits/all
credit_history

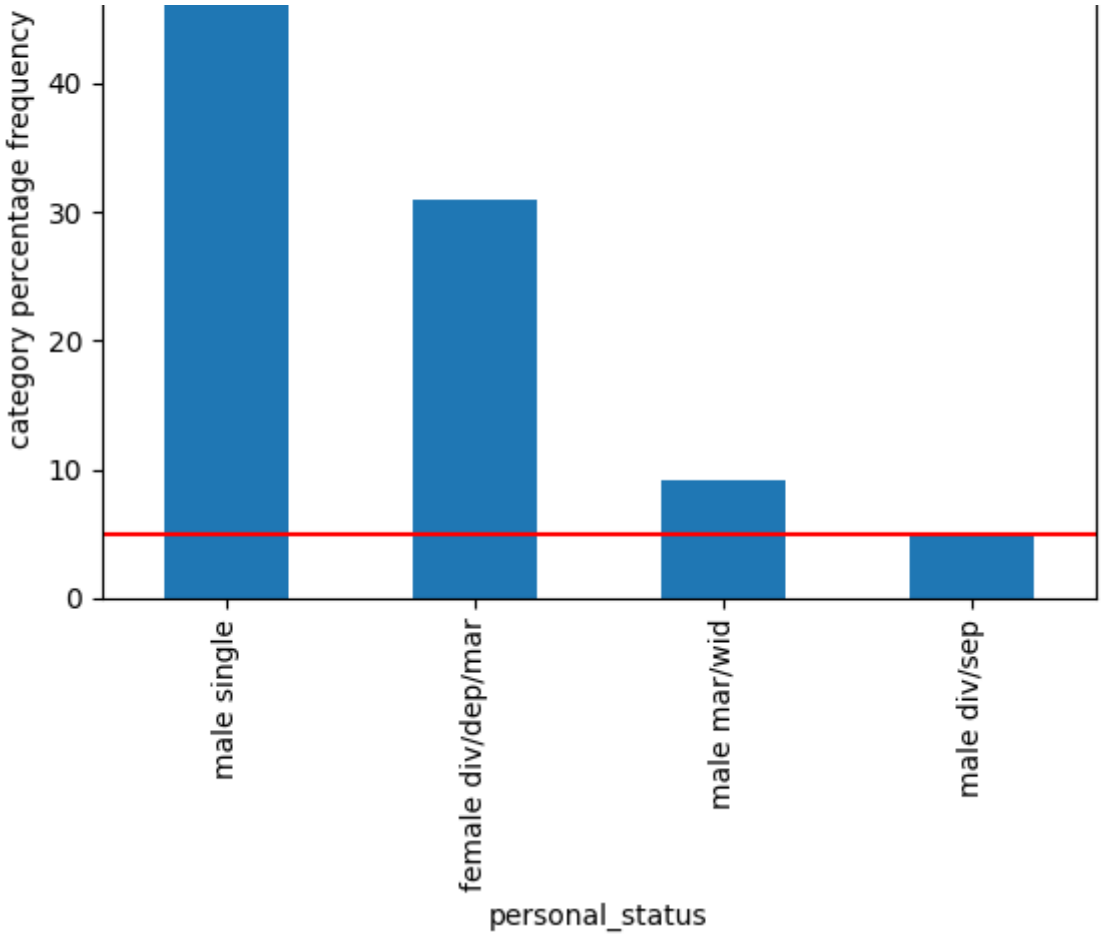

Identifying Rare Categories for purpose
category percentage frequency
0
5
10
15
20
25
radio/tv
new car
furniture/equipment
used car
business
education
repairs
domestic appliance
other
retraining
purpose

