COMP4211 Programming Assignment 1 Report

Name: CHAN, Chun Hin

Student ID: 20853893

Email: chchanec@connect.ust.hk

Table of Contents

Part 1: Data Exploration and Preparation 3

Part 2: Data Preprocessing Techniques 14

Part 3: Regression 24

Part 4: Classification 33

Part 5: Performance Enhancement 45

Part 1: Data Exploration and Preparation

[Q1] Dataset Overview

Size of the Dataset:
There are 3539 instances in total.
There are 31 features in total.

Feature Types:
Numerical features are columns C6, C14, C16, C17, C18, C19, C20, C21, C22, C23, C24, C25, C26, C27, C28, C29, C30.
Categorical features are columns C0, C1, C2, C3, C4, C5, C7, C8, C9, C10, C11, C12, C13, C15.

[Q2] Missing Values

Identification:
Features with missing values are columns C0, C4, C5, C8, C9, C11, C12, C13, C15, C17, C20, C22, C23, C25, C29.
Proposition of missing values for the columns are as follows:
C0: 3539 – 3510 = 29
C4: 3539 – 3490 = 49
C5: 3539 – 3431 = 108
C8: 3539 – 3457 = 82
C9: 3539 – 3397 = 142
C11: 3539 – 3381 = 158
C12: 3539 – 3369 = 170
C13: 3539 – 3419 = 120
C15: 3539 – 3401 = 138
C17: 3539 – 3391 = 148
C20: 3539 – 3368 = 171
C22: 3539 – 3523 = 16
C23: 3539 – 3511 = 28
C25: 3539 – 3395 = 144
C29: 3539 – 3379 = 160

Potential Impact:

The missing values will lead to a biased analysis. It is because if those missing values are related to some important features or even target features, then the analysis will be biased. The bias problem will become more serious if a few features contain too many missing values and need to be removed. This will make the size of the dataset become smaller. This can possibly seriously affect the analysis result.
Also, this will worsen the model performance. It is because most of the machine learning algorithms fail to handle the dataset with missing values. In addition, if we cannot properly handle those missing values, then the model is likely to fail to learn the critical relationship between different features and also the important patterns. This can result in a inaccurate learning and a very bad model performance.

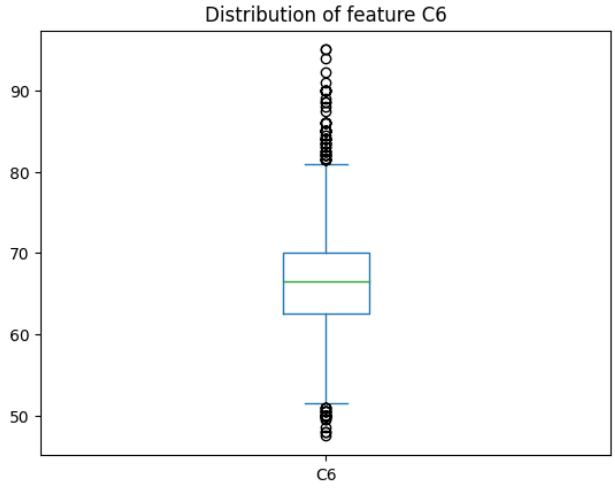
[Q3] Feature Distribution

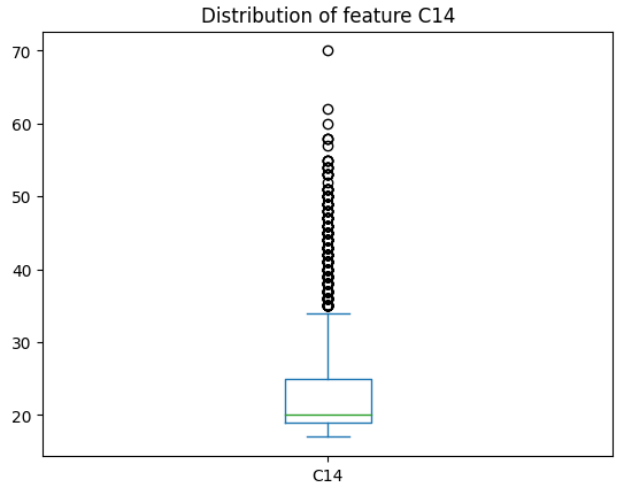
Numerical Features:
The features that are discrete are columns C14, C16, C17, C18, C19, C21, C22, C23, C24, C25, C27.
The features that are continuous are columns C6, C20, C26, C28, C29, C30

