CRITICAL ANALYSIS REPORT

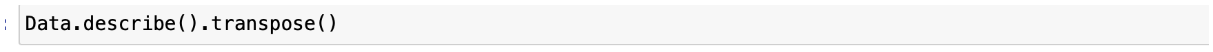
Albina Maharjan

Student ID: 48045977

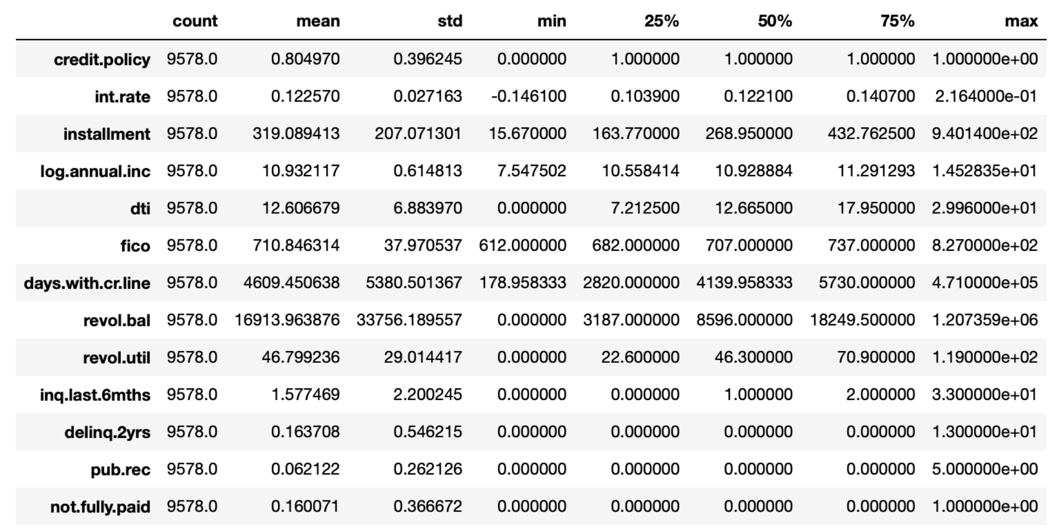
Table of Contents

1. Load Dataset and Clean Dataset

Given Code:

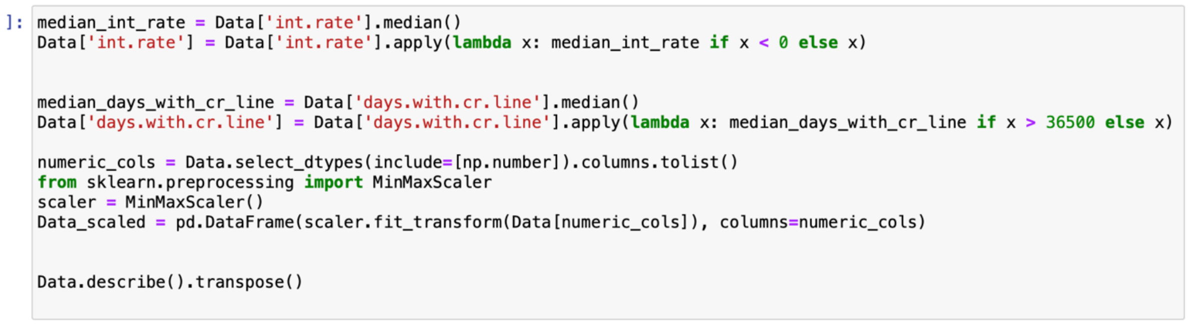


```
Data.describe().transpose()
```

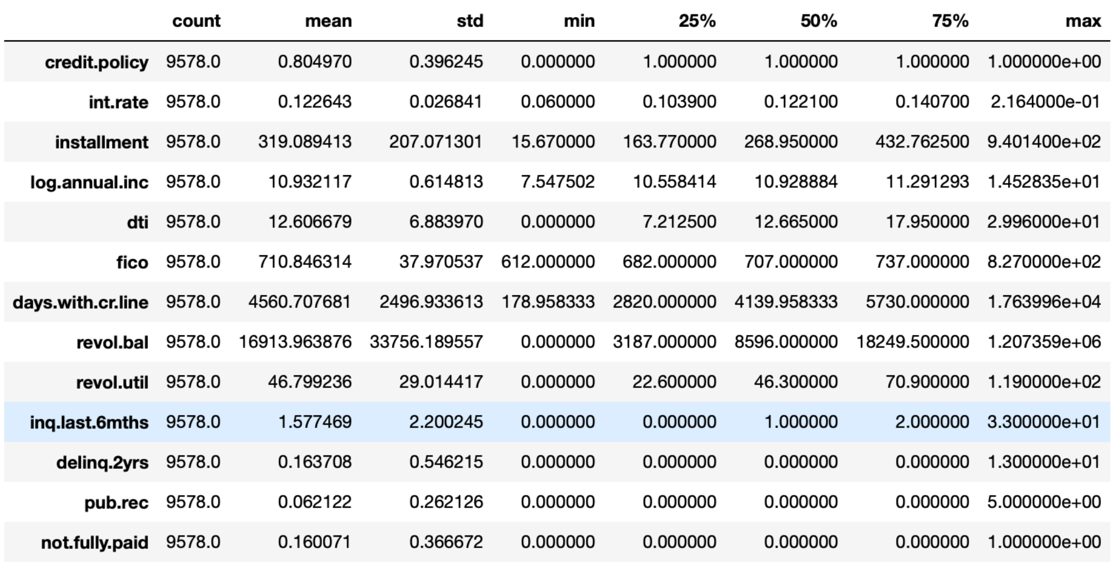


	count	mean	std	min	25%	50%	75%	max
credit.policy	9578.0	0.804970	0.396245	0.000000	1.000000	1.000000	1.000000	1.000000e+00
int.rate	9578.0	0.122570	0.027163	-0.146100	0.103900	0.122100	0.140700	2.164000e-01
installment	9578.0	319.089413	207.071301	15.670000	163.770000	268.950000	432.762500	9.401400e+02
log.annual.inc	9578.0	10.932117	0.614813	7.547502	10.558414	10.928884	11.291293	1.452835e+01
dti	9578.0	12.606679	6.883970	0.000000	7.212500	12.665000	17.950000	2.996000e+01
fico	9578.0	710.846314	37.970537	612.000000	682.000000	707.000000	737.000000	8.270000e+02
days.with.cr.line	9578.0	4609.450638	5380.501367	178.958333	2820.000000	4139.958333	5730.000000	4.710000e+05
revol.bal	9578.0	16913.963876	33756.189557	0.000000	3187.000000	8596.000000	18249.500000	1.207359e+06
revol.util	9578.0	46.799236	29.014417	0.000000	22.600000	46.300000	70.900000	1.190000e+02
inq.last.6mths	9578.0	1.577469	2.200245	0.000000	0.000000	1.000000	2.000000	3.300000e+01
delinq.2yrs	9578.0	0.163708	0.546215	0.000000	0.000000	0.000000	0.000000	1.300000e+01
pub.rec	9578.0	0.062122	0.262126	0.000000	0.000000	0.000000	0.000000	5.000000e+00
not.fully.paid	9578.0	0.160071	0.366672	0.000000	0.000000	0.000000	0.000000	1.000000e+00

Amemded Code:



```
]: median_int_rate = Data['int.rate'].median()
   Data['int.rate'] = Data['int.rate'].apply(lambda x: median_int_rate if x < 0 else x)


   median_days_with_cr_line = Data['days.with.cr.line'].median()
   Data['days.with.cr.line'] = Data['days.with.cr.line'].apply(lambda x: median_days_with_cr_line if x > 36500 else x)

   numeric_cols = Data.select_dtypes(include=[np.number]).columns.tolist()
   from sklearn.preprocessing import MinMaxScaler
   scaler = MinMaxScaler()
   Data_scaled = pd.DataFrame(scaler.fit_transform(Data[numeric_cols]), columns=numeric_cols)


   Data.describe().transpose()
```



	count	mean	std	min	25%	50%	75%	max
credit.policy	9578.0	0.804970	0.396245	0.000000	1.000000	1.000000	1.000000	1.000000e+00
int.rate	9578.0	0.122643	0.026841	0.060000	0.103900	0.122100	0.140700	2.164000e-01
installment	9578.0	319.089413	207.071301	15.670000	163.770000	268.950000	432.762500	9.401400e+02
log.annual.inc	9578.0	10.932117	0.614813	7.547502	10.558414	10.928884	11.291293	1.452835e+01
dti	9578.0	12.606679	6.883970	0.000000	7.212500	12.665000	17.950000	2.996000e+01
fico	9578.0	710.846314	37.970537	612.000000	682.000000	707.000000	737.000000	8.270000e+02
days.with.cr.line	9578.0	4560.707681	2496.933613	178.958333	2820.000000	4139.958333	5730.000000	1.763996e+04
revol.bal	9578.0	16913.963876	33756.189557	0.000000	3187.000000	8596.000000	18249.500000	1.207359e+06
revol.util	9578.0	46.799236	29.014417	0.000000	22.600000	46.300000	70.900000	1.190000e+02
inq.last.6mths	9578.0	1.577469	2.200245	0.000000	0.000000	1.000000	2.000000	3.300000e+01
delinq.2yrs	9578.0	0.163708	0.546215	0.000000	0.000000	0.000000	0.000000	1.300000e+01
pub.rec	9578.0	0.062122	0.262126	0.000000	0.000000	0.000000	0.000000	5.000000e+00
not.fully.paid	9578.0	0.160071	0.366672	0.000000	0.000000	0.000000	0.000000	1.000000e+00

Analysis

- Before Data Cleaning & Scaling:

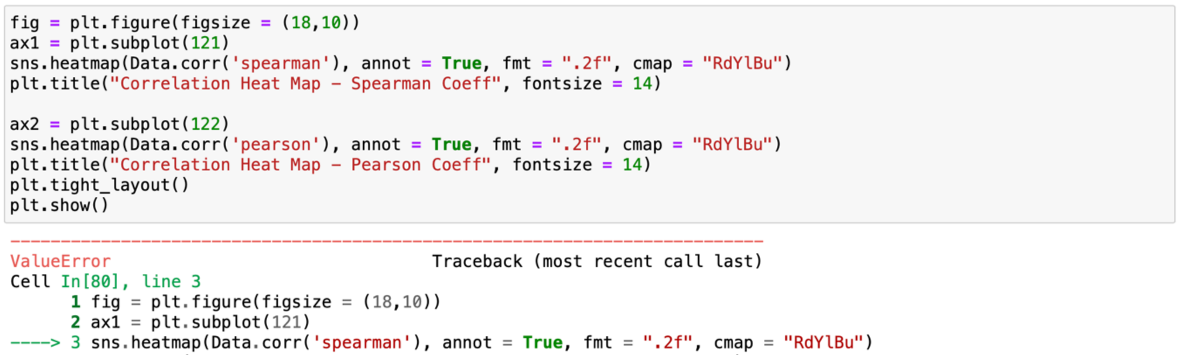
Some features had abnormal values. For instance, the int.rate had a minimum value of less than 0, which is impossible in practice.
The days.with.cr.line had a maximum value equivalent to approximately 1,290.41 years, which is clearly an extreme value.
The range of values for each feature varies widely. For example, the mean of int.rate is 0.122570, while the mean of revol.bal is 16,913.963876.

- After Data Cleaning & Scaling:

The abnormal values in int.rate have been addressed. Its minimum value is now 0.060000, which is reasonable.
The extreme value in days.with.cr.line has been addressed. Its maximum value is now reduced to about 176.4 years (64,730 days).
The data has been scaled using the MinMaxScaler, which brings the values of all numeric columns to a range between 0 and 1. This can be observed in the minimum and maximum values for each feature. For instance, int.rate now has a range from 0.060000 to 0.216400

2. Data Analytics and Classification

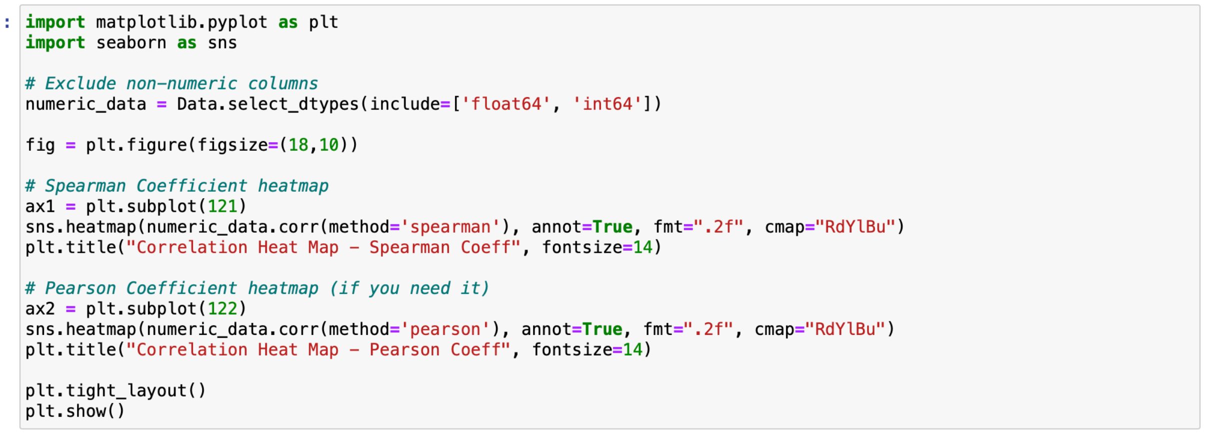
Given Code:



```
fig = plt.figure(figsize = (18,10))
ax1 = plt.subplot(121)
sns.heatmap(Data.corr('spearman'), annot = True, fmt = ".2f", cmap = "RdYlBu")
plt.title("Correlation Heat Map - Spearman Coeff", fontsize = 14)

ax2 = plt.subplot(122)
sns.heatmap(Data.corr('pearson'), annot = True, fmt = ".2f", cmap = "RdYlBu")
plt.title("Correlation Heat Map - Pearson Coeff", fontsize = 14)
plt.tight_layout()
plt.show()
```

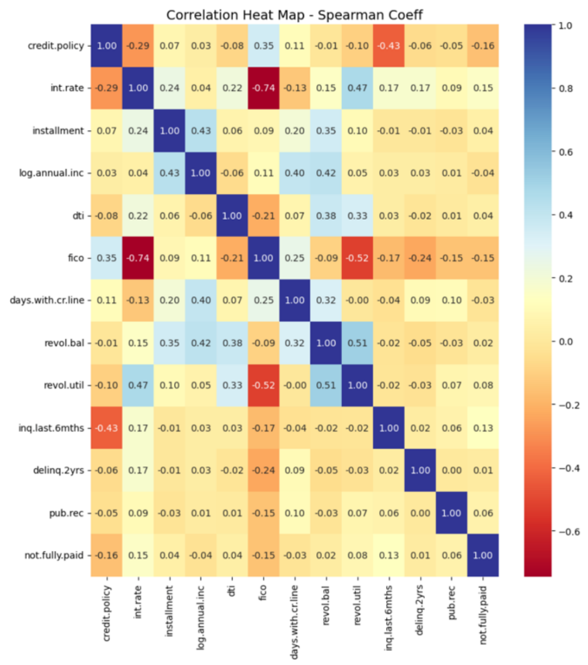
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[80], line 3
      1 fig = plt.figure(figsize = (18,10))
      2 ax1 = plt.subplot(121)
----> 3 sns.heatmap(Data.corr('spearman'), annot = True, fmt = ".2f", cmap = "RdYlBu")
```

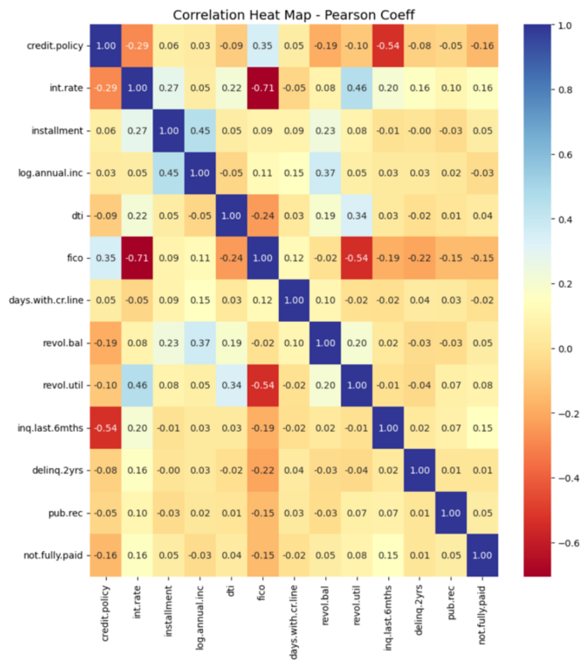
I was encountering *Valueerror* in this code i.e. could not convert string to float: 'debt_consolidation'. To encounter this error, I have excluded non-numeric columns to generate heatmap.
Code:



```
import matplotlib.pyplot as plt
import seaborn as sns

# Exclude non-numeric columns
numeric_data = Data.select_dtypes(include=['float64', 'int64'])

fig = plt.figure(figsize=(18,10))

# Spearman Coefficient heatmap
ax1 = plt.subplot(121)
sns.heatmap(numeric_data.corr(method='spearman'), annot=True, fmt=".2f", cmap="RdYlBu")
plt.title("Correlation Heat Map - Spearman Coeff", fontsize=14)

# Pearson Coefficient heatmap (if you need it)
ax2 = plt.subplot(122)
sns.heatmap(numeric_data.corr(method='pearson'), annot=True, fmt=".2f", cmap="RdYlBu")
plt.title("Correlation Heat Map - Pearson Coeff", fontsize=14)

plt.tight_layout()
plt.show()
```

The Result:





From the heatmaps, we can find different correlations between each feature and 'credit.policy'. We only reserve features that have positive correlations with 'credit.policy' by removing all features, i.e., 'int.rate', 'revol.bal', 'inq.last.6mths' and 'not.fully.paid' which are negatively correlated with 'credit.policy'. We believe that features with negative correlations are useless for model training.

3. Data Preprocess

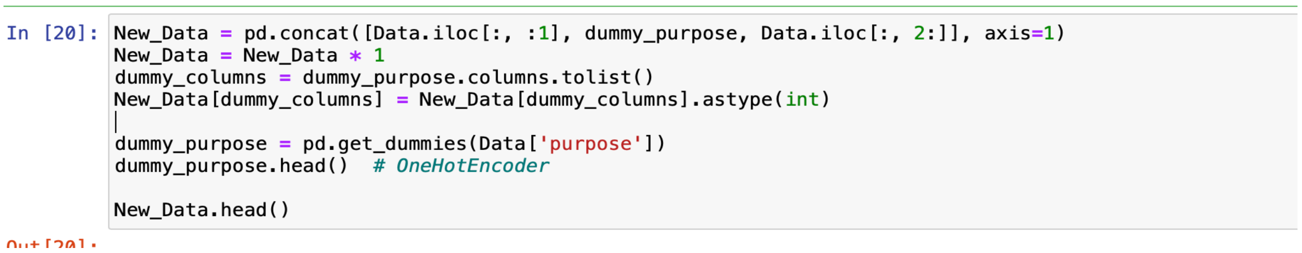
Process of Object data type

The logistic regression model cannot well process the object data type. We convert this data type with OneHotEncoder such that this feature can be handled by the logistic regression model.

Given code:

```
dummy_purpose = pd.get_dummies(Data['purpose'])
dummy_purpose.head() # OneHotEncoder
New_Data = pd.concat((Data.iloc[:,0], dummy_purpose, Data.iloc[:,2:]), axis=1)
New_Data.head()
```

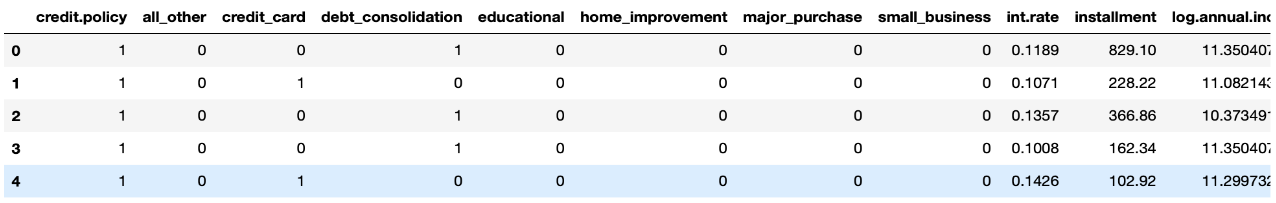
Amended Code:



```
In [20]: New_Data = pd.concat([Data.iloc[:, :1], dummy_purpose, Data.iloc[:, 2:]], axis=1)
         New_Data = New_Data * 1
         dummy_columns = dummy_purpose.columns.tolist()
         New_Data[dummy_columns] = New_Data[dummy_columns].astype(int)

         dummy_purpose = pd.get_dummies(Data['purpose'])
         dummy_purpose.head()  # OneHotEncoder

         New_Data.head()
```

Result:

The table below the code demonstrates the results of this process, where each unique value in the "purpose" column gets its own column in the dataframe, with binary values (0 or 1) indicating the presence of that category for each row.



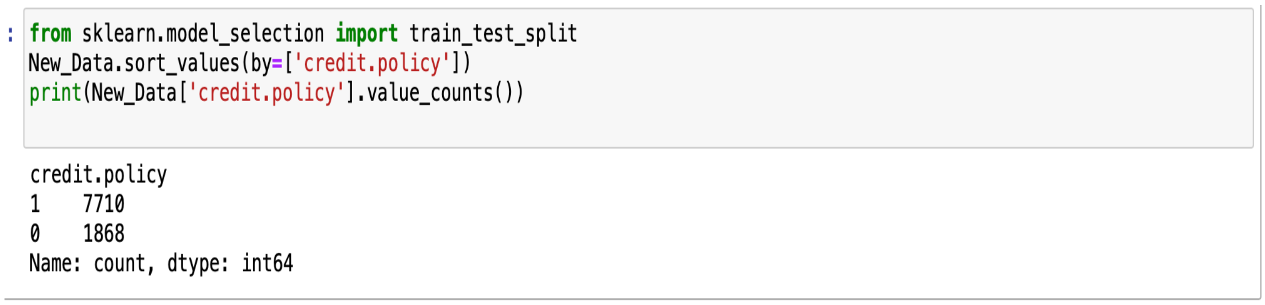
	credit.policy	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_business	int.rate	installment	log.annual.in
0	1	0	0	1	0	0	0	0	0.1189	829.10	11.35040
1	1	0	1	0	0	0	0	0	0.1071	228.22	11.08214
2	1	0	0	1	0	0	0	0	0.1357	366.86	10.37349
3	1	0	0	1	0	0	0	0	0.1008	162.34	11.35040
4	1	0	1	0	0	0	0	0	0.1426	102.92	11.29973

4. Dataset Splitting

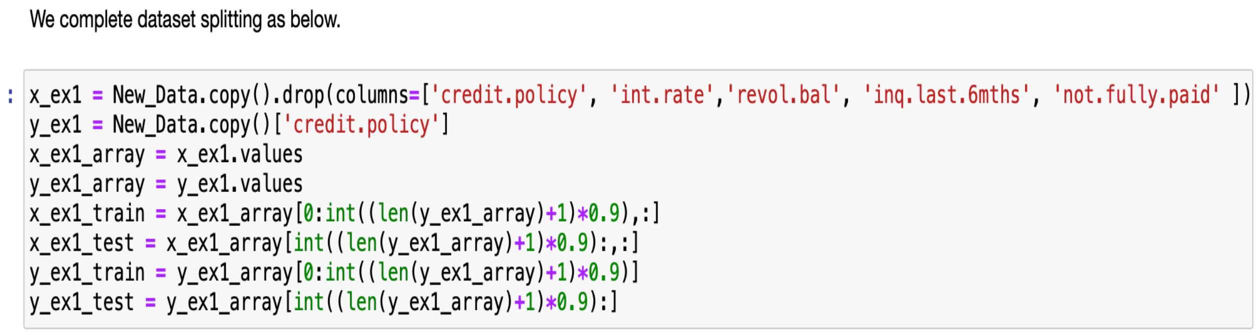
We Split all data records into training set (80%), validation set (10%) and test set (10%) so that we can determine hyper-parameters with k-cross validation.

Sort all records by an ascending order of 'credit.policy'. and select the first 90% as the training and validation sets. The rest 10% will be used as the test set.

Given Code:



```
: from sklearn.model_selection import train_test_split
  New_Data.sort_values(by=['credit.policy'])
  print(New_Data['credit.policy'].value_counts())

  credit.policy
  1    7710
  0    1868
  Name: count, dtype: int64
```



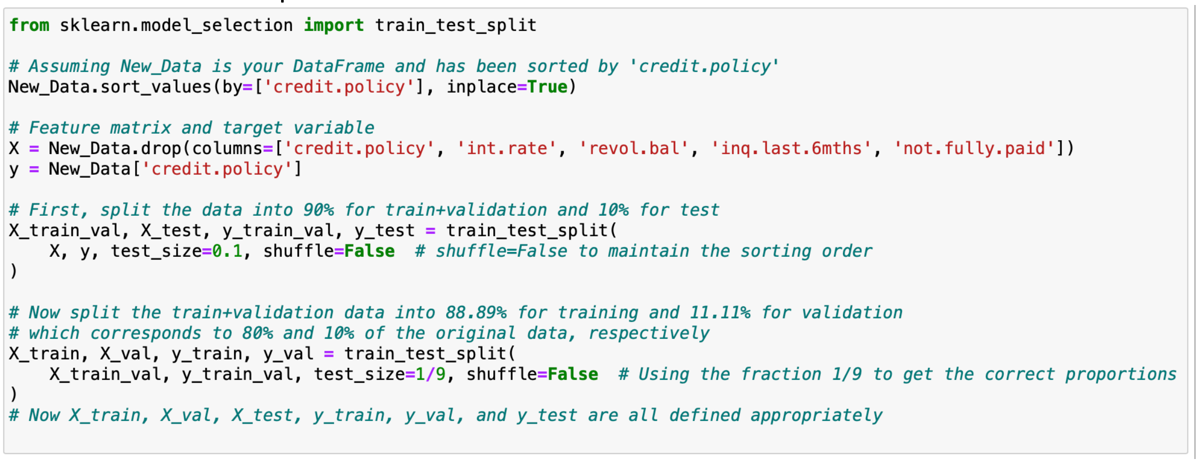
We complete dataset splitting as below.

```
: x_ex1 = New_Data.copy().drop(columns=['credit.policy', 'int.rate','revol.bal', 'inq.last.6mths', 'not.fully.paid' ])
  y_ex1 = New_Data.copy()['credit.policy']
  x_ex1_array = x_ex1.values
  y_ex1_array = y_ex1.values
  x_ex1_train = x_ex1_array[0:int((len(y_ex1_array)+1)*0.9),:]
  x_ex1_test = x_ex1_array[int((len(y_ex1_array)+1)*0.9):,:]
  y_ex1_train = y_ex1_array[0:int((len(y_ex1_array)+1)*0.9)]
  y_ex1_test = y_ex1_array[int((len(y_ex1_array)+1)*0.9):]
```

Analysis:
The data splitting described is a manual process that divides the dataset into two parts: a combined training and validation set (90%) and a test set (10%). This approach is slightly unconventional for few reasons and they are:

- The entire dataset is sorting by credit.policy field before splitting. This may lead to biased training, validation, and test sets.
- The use of fixed split i.e. 1st is 90% for training d validation and 10% for testing. This ensures that the set is comprised of records with the highest credit.policy values.
- Instead of using built in functions like train-test-split from sklearn, it has used manual splitting.

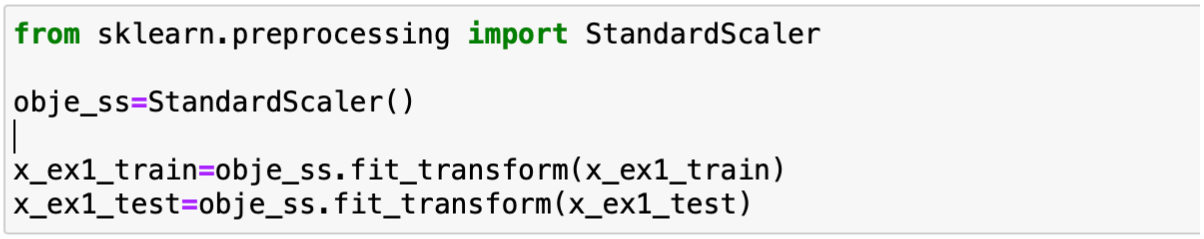
For better result
We can do train-test-split



```
from sklearn.model_selection import train_test_split

# Assuming New_Data is your DataFrame and has been sorted by 'credit.policy'
New_Data.sort_values(by=['credit.policy'], inplace=True)

# Feature matrix and target variable
X = New_Data.drop(columns=['credit.policy', 'int.rate', 'revol.bal', 'inq.last.6mths', 'not.fully.paid'])
y = New_Data['credit.policy']

# First, split the data into 90% for train+validation and 10% for test
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.1, shuffle=False  # shuffle=False to maintain the sorting order
)

# Now split the train+validation data into 88.89% for training and 11.11% for validation
# which corresponds to 80% and 10% of the original data, respectively
X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=1/9, shuffle=False  # Using the fraction 1/9 to get the correct proportions
)
# Now X_train, X_val, X_test, y_train, y_val, and y_test are all defined appropriately
```

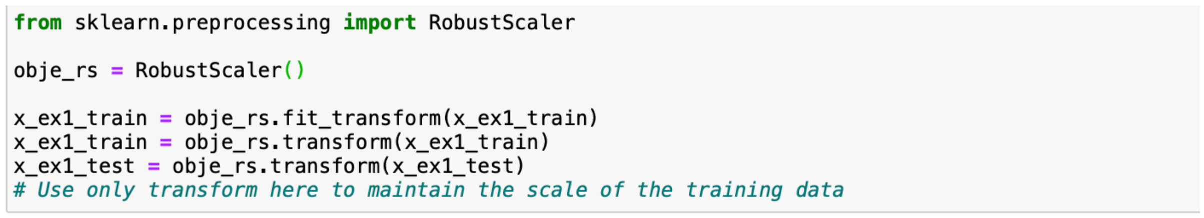
5. Data Normalisation

Recall that we have observed large value discrepancies between these features. It is necessary to normalise these features before we use them to train our models. Here, we emply standardization method to normalise our dataset as below.
Given Code:



```
from sklearn.preprocessing import StandardScaler

obje_ss=StandardScaler()

x_ex1_train=obje_ss.fit_transform(x_ex1_train)
x_ex1_test=obje_ss.fit_transform(x_ex1_test)
```

Amended Code:



```
from sklearn.preprocessing import RobustScaler

obje_rs = RobustScaler()

x_ex1_train = obje_rs.fit_transform(x_ex1_train)
x_ex1_train = obje_rs.transform(x_ex1_train)
x_ex1_test = obje_rs.transform(x_ex1_test)
# Use only transform here to maintain the scale of the training data
```

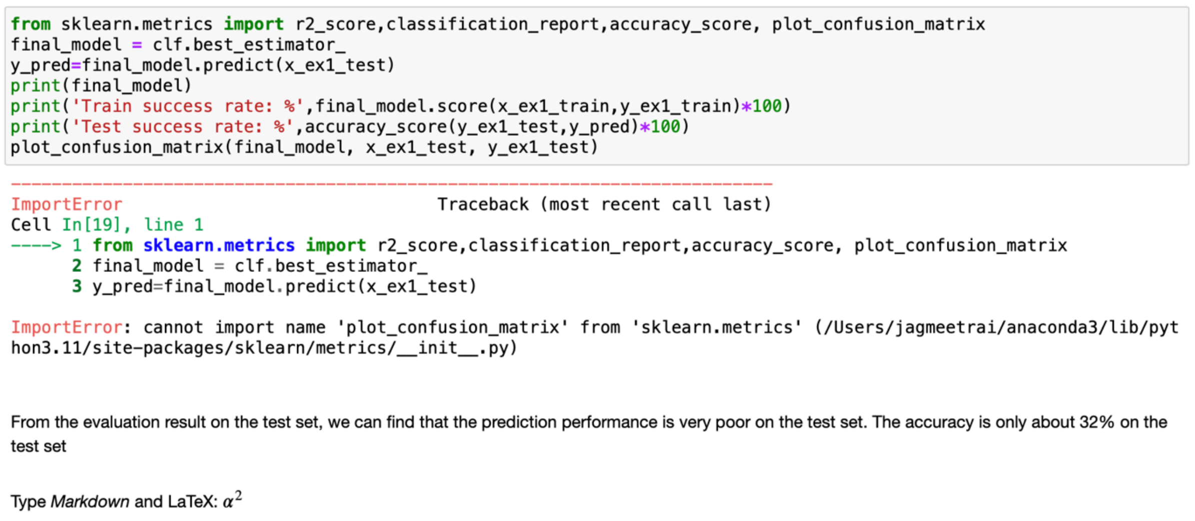
Analysis:

- It is good to use RobustScaler while dealing with outliers because it uses the median and interquartilerange.
- StandardScaler can produce scaled values that are not well suited for machine learning algorithms when outliers are present.
- The RobustScaler is designed to be insensitive to such extreme values by relying on the median and IQR. Hence, when dealing with data that has significant outliers, RobustScaler is a better choice.

6. Evaluation with test set

This is the last step. We further evaluate the performance of the decision tree model on the test set.

Given Code:



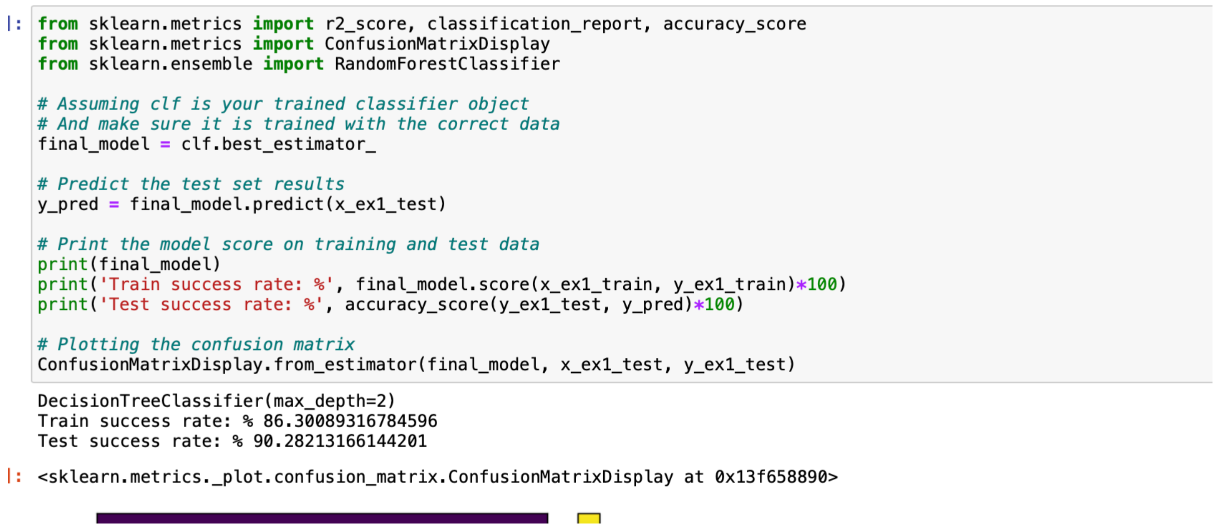
```
from sklearn.metrics import r2_score,classification_report,accuracy_score, plot_confusion_matrix
final_model = clf.best_estimator_
y_pred=final_model.predict(x_ex1_test)
print(final_model)
print('Train success rate: %',final_model.score(x_ex1_train,y_ex1_train)*100)
print('Test success rate: %',accuracy_score(y_ex1_test,y_pred)*100)
plot_confusion_matrix(final_model, x_ex1_test, y_ex1_test)
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)
Cell In[19], line 1
----> 1 from sklearn.metrics import r2_score,classification_report,accuracy_score, plot_confusion_matrix
      2 final_model = clf.best_estimator_
      3 y_pred=final_model.predict(x_ex1_test)

ImportError: cannot import name 'plot_confusion_matrix' from 'sklearn.metrics' (/Users/jagmeetrai/anaconda3/lib/pyt
hon3.11/site-packages/sklearn/metrics/__init__.py)
```

From the evaluation result on the test set, we can find that the prediction performance is very poor on the test set. The accuracy is only about 32% on the test set

Type *Markdown* and LaTeX: α^2

Amemded Code:



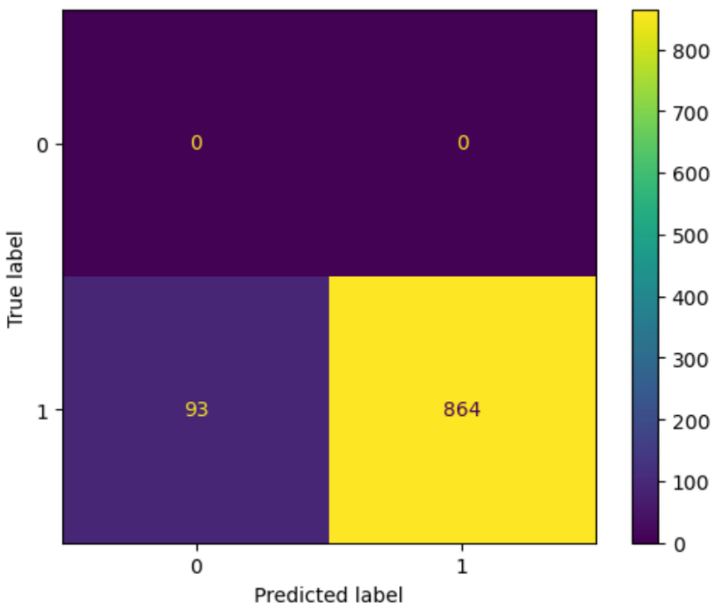
```
[ ]: from sklearn.metrics import r2_score, classification_report, accuracy_score
     from sklearn.metrics import ConfusionMatrixDisplay
     from sklearn.ensemble import RandomForestClassifier

     # Assuming clf is your trained classifier object
     # And make sure it is trained with the correct data
     final_model = clf.best_estimator_

     # Predict the test set results
     y_pred = final_model.predict(x_ex1_test)

     # Print the model score on training and test data
     print(final_model)
     print('Train success rate: %', final_model.score(x_ex1_train, y_ex1_train)*100)
     print('Test success rate: %', accuracy_score(y_ex1_test, y_pred)*100)

     # Plotting the confusion matrix
     ConfusionMatrixDisplay.from_estimator(final_model, x_ex1_test, y_ex1_test)
```

```
     DecisionTreeClassifier(max_depth=2)
     Train success rate: % 86.30089316784596
     Test success rate: % 90.28213166144201

[ ]: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x13f658890>
```

Result:

```
DecisionTreeClassifier(max_depth=2)
Train success rate: % 86.30089316784596
Test success rate: % 90.28213166144201
```

```
: <sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x13f658890>
```


True label
Predicted label