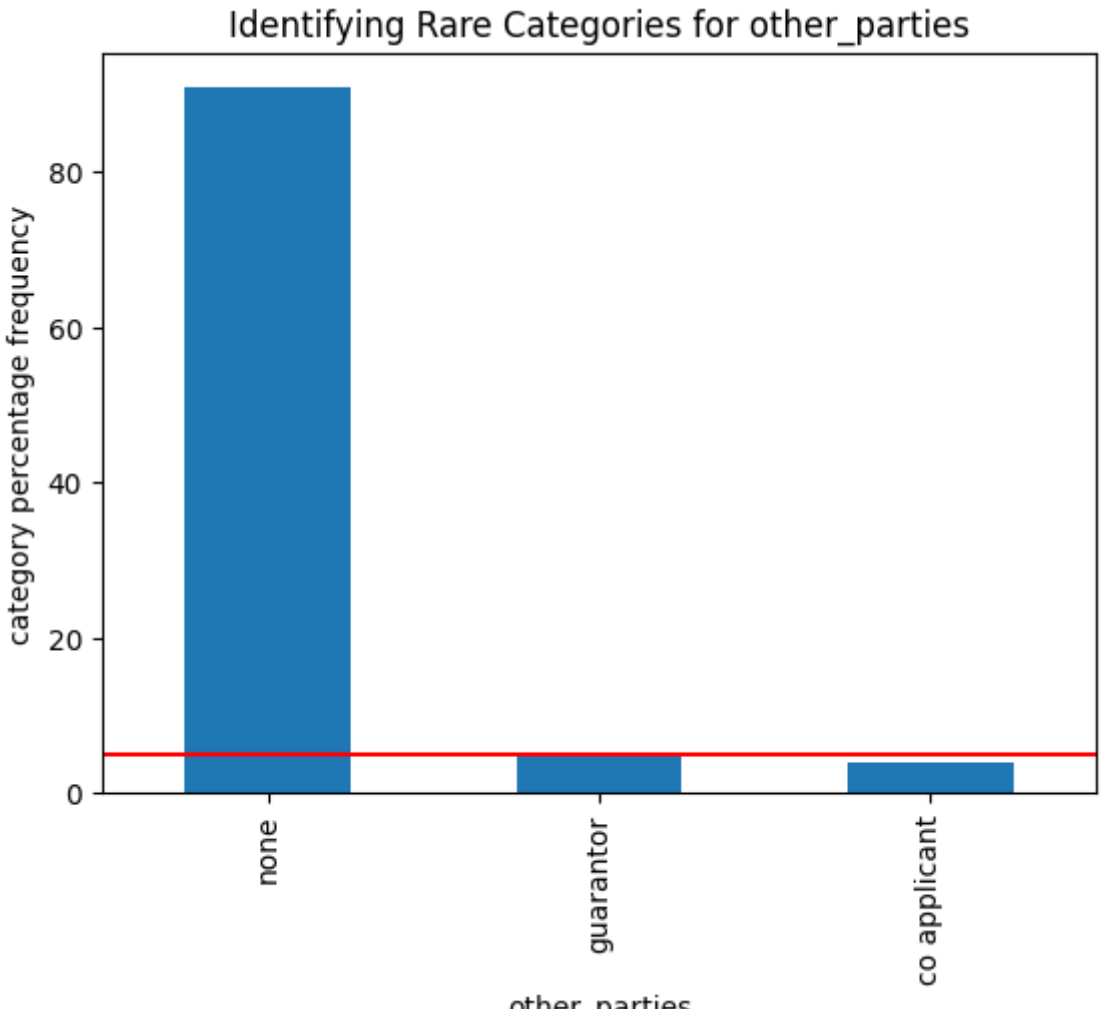

Identifying Rare Categories for savings_status
40
50
60

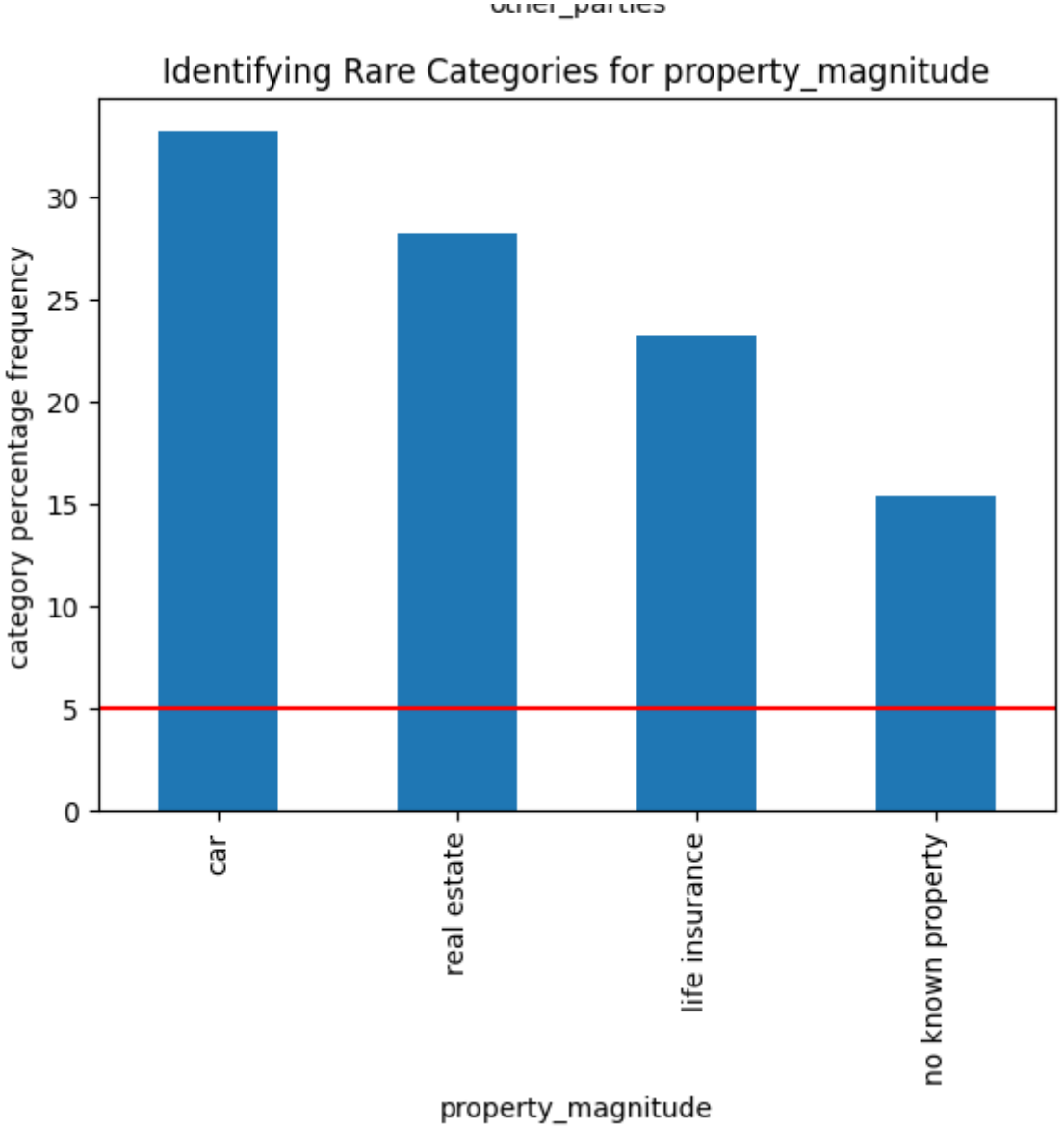

category percen
30
20
10
0
<100
no known savings
100<=X<500
500<=X<1000
>=1000
savings_status

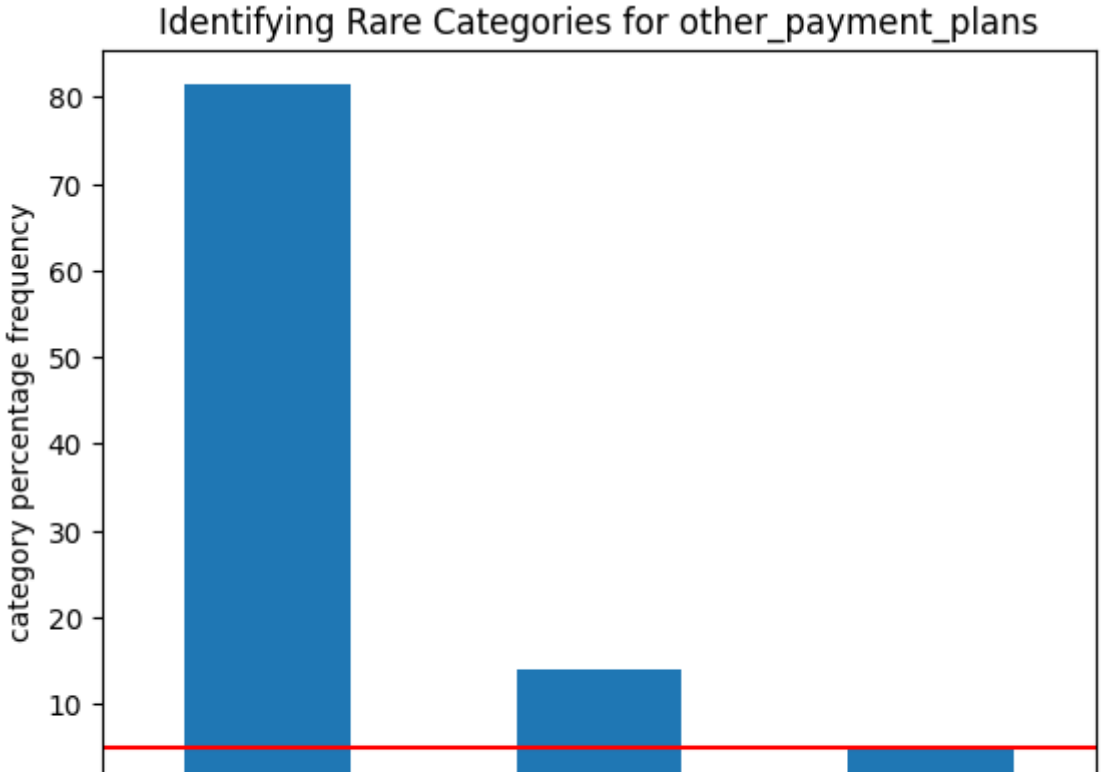

Identifying Rare Categories for employment
category percentage frequency
35
30
25
20
15
10
5
0
1<=X<4
>=7
4<=X<7
<1
unemployed
employment

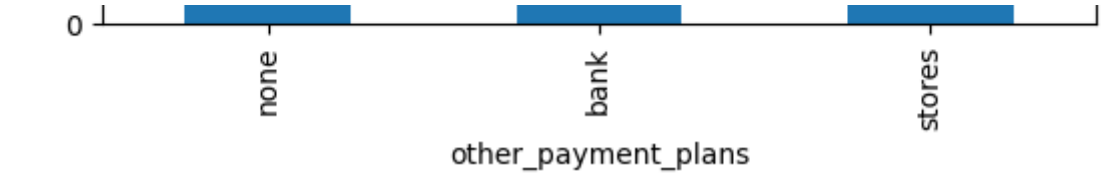

Identifying Rare Categories for personal_status
50

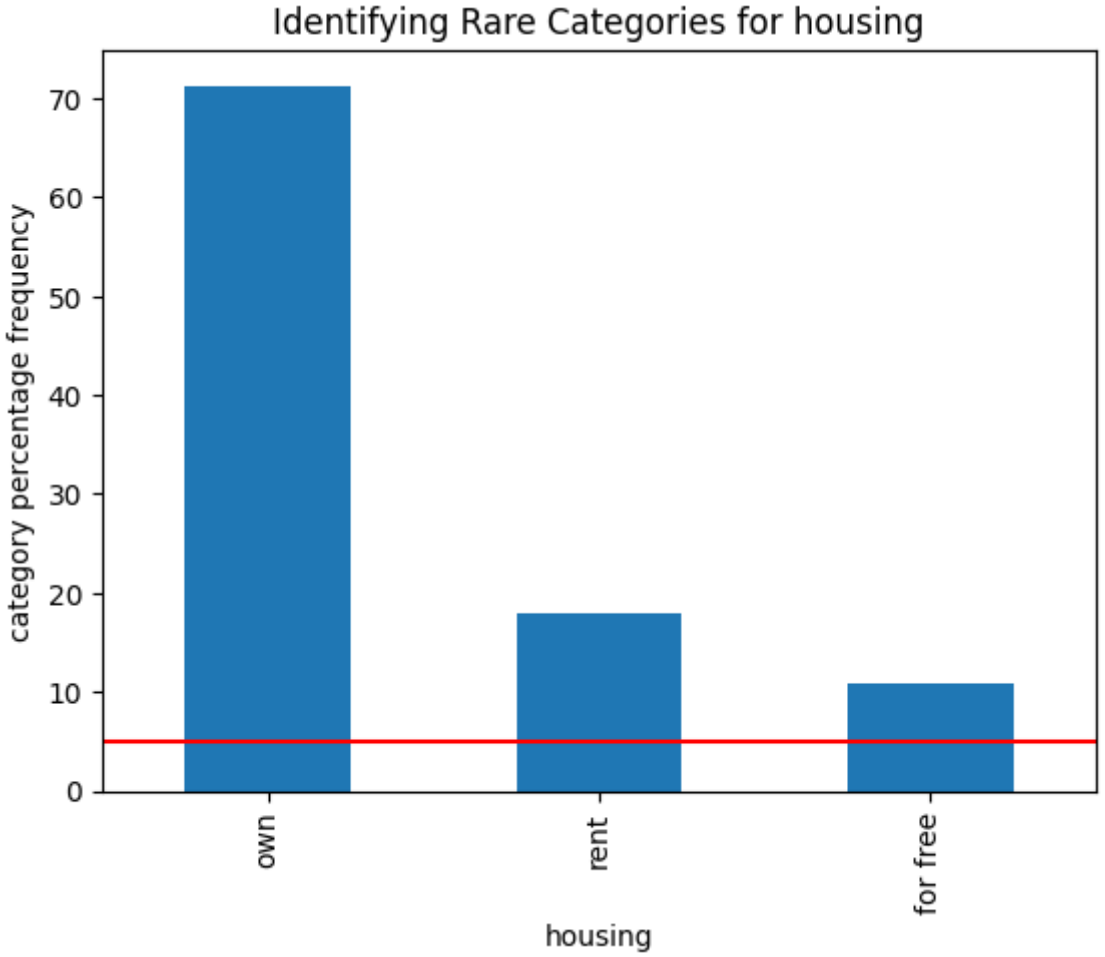

category percentage frequency
0
10
20
30
40
male single
female div/dep/mar
male mar/wid
male div/sep
personal_status

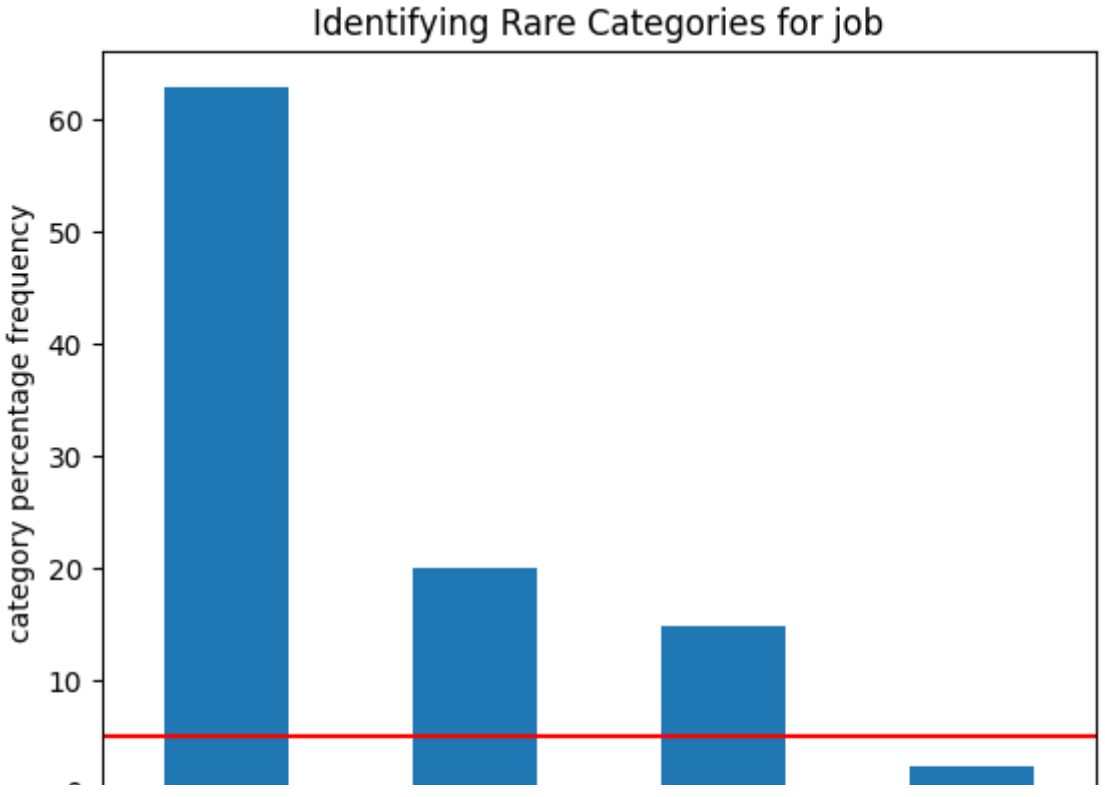

Identifying Rare Categories for other_parties
category percentage frequency
0
20
40
60
80
none
guarantor
co applicant
other_parties

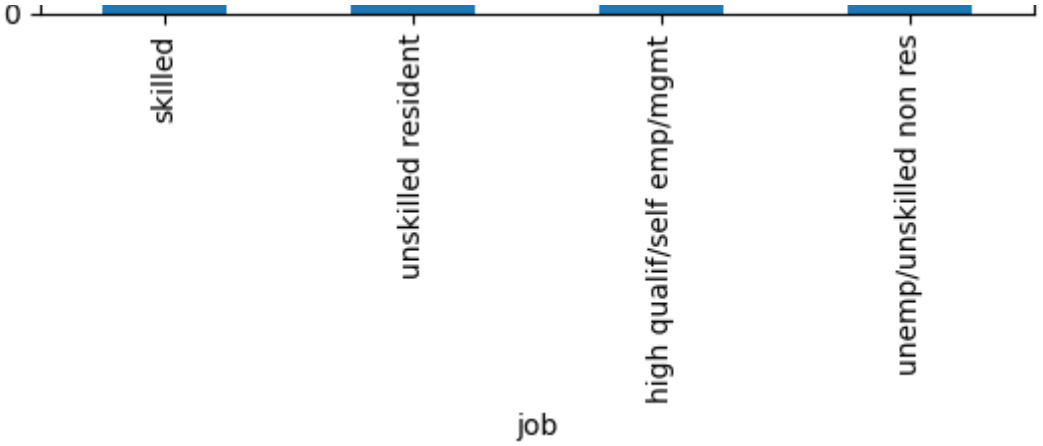

Identifying Rare Categories for property_magnitude
category percentage frequency
0
5
10
15
20
25
30
car
real estate
life insurance
no known property
property_magnitude

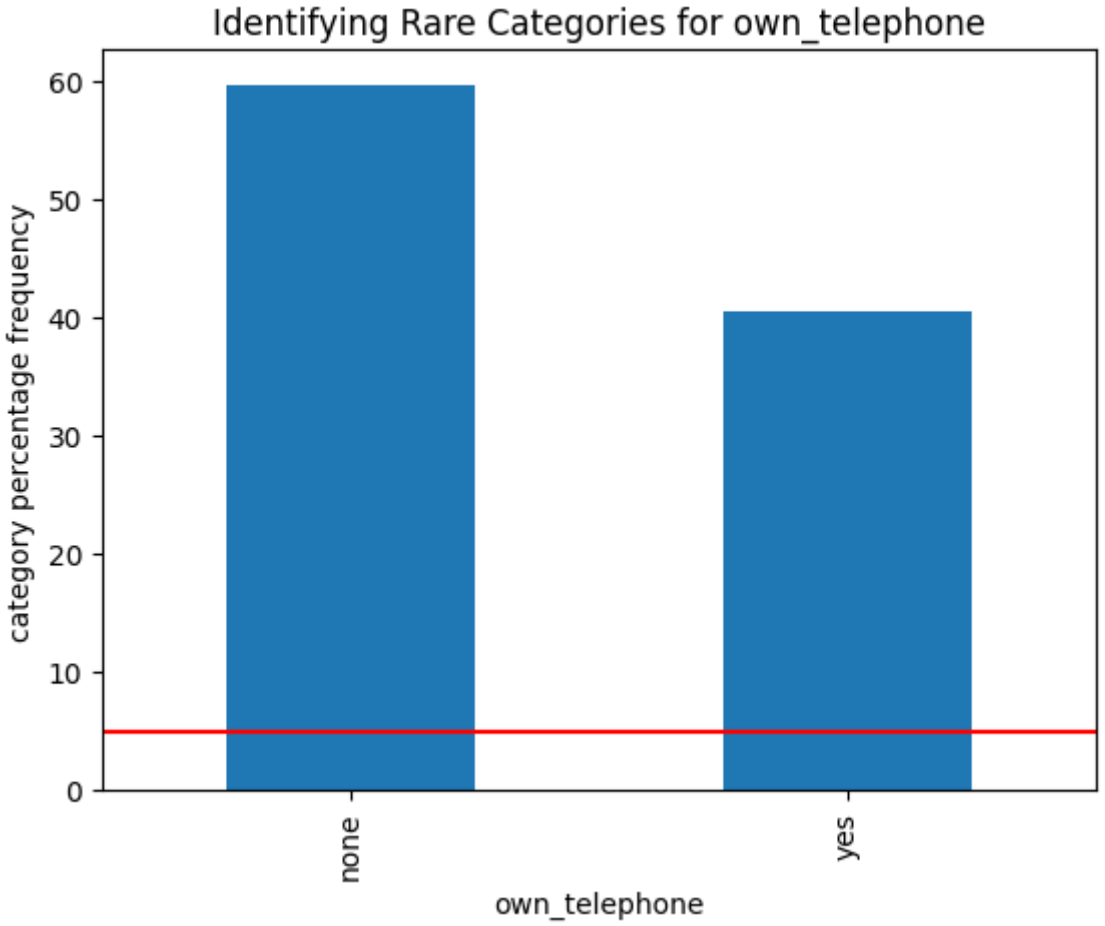

Identifying Rare Categories for other_payment_plans
category percentage frequency
10
20
30
40
50
60
70
80

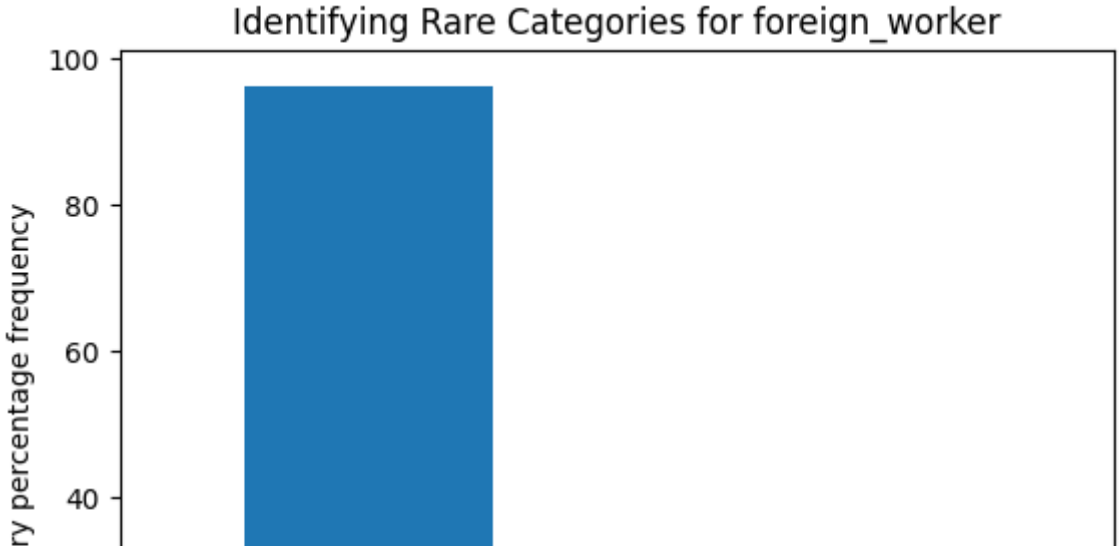

0
none
bank
stores
other_payment_plans


Identifying Rare Categories for housing
category percentage frequency
0
10
20
30
40
50
60
70
own
rent
for free
housing


Identifying Rare Categories for job
category percentage frequency
10
20
30
40
50
60


0
skilled
unskilled resident
high qualif/self emp/mgmt
unemp/unskilled non res
job


Identifying Rare Categories for own_telephone
category percentage frequency
60
50
40
30
20
10
0
none
yes
own_telephone

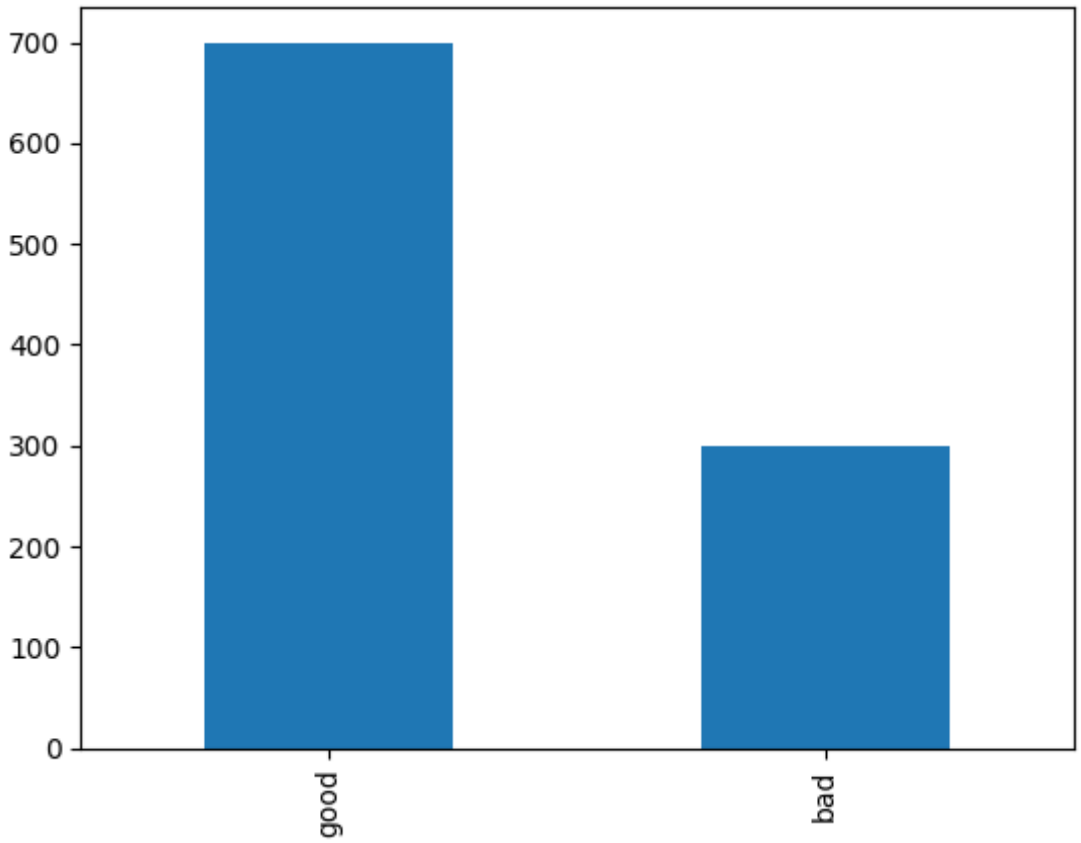

Identifying Rare Categories for foreign_worker
100
80
60
40

Check distribution of target variable

foreign_worker

```
#print(y.value_counts())
#dv = pd.DataFrame(y)
#dv.head()
#dv.rename(columns={'class':'output'})
y.value_counts().plot.bar()
#y.info()
#df_merged = pd.concat([X, y], axis=1)
#df_merged.head()
```

```
<Axes: >
```



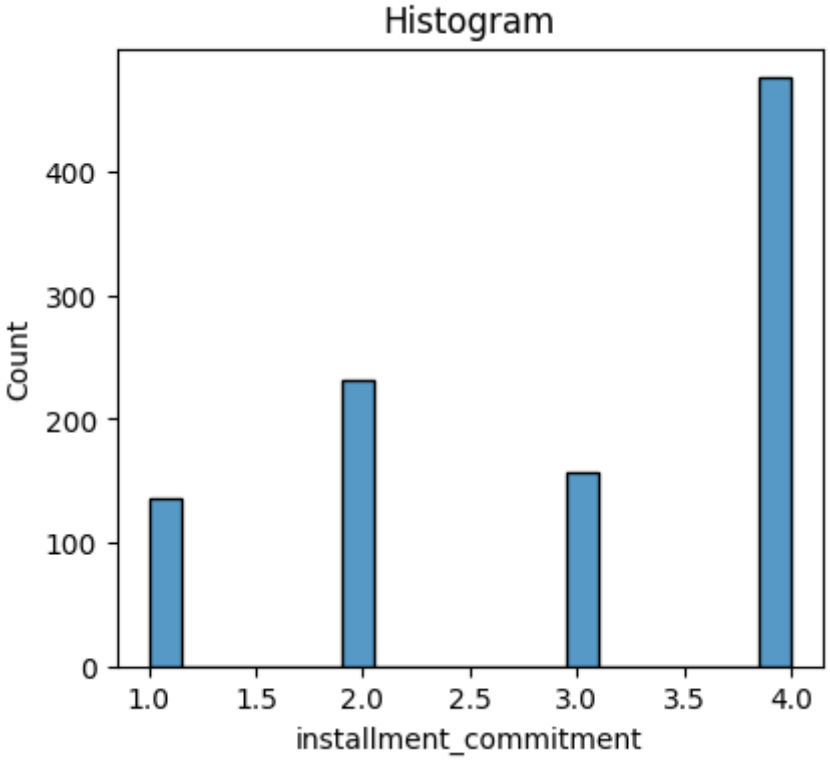
Distribution of continuous and discrete variables

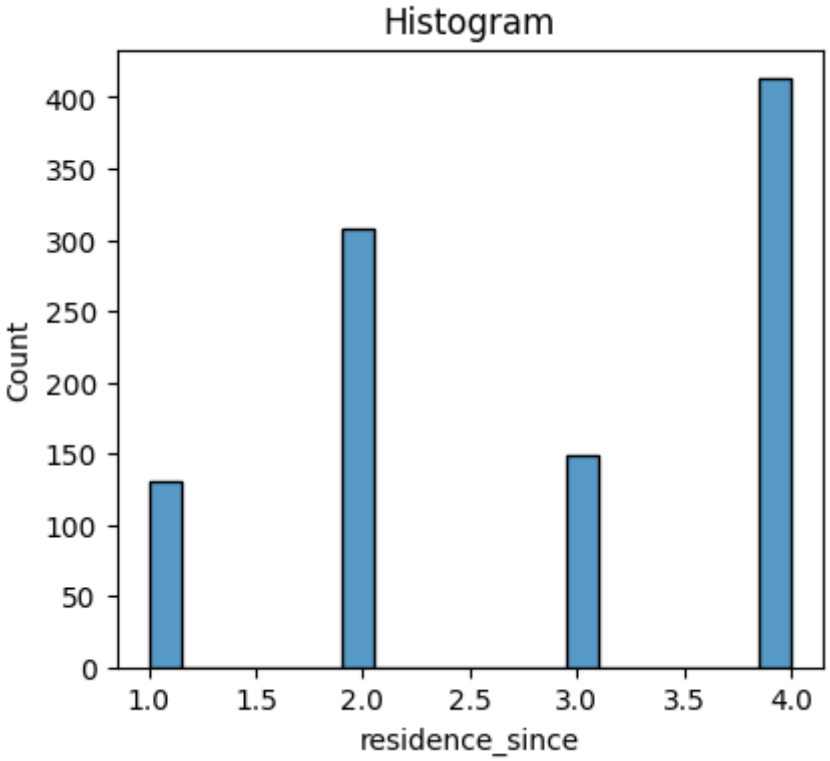
Distribution of continuous variables

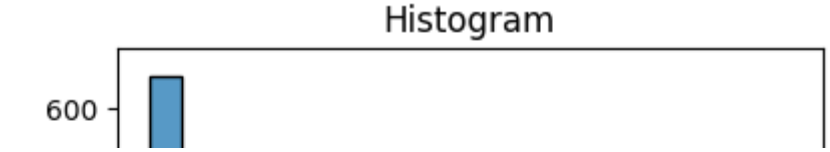
```
for var in continous:
  diagnostic_plots(X, var)
```

Distribution of discrete variables

```
for var in discrete:
  plt.figure(figsize=(10, 4))
  plt.subplot(1, 2, 1)
  sns.histplot(X[var], bins=20)
  plt.title('Histogram')
```







Visualizing Relationships between variables (1 Point)

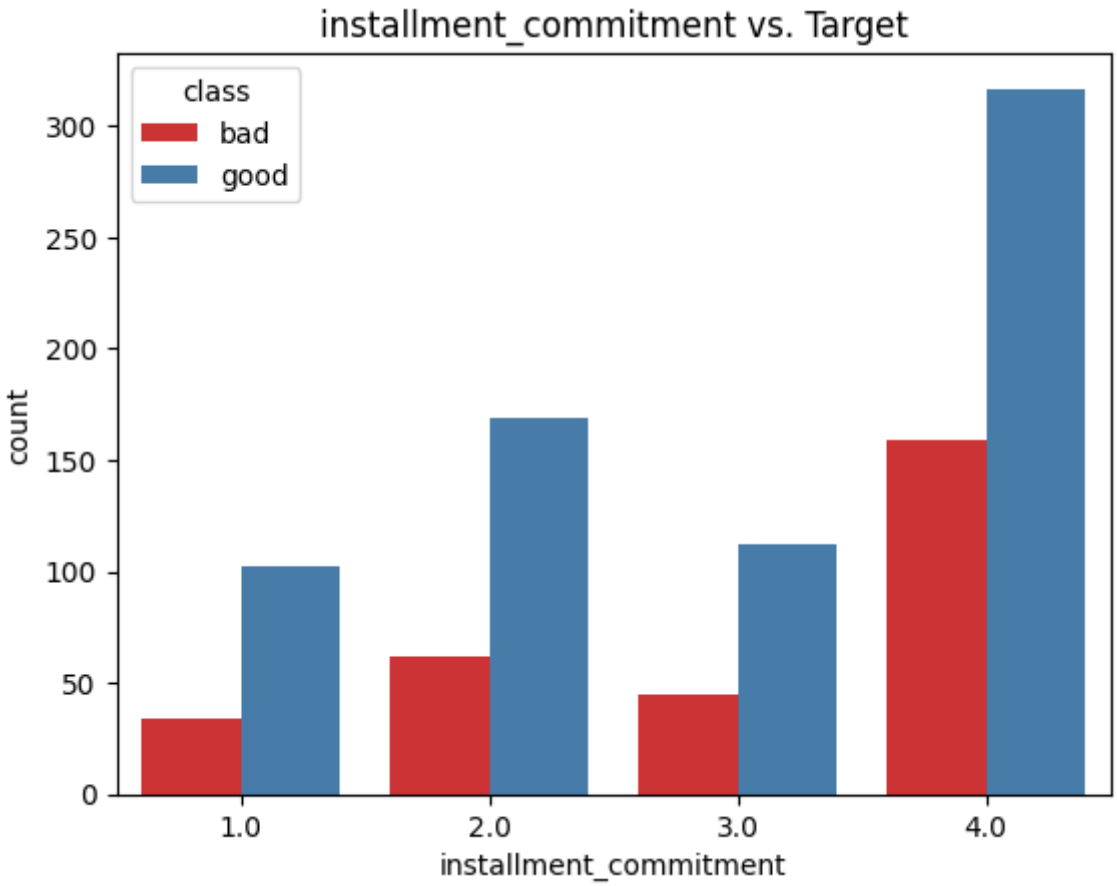
Correlation Matrix

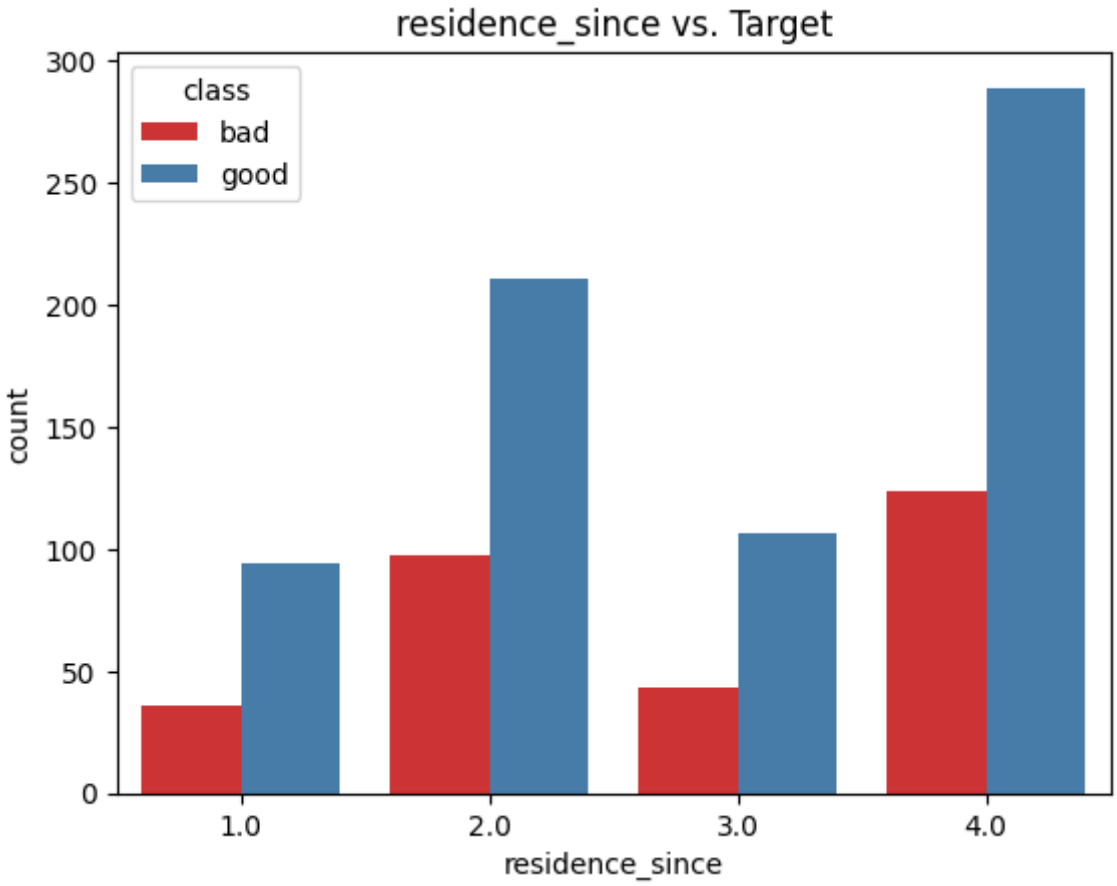
```
corrmat = X[discrete + continous].corr().round(2)
top_corr_features = corrmat.index
plt.figure(figsize=(7, 7))
sns.heatmap(X[top_corr_features].corr(),annot=True, square=True, fmt='.2f',
            cbar_kws={"shrink": .80}, linewidths=.5, cmap='RdYlGn');
```

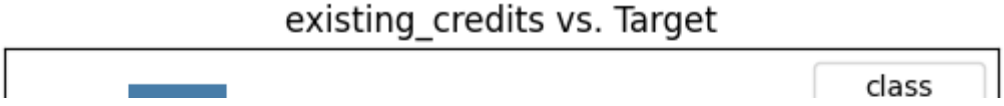
Relationship between Target variable and categorical variables

```
#for category in categorical + discrete:
  #plot_target_by_category(df_merged, 'class' ,category,'Credit-g')
```

```
for var in categorical and discrete:
    sns.countplot(x=var, hue=y, data=X, palette="Set1")
    plt.title(f"{var} vs. Target")
    plt.show()
```


installment_commitment vs. Target
class
bad
good
count
0
50
100
150
200
250
300
1.0
2.0
3.0
4.0
installment_commitment


residence_since vs. Target
class
bad
good
count
0
50
100
150
200
250
300
1.0
2.0
3.0
4.0
residence_since


existing_credits vs. Target
class

Conclusion from EDA (1.5 Points)

Conclusions:

- We have no Null values in the Dataset.
- All the continous variables have outliers in the and all are left skewed as the median is less than mean as seen in stats
- We have a few categories which distribution have rare values. One of the ways to avoid rare is to group all the rare categories to a single column
- Every category w.r.t. good/bad credit score is following a linear pattern without any anomolies
- Credit amount and duration are highly positively correlated. If question of computing, on ef the column is redundant
- Insatllement and credit amount are negatively moderately corelated.
- Discrete variables do not have normal distrubution but no anomolies as well
- We are doing one hot encoding to the categories to create dummy variables.

Task2: Preprocessing (2 Points)

Split data into training and test set. In this HW, we will only do one type of preprocessing - (1) Encode categorical variables (your choice of encoder). You can use sklearn or feature-engine. Encode categorical variables in both Train and Test 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,random_state=0)
```

```
# Importing One Hot encoder from feature engine, use drop last = True
from feature_engine.encoding import OneHotEncoder
ohe = OneHotEncoder(variables=categorical,
      drop_last= True)
```

```
# fit_transform on train
X_train_fe = ohe.fit_transform(X_train)
```

```
# only transform on test
X_test_fe = ohe.transform(X_test)
```

```
X_train_fe.head()
```

	duration	credit_amount	installment_commitment	residence_since	age	exist
253	24.0	4151.0	2.0	3.0	35.0	
667	48.0	3609.0	1.0	1.0	27.0	
85	12.0	1412.0	4.0	2.0	29.0	
969	11.0	3939.0	1.0	2.0	40.0	
75	12.0	1526.0	4.0	4.0	66.0	

5 rows × 48 columns

```
X_test_fe.head()
```

	duration	credit_amount	installment_commitment	residence_since	age	exist
993	36.0	3959.0	4.0	3.0	30.0	
859	9.0	3577.0	1.0	2.0	26.0	
298	18.0	2515.0	3.0	4.0	43.0	
553	12.0	1995.0	4.0	1.0	27.0	
672	60.0	10366.0	2.0	4.0	42.0	

5 rows × 48 columns

Task3: Train KNN calsssifier using Train Data (2 points)

You will use Gridserach to find the best value of number of neighbors. Remember we have to find the best value using cross validation error. Then use the score method to find accuracy on both test and train datasets.

Round 1

```
#Instructions
# 1. Convert Pandas DataFrame to NumPy arrays. The variables are named X_train_fe, y_
# Replace 'values' with the method to convert a DataFrame to a NumPy array.
X_train_fe_numpy = X_train_fe.values
y_train_numpy  = y_train.values
X_test_fe_numpy  = X_test_fe.values
y_test_numpy  = y_test.values
```

```
# 2. Create a dictionary for hyperparameter tuning. The hyperparameter to tune is 'n_
# The possible values are from 1 to 30 with a step of 4.
# Replace '# CODE HERE' with the dictionary.
param_grid = {'n_neighbors': np.arange(1,31,4)}

# 3. Initialize GridSearchCV with the kNN classifier and 5-fold cross-validation.
# Use 'return_train_score=True' to get the training score.
# Replace '# CODE HERE' with the initialization code.
grid = GridSearchCV(KNeighborsClassifier(),param_grid=param_grid,cv=5,return_train_sc


# 4. Fit the GridSearchCV object to the training data.
# Replace '# CODE HERE' with the code to fit GridSearchCV to the training data.
grid.fit(X_train_fe_numpy,y_train_numpy)

# 5. Obtain the best mean cross-validation score.
# Replace '#CODE HERE' with the attribute that holds this value from the GridSearchCV
print(f"Best mean cross-validation score: {grid.best_score_}")

# 6. Obtain the optimal 'n_neighbors' parameter value.
# Replace 'CODE HERE' with the attribute that holds this value from the GridSearchCV
print(f"Optimal 'n_neighbors' value: {grid.best_params_}")

# 7. Evaluate and print the training set accuracy.
# Replace '# CODE HERE' with the code to get the training set accuracy using the best
print(f"Training set accuracy: {grid.score(X_train_fe_numpy, y_train_numpy):.4f}")

# 8. Evaluate and print the test set accuracy.
# Replace '# CODE HERE' with the code to get the test set accuracy using the best hyp
print(f"Test set accuracy: {grid.score(X_test_fe_numpy,y_test_numpy):.4f}")
```

```
Best mean cross-validation score: 0.7
Optimal 'n_neighbors' value: {'n_neighbors': 17}
Training set accuracy: 0.7200
Test set accuracy: 0.6880
```

Round 2:

```
# Implement k-Nearest Neighbors (kNN) classification with hyperparameter tuning via G
# Define the hyperparameter grid for 'n_neighbors' based on your initial findings.
# If the best 'n_neighbors' value from the initial run is X, consider a range around
# For example, if your initial best value was 25 with a step size of 4, you might con
param_grid_1 = {'n_neighbors': np.arange(14,21,1)}


# Initialize GridSearchCV with kNN classifier, specifying 5-fold cross-validation.
# The option 'return_train_score=True' enables inspection of training scores.
```

```
grid_1 = GridSearchCV(KNeighborsClassifier(), param_grid = param_grid_1,cv = 5, retur

# Fit the GridSearchCV model to the training data.
# Replace '# CODE HERE' with the appropriate code to fit the GridSearchCV object.
grid_1.fit(X_train_fe_numpy,y_train_numpy)

# Retrieve and display optimal performance metrics.
# 1. Replace '#CODE HERE' with code to obtain the best mean cross-validation score fr
# 2. Replace '# CODE HERE' with code to get the optimal 'n_neighbors' parameter value
print(f"Best mean cross-validation score: {grid_1.best_score_}")
print(f"Optimal 'n_neighbors' value: {grid_1.best_params_}")

# Evaluate the model on both training and test sets using the best hyperparameters.
# Replace '# CODE HERE' with code to obtain accuracy for the training and test sets.
print(f"Training set accuracy: {grid_1.score(X_train_fe_numpy,y_train_numpy)}")
print(f"Test set accuracy: {grid_1.score(X_test_fe_numpy,y_test_numpy)}")
```

```
Best mean cross-validation score: 0.7
Optimal 'n_neighbors' value: {'n_neighbors': 17}
Training set accuracy: 0.72
Test set accuracy: 0.688
```

Task4: Change the cross-validation strategy (2 Points)

Round 1

```
folds_1 = ShuffleSplit(n_splits=5,test_size=0.4,random_state=0)
grid_2 = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid_1 , cv = folds_1,
grid_2.fit(X_train_fe_numpy,y_train_numpy)
print(f'best parameter is {grid_2.best_score_}')
print(f"Optimal 'n_neighbors' value: {grid_2.best_params_}")
print(f"Training set accuracy: {grid_2.score(X_train_fe_numpy,y_train_numpy)}")
print(f"Test set accuracy: {grid_2.score(X_test_fe_numpy,y_test_numpy)}")
```

```
best parameter is 0.6853333333333333
Optimal 'n_neighbors' value: {'n_neighbors': 19}
Training set accuracy: 0.716
Test set accuracy: 0.7
```

Round 2:

```
folds_2 = KFold(n_splits=5, shuffle=True, random_state=0)
grid_3 = GridSearchCV(KNeighborsClassifier(), param_grid=param_grid_1 , cv = folds_2,
grid_3.fit(X_train_fe_numpy,y_train_numpy)
print(f'best parameter is {grid_3.best_score_}')
print(f"Optimal 'n_neighbors' value: {grid_3.best_params_}")
print(f"Training set accuracy: {grid_3.score(X_train_fe_numpy,y_train_numpy)}")
print(f"Test set accuracy: {grid_3.score(X_test_fe_numpy,y_test_numpy)}")
```

```
best parameter is 0.6880000000000001
Optimal 'n_neighbors' value: {'n_neighbors': 19}
Training set accuracy: 0.716
Test set accuracy: 0.7
```