Preprocessing

In [3]:
```
import pandas as pd
from path import Path
# Load data
file_path = Path("../Resources/loans_data_encoded.csv")
loans_df = pd.read_csv(file_path)
loans_df.head(15)
```

Out[3]:

	amount	term	age	bad	month_num	education_Bachelor	education_High School or Below	education_Master or Above	e
0	1000	30	45	0	6	0	1	0	
1	1000	30	50	0	7	1	0	0	
2	1000	30	33	0	8	1	0	0	
3	1000	15	27	0	9	0	0	0	
4	1000	30	28	0	10	0	0	0	
5	300	7	35	0	7	0	0	1	
6	1000	30	29	0	9	0	0	0	
7	1000	30	36	0	5	0	0	0	
8	1000	30	28	0	5	0	0	0	
9	800	15	26	0	4	0	0	0	
10	300	7	29	0	4	0	0	0	
11	1000	15	39	0	9	0	1	0	
12	1000	30	26	0	12	0	0	0	
13	900	7	26	0	5	0	0	0	
14	1000	7	27	0	5	0	1	0	

In [4]:
```
# Define features set
X = loans_df.copy()
X = X.drop("bad", axis=1)
X.head()
```

Out[4]:

	amount	term	age	month_num	education_Bachelor	education_High School or Below	education_Master or Above	educati
0	1000	30	45	6	0	1	0	
1	1000	30	50	7	1	0	0	
2	1000	30	33	8	1	0	0	
3	1000	15	27	9	0	0	0	
4	1000	30	28	10	0	0	0	

In [5]:
```
# Define target vector
y = loans_df["bad"].values
```

In [6]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
# Splitting into Train and Test sets
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    random_state=1)

# Creating StandardScaler instance
scaler = StandardScaler()

# Fitting Standard Scaler
X_scaler = scaler.fit(X_train)

# Scaling data
X_train_scaled = X_scaler.transform(X_train)
X_test_scaled = X_scaler.transform(X_test)
```

```
/Users/ruiminma/opt/anaconda3/envs/mlenv/lib/python3.7/site-packages/sklearn/preprocessing/data.py:625: DataConversionWarning: Data with input dtype int64 were all converted to float64 by StandardScaler.
  return self.partial_fit(X, y)
/Users/ruiminma/opt/anaconda3/envs/mlenv/lib/python3.7/site-packages/ipykernel_launcher.py:15: DataConversionWarning: Data with input dtype int64 were all converted to float64 by StandardScaler.
  from ipykernel import kernelapp as app
/Users/ruiminma/opt/anaconda3/envs/mlenv/lib/python3.7/site-packages/ipykernel_launcher.py:16: DataConversionWarning: Data with input dtype int64 were all converted to float64 by StandardScaler.
  app.launch_new_instance()
```

Choose best learning rate

In [7]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Create a classifier object
learning_rates = [0.05, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in learning_rates:
    classifier = GradientBoostingClassifier(n_estimators=20,
                                            learning_rate=learning_rate,
                                            max_features=5,
                                            max_depth=3,
                                            random_state=0)

    # Fit the model
    classifier.fit(X_train_scaled, y_train)
    print("Learning rate: ", learning_rate)

    # Score the model
    print("Accuracy score (training): {0:.3f}".format(
        classifier.score(
            X_train_scaled,
            y_train)))
    print("Accuracy score (validation): {0:.3f}".format(
        classifier.score(
            X_test_scaled,
            y_test)))
    print()
```

```
Learning rate:  0.05
Accuracy score (training): 0.627
Accuracy score (validation): 0.520

Learning rate:  0.1
Accuracy score (training): 0.667
Accuracy score (validation): 0.528

Learning rate:  0.25
Accuracy score (training): 0.723
Accuracy score (validation): 0.536

Learning rate:  0.5
Accuracy score (training): 0.755
Accuracy score (validation): 0.560

Learning rate:  0.75
Accuracy score (training): 0.781
Accuracy score (validation): 0.520

Learning rate:  1
Accuracy score (training): 0.792
Accuracy score (validation): 0.480
```

Create Gradient Boosting Classifier

In [9]:
```
# Choose a learning rate and create classifier
classifier = GradientBoostingClassifier(n_estimators=20,
                                        learning_rate=0.5,
                                        max_features=5,
                                        max_depth=3,
                                        random_state=0)

# Fit the model
classifier.fit(X_train_scaled, y_train)

# Make Prediction
predictions = classifier.predict(X_test_scaled)
pd.DataFrame({"Prediction": predictions, "Actual": y_test}).head(20)
```

Out[9]:

	Prediction	Actual
0	0	1
1	0	1
2	0	0
3	1	0
4	0	1
5	1	1
6	0	1
7	0	0
8	0	0
9	0	0
10	1	1
11	0	0
12	0	0
13	0	0
14	0	1
15	0	0
16	0	1
17	0	0
18	0	0
19	0	0

Evaluate the model

In [11]:
```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
# Calculating the accuracy score
acc_score = accuracy_score(y_test, predictions)
print(f"Accuracy Score : {acc_score}")
```

```
Accuracy Score : 0.56
```

In [12]:
```
# Generate the confusion matrix
cm = confusion_matrix(y_test, predictions)
cm_df = pd.DataFrame(
    cm, index=["Actual 0", "Actual 1"],
    columns=["Predicted 0", "Predicted 1"]
)

# Displaying results
display(cm_df)
```

	Predicted 0	Predicted 1
Actual 0	49	16
Actual 1	39	21

In [13]:
```
# Generate classification report
print("Classification Report")
print(classification_report(y_test, predictions))
```

```
Classification Report
              precision    recall  f1-score   support

           0       0.56      0.75      0.64        65
           1       0.57      0.35      0.43        60

   micro avg       0.56      0.56      0.56       125
   macro avg       0.56      0.55      0.54       125
weighted avg       0.56      0.56      0.54       125
```

In [ ]:
```

```