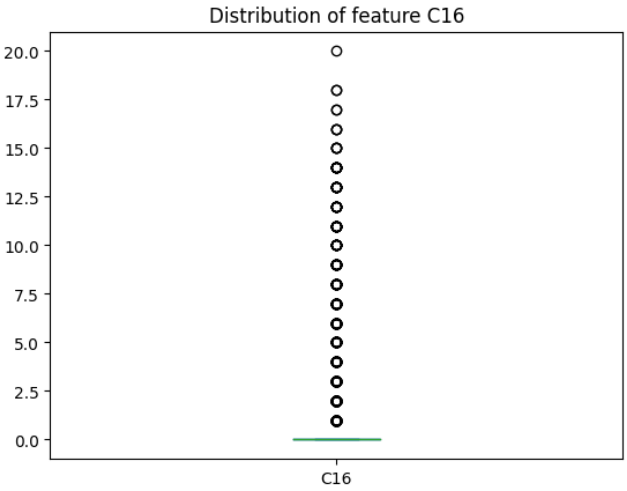
The distribution of the first three numerical features (C6, C14, C16):

Distribution\Features	C6	C14	C16
Mean	66.325	23.204	0.684
Median	66.55	20	0
Range	47.5	53	20
Variance	43.824	55.179	5.176

Visualization of the distribution of feature C6, C14, C16 (from up to down in order):







Categorical Features:

The features that are binary are columns C4, C8, C9, C10, C11, C12, C13, C15.

The features that are nominal are columns C0, C1, C3, C4, C7, C8, C9, C10, C11, C12, C13, C15.

The features that are ordinal are columns C2, C5.

The distribution of the first three categorical features (C0, C1, C2):

The distribution of feature C0:

Category	Count
single	3115
married	296
divorced	75
NaN (Not a Number)	29
facto union	18
legally separated	3

widower	3

The distribution of feature C1:

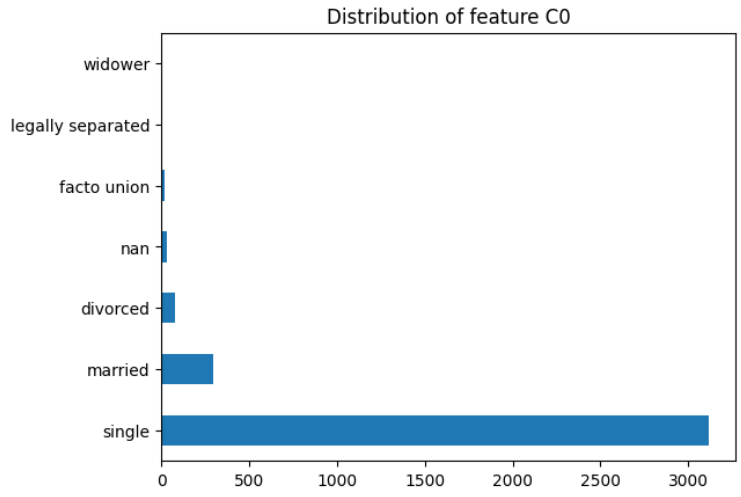
Category	Count
1st phase - general contingent	1351
2nd phase - general contingent	708
Over 23 years old	630
Change of course	252
Technological specialization diploma holders	160
Holders of other higher courses	109
3rd phase - general contingent	105
Transfer	67
Change of institution/course	45
Short cycle diploma holders	30
1st phase - special contingent (Madeira Island)	29
International student (bachelor)	26
1st phase - special contingent (Azores Island)	14
Ordinance No. 854-B/99	7
Ordinance No. 612/93	2
Ordinance No. 533-A/99, item b3 (Other Institution)	1
Ordinance No. 533-A/99, item b2) (Different Plan)	1
Change of institution/course (International)	1

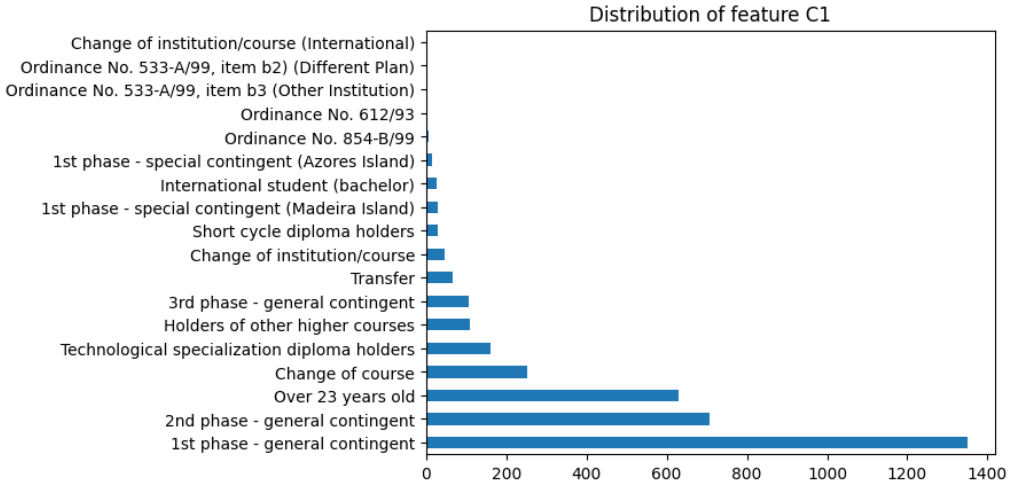
The distribution of feature C2:

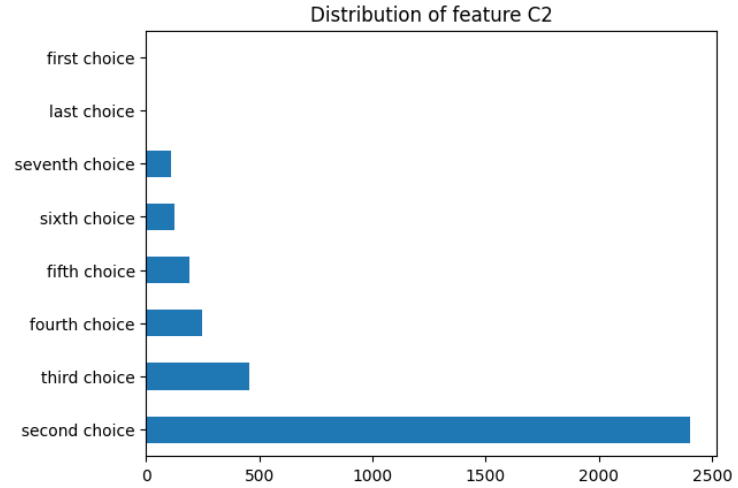
Category	Count
second choice	2402
third choice	457
fourth choice	247
fifth choice	194
sixth choice	125
seventh choice	112

last choice	1
first choice	1

Visualization of the distribution of feature C0, C1, C2 (from up to down in order):







[Q4] Outliers

Detection:

Method 1: Z-Score
The first method is to detect the outliers using Z-Score. In my code, I treat those data with Z-Score Z > 4 or Z<-4 as potential outliers.

By using the first method, there are 3 outliers in feature column C6, 22 outliers in feature column C14 and 68 outliers in feature column respectively. They are outliers because for feature column C6, the mean of feature C6 is around 66, a value of 95 is obviously too far away from the mean when the standard deviation is only around 6.6; for feature column C14, the mean is around 23, a value of 70 is obviously too far away from the mean when the standard deviation is only around 7.4; for feature column C16, the mean is around 0.68, a value of 18 is obviously too far away from the mean when the standard deviation is only around 2.2. Note that the proportion that the data points not within ± 4 standard deviation around the mean in normal distribution is only 0.0063%. This implies those outliers are deviated from the majority of the data by extreme extent.

Below are the results using Z-Score for features C6, C14, C16:

```
The potential outliers of feature C6 are:
N-th instances  value
1156    95.0
2282    94.0
3373    95.0
Name: C6, dtype: float64
```

```
The potential outliers of feature C14 are:
N-th instances  value
394     58
513     53
553     57
985     53
1244    54
1316    60
1388    55
1581    53
1647    54
1756    58
1891    55
2260    54
2450    53
2509    70
2827    54
2895    58
3048    53
3075    55
3077    55
3126    53
3187    62
3343    54
Name: C14, dtype: int64
```

```
The potential outliers of feature C16 are:
N-th instances  value
18      11
152     18
173     13
337     15
362     10
        ..
3068    11
3086    13
3273    14
3470    11
3500    11
Name: C16, Length: 68, dtype: int64
```

Method 2: Interquartile Range (IQR)

The second method is to detect the outliers using Interquartile Range (IQR). In my code, for each numerical features, I first find the 75th percentile (i.e. upper quartile Q3) and the 25th percentile (i.e. lower quartile Q1), and then I get IQR = Q3 – Q1. I treat those data outside the range of 2.5 times the IQR below Q1 or 2.5 times the IQR above Q3 as potential outliers.

By using the second method, there are 14 outliers in feature column C6, 171 outliers in feature column C14, 454 outliers in feature column C16 respectively. They are outliers because for the feature C6, 2.5 times the IQR is 7.5×2.5=18.75, so the cutoff line is 66.55+18.75=85.3, it is very obvious that a value of 95 is much larger than 85.3 when the IQR is just 7.5; for the feature C14, 2.5 times the IQR is 6×2.5=15, so the cutoff line is 20+15=35, it is very obvious that a value of 50 is much larger than 35 when the IQR is just 6; for the feature C16, 2.5 times the IQR is 2.5×0=0, so the cutoff line is 0+0=0, it is obvious that a value of 11 is much larger than 0 when the IQR is just 0, not evening mentioning at least 75% of data are 0. In general, people usually treat those data outside the range of 2 times the IQR below Q1 or 2 times the IQR above Q3 as potential outliers, so my setting is just looser than that and thus this is more confident to say they are outliers.

Below are the results using IQR for features C6, C14, C16:

```
The potential outliers of feature C6 are:
N-th instances  value
176     90.0
207     90.0
342     90.0
971     90.0
1156    95.0
1840    91.0
2272    89.0
2282    94.0
2448    92.2
2535    90.0
3259    90.0
3300    90.0
3373    95.0
3487    90.0
Name: C6, dtype: float64
```

```
The potential outliers of feature C14 are:
N-th instances  value
3       42
49      42
99      48
136     43
173     43
        ..
3388    44
3412    50
3415    43
3448    45
3463    41
Name: C14, Length: 171, dtype: int64
```

```
The potential outliers of feature C16 are:
N-th instances  value
5        2
12       2
14       1
17       3
18      11
        ..
3499     5
3500    11
3512     7
3528     8
3534     1
Name: C16, Length: 454, dtype: int64
```

Consideration:

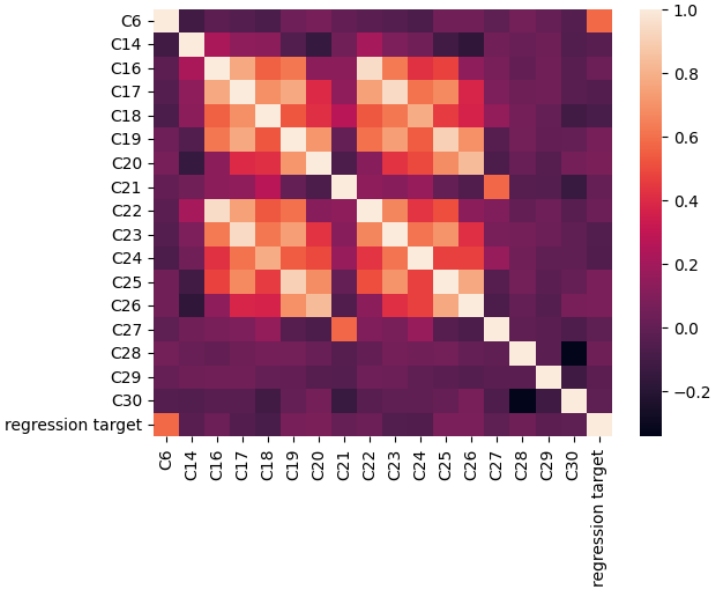
Note that mean, standard deviations and variance can greatly be affected by the extreme values (i.e. the potential outliers). Therefore, the outliers will make data point distribution distorted, making the data distribution left-skewed, right skewed, abnormal etc.. Therefore, this will require us to wisely choose some appropriate techniques to handle the outliers to minimize their impact to making an accurate model. Some common preprocessing techniques to handle the outliers are capping the outliers to predefined range, trimming and median imputation.

Also, the outliers can greatly affect us which models should we trained to get the most accurate result. It is because the existence of a few outliers can already make the corresponding model too complicated, which easily lead to overfitting and poor generalization. Some of the models are highly dependent on the data point distribution, for example, linear regression is highly depends on the sum of squared error (and also the cross-entropy loss). Since linear regression is trying to minimize the sum of squared error and also the cross-entropy loss, the outliers can greatly affect the best fit line of the model due to their extreme effect on the mean. Hence, the trained model will have a bad accuracy and estimator, which require us to avoid using linear regression when number of outliers are large.

[Q5] Correlation Analysis

Feature Correlation:

The correlation heatmap:



Insights:

The correlation between two features is stronger if the correlation number is closer to +1 (i.e. those blocks with lighter color). Based on the heatmap as shown above, there are some pairs of features are having strong correlations. They are: C17-C16 pair, C18-C17 pair, C19-C16 pair, C19-C17 pair, C22-C16 pair, C22-C17 pair, C23-C16 pair, C23-C17 pair, C23-C19 pair, C23-C22 pair, C24-C18 pair, C25-C17 pair, C25-C19 pair, C25-C20 pair, C25-C23 pair, C26-C19 pair, C26-C20 pair, C26-C25 pair, C27-C21 pair, regression target-C6 pair.

[Q6] Initial Thoughts on Preprocessing

Based on the exploration, briefly outline the preprocessing steps that are necessary for the regression models or neural networks to run:

In summary, there are 8 preprocessing steps that are necessary for the regression models or neural networks to run.

1. Get the dataset: We need to get the relevant training dataset and the test dataset for training the model and testing the accuracy of the model respectively. For example, if you need to build a model related to business analysis, you should use business datasets, but not other types of datasets.
2. Import the necessary libraries and packages: We need to import the relevant Python built-in libraries to process the dataset and do further analysis of them. We may also need those libraries to plot different graphs and build some models. Some popular libraries are NumPy, Pandas, Matplotlib, Seaborn, TensorFlow etc.

3. Import the dataset to your working environment/IDE: Most working environments/IDE would require you to import the dataset before you can explore the dataset briefly and extract the information and use them in your machine learning project.
4. Identifying and handling missing values: Most machine learning algorithms would not be able to handle the dataset with missing values. We need to identify which features contain missing values, and we need to properly handle them using different techniques, such as trimming and mean imputation. Ignoring missing values can lead to wrong generalization of the model.
5. Identifying the data distribution and handling outliers: Linear regression and some neural networks can be greatly affected by the outliers as they depend on the distribution of the data, especially the mean. We need to first know a bit about the distribution of each feