COMP6200: DATA SCIENCE

CRITICAL ANALYSIS REPORT

Name: Mai Vy (Vivian) Nguyen

Student ID: 47554029

Table of Contents

Introduction

The Jupyter Notebook critically analyzes the processing of the LendingClub dataset, highlighting the importance of handling the dataset to ensure efficient and accurate predictive modelling. There are some steps that can further refine the processes of the dataset and the model's efficiency. This report aims to identify, explain, and provide solutions to some of these issues, ensuring that the Jupyter Notebook for LendingClub dataset meets the highest standards of data processing and machine learning modelling.

Issues and Solution

1. Handling Missing Values

```python
Data['not.fully.paid'] = Data['not.fully.paid'].fillna(Data['not.fully.paid'].mean())
Data.info()
```

Python

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   credit.policy      9578 non-null   int64
 1   purpose            9578 non-null   object
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64
 11  delinq.2yrs        9578 non-null   int64
 12  pub.rec            9578 non-null   int64
 13  not.fully.paid     9578 non-null   float64
dtypes: float64(7), int64(6), object(1)
memory usage: 1.0+ MB
```

Issue: The method used to resolve missing values in the "not.fully.paid" attribute causes problems that affect the consistency of values in the column. Replacing a null value with an average value is inappropriate for this attribute, due to its obvious binary nature of representing 0 or 1. The average provided a different value than other values that can distort the clarity and meaning of the feature "not.fully.paid" to answer whether the borrower will be fully paid or not.

Solution: A more efficient approach for such binary attributes would be to exploit the most frequently occurring mode to substitute for any missing items. Taking advantage of this mode ensures that the values specified are consistent with the inherent binary classification, thereby maintaining the integrity and clarity of the feature.

Here is my fix code:

```
Data['not.fully.paid'] = Data['not.fully.paid'].fillna(Data['not.fully.paid'].mode()[0])
Data.info()
```

✓ 0.0s

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 9578 entries, 0 to 9577
Data columns (total 14 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   credit.policy      9578 non-null   int64  
 1   purpose            9578 non-null   object 
 2   int.rate           9578 non-null   float64
 3   installment        9578 non-null   float64
 4   log.annual.inc     9578 non-null   float64
 5   dti                9578 non-null   float64
 6   fico               9578 non-null   int64  
 7   days.with.cr.line  9578 non-null   float64
 8   revol.bal          9578 non-null   int64  
 9   revol.util         9578 non-null   float64
 10  inq.last.6mths     9578 non-null   int64  
 11  delinq.2yrs        9578 non-null   int64  
 12  pub.rec            9578 non-null   int64  
 13  not.fully.paid     9578 non-null   float64
dtypes: float64(7), int64(6), object(1)
memory usage: 1.0+ MB
```

2. Missing the step of removing outlier

```
Data.describe().transpose()
✓ 0.0s                                                                 Python
```

	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	9577.0	0.160071	0.366691	0.000000	0.000000	0.000000	0.000000	1.000000e+00

Issue: The dataset contains some outliers. For example, the minimal value of "int.rate" is less than 0, which is impossible in practice. The maximum value of "days.with.cr.line" is 4.710000e5 days, which is comparable to 1,290.41 years, which is also unusual. To clean our dataset, it would be correct to have these records with these severe values.

Solution: Firstly, any instances where the "int.rate" is less than 0 are logically implausible and should be considered outliers. Secondly, it's highly unlikely for any loan to span more than 100 years. Hence, entries where "days.with.cr.line" exceeds 36,500 (representing days in 100 years) are anomalies. For the "int.rate" outliers, replacing the values less than 0 with the median interest rate of the loan is a suitable solution. Similarly, for the "days.with.cr.line" outliers exceeding 36,500 days, substituting these with the median of "days.with.cr.line" provides a robust remedy. This ensures that the revised values are representative and won't significantly sway the dataset's distribution.

This can be executed using:

```
# Check the data that interest rate is less than 0
Data.loc[Data['int.rate']<0, 'int.rate']
✓ 0.0s

8       -0.1134
6370    -0.0740
9573    -0.1461
Name: int.rate, dtype: float64
```

```
# Check the data that loan larger than 100 years
Data.loc[Data['days.with.cr.line']>36500, 'days.with.cr.line']
✓ 0.0s

2    471000.0
Name: days.with.cr.line, dtype: float64
```

```
Data.loc[Data['int.rate']<0,'int.rate']=Data['int.rate'].median()
Data.loc[Data['days.with.cr.line']>36500, 'days.with.cr.line']=Data['days.with.cr.line'].median()
Data.describe().transpose()

✓ 0.0s
```

	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

3. Data Splitting

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

```
1    7710
0    1868
Name: credit.policy, 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):]
```

Issue: The current method drops features like “int.rate”, “revol.bal”, “inq.last.6mths”, “not.fully.paid” without any clear justification or evidence that these features are not significant predictors of “credit.policy”. While manually sorting the dataset by 'credit.policy' and then using array slicing to partition the data into training and testing sets, this approach lacks randomness in the split. This could lead to potential biases, and certain classes might be overrepresented in one split while underrepresented in another.

Solution: Before removing any feature from the dataset, it's crucial to understand its significance concerning the target variable “credit.policy”. This can be achieved by investigating the correlation between “credit.policy” and other attributes. Features with a very low correlation might not be significant, but dropping them should be done with caution, ensuring that they genuinely do not contain useful information for the predictive task. Instead of manually partitioning the dataset, utilizing the train_test_split method from the sklearn.model_selection module ensures a random split, providing better generalization capabilities.

This is my suggested code:

```
# Check the correlation
New_Data.corr()
```

✓ 0.0s

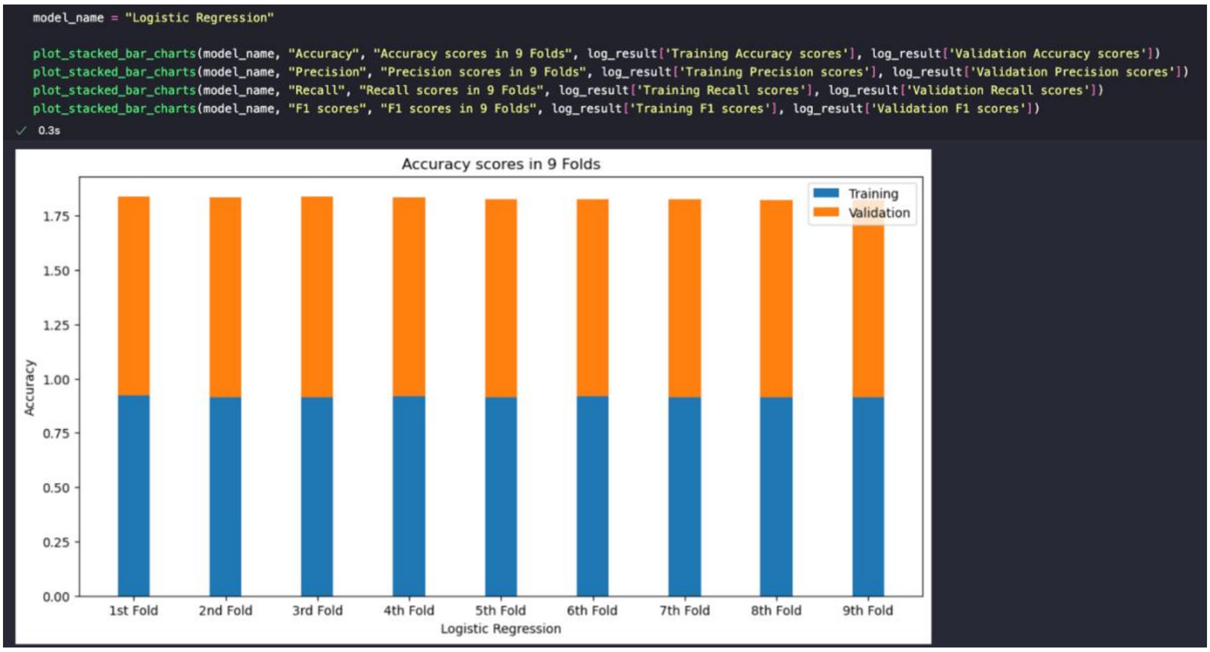
	credit.policy	all_other	credit_card	debt_consolidation	educational	home_improvement	major_purchase	small_business	int.rate
credit.policy	1.000000	-0.025412	0.003216	0.020193	-0.031346	0.006036	0.024281	-0.003511	-0.289093
all_other	-0.025412	1.000000	-0.220935	-0.475848	-0.109300	-0.150359	-0.124004	-0.149076	-0.125046
credit_card	0.003216	-0.220935	1.000000	-0.326850	-0.075076	-0.103279	-0.085176	-0.102397	-0.040620
debt_consolidation	0.020193	-0.475848	-0.326850	1.000000	-0.161698	-0.222441	-0.183451	-0.220542	0.124319
educational	-0.031346	-0.109300	-0.075076	-0.161698	1.000000	-0.051094	-0.042138	-0.050658	-0.018896
home_improvement	0.006036	-0.150359	-0.103279	-0.222441	-0.051094	1.000000	-0.057967	-0.069687	-0.052947
major_purchase	0.024281	-0.124004	-0.085176	-0.183451	-0.042138	-0.057967	1.000000	-0.057472	-0.067614
small_business	-0.003511	-0.149076	-0.102397	-0.220542	-0.050658	-0.069687	-0.057472	1.000000	0.150160
int.rate	-0.289093	-0.125046	-0.040620	0.124319	-0.018896	-0.052947	-0.067614	0.150160	1.000000
installment	0.058770	-0.203103	0.000774	0.161658	-0.094510	0.023024	-0.079836	0.145654	0.274127
log.annual.inc	0.034906	-0.080077	0.072942	-0.026214	-0.119799	0.116375	-0.031020	0.091540	0.052440
dti	-0.090901	-0.125825	0.084476	0.179149	-0.035325	-0.092788	-0.077719	-0.069245	0.218020
fico	0.348319	0.067184	-0.012512	-0.154132	-0.013012	0.097474	0.067129	0.063292	-0.705605
days.with.cr.line	0.050409	-0.031386	0.017924	0.006460	-0.021523	0.029198	-0.011520	0.013810	-0.053766
revol.bal	-0.187518	-0.067728	0.072316	0.005785	-0.034743	0.003258	-0.062395	0.083069	0.083600
revol.util	-0.104095	-0.138535	0.091321	0.211869	-0.053128	-0.114449	-0.108079	-0.060962	0.458499
inq.last.6mths	-0.535511	0.017795	-0.033640	-0.044240	0.024243	0.043827	-0.001445	0.042567	0.200582
delinq.2yrs	-0.076318	0.016658	-0.008817	-0.000697	-0.002214	-0.013098	0.004085	-0.004148	0.155030
pub.rec	-0.054243	-0.030451	0.014842	0.026845	-0.013521	0.004704	-0.011734	-0.005595	0.097627
not.fully.paid	-0.158098	0.009207	-0.047154	-0.017488	0.021601	0.007260	-0.028590	0.084449	0.155746

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(New_Data.drop(columns=['credit.policy']), New_Data['credit.policy'], test_size = 0.1, random_state = 42)
print('X_train:',X_train.shape)
print('y_train:',y_train.shape)
print('X_test:',X_test.shape)
print('y_test:',y_test.shape)
```

✓ 0.0s

```
X_train: (8620, 19)
y_train: (8620,)
X_test: (958, 19)
y_test: (958,)
```

4. Visualization in Logistic Regression Model



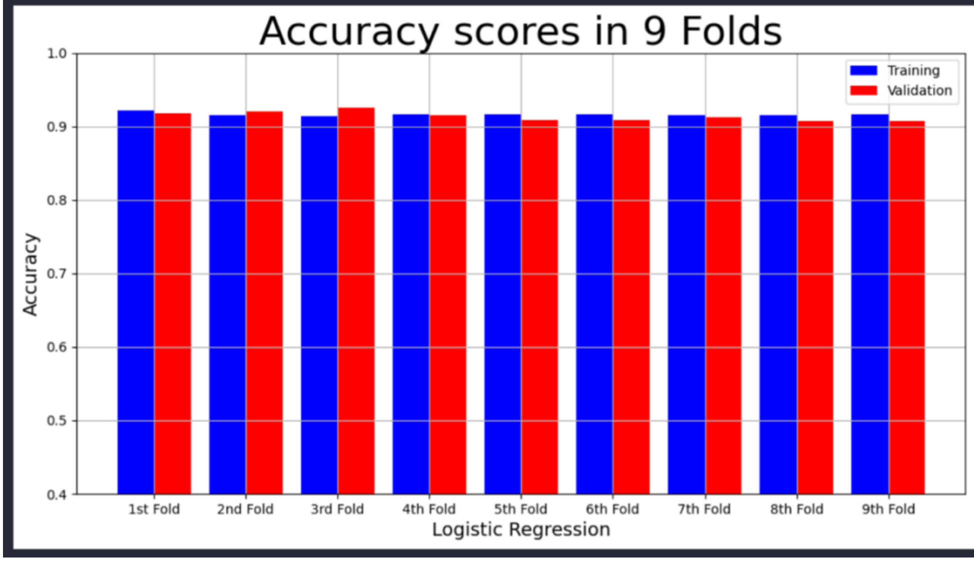
Issue: While stacked bar charts have their merits, in this context, they introduce a degree of ambiguity. The chief concern is the ability to draw accurate comparisons between the validation bars, which can be particularly challenging when the training bars exhibit even minor variances in height. The entire essence of visually comparing training and validation scores across multiple folds could become obscured, potentially leading to misinterpretations.

Solution: In order to address this issue, I will use grouped bar charts instead of stacked bar charts to visualize the result of the Logistic Regression Model. This would facilitate an easier juxtaposition of training and validation metrics, thus offering a more straightforward visual summary of the model's performance.

These can be identified with:

```
model_name = "Logistic Regression"

plot_bar_result(model_name, "Accuracy", "Accuracy scores in 9 Folds", log_result['Training Accuracy scores'], log_result['Validation Accuracy scores'])
plot_bar_result(model_name, "Precision", "Precision scores in 9 Folds", log_result['Training Precision scores'], log_result['Validation Precision scores'])
plot_bar_result(model_name, "Recall", "Recall scores in 9 Folds", log_result['Training Recall scores'], log_result['Validation Recall scores'])
plot_bar_result(model_name, "F1 scores", "F1 scores in 9 Folds", log_result['Training F1 scores'], log_result['Validation F1 scores'])
```
✓ 0.3s



Conclusion

Critical analysis of any data science project is pivotal for its success, as it offers a pathway to rectify errors and make necessary enhancements. The Jupyter Notebook on the LendingClub dataset serves as a foundational step towards predicting a borrower's ability to repay loans based on various features. However, as illustrated in this report, several technicalities need refinement to optimize the project's quality. By addressing the identified issues, from handling missing values and outliers to ensuring a rigorous data-splitting method and adopting clear visualization techniques, I can enhance the accuracy and reliability of the prediction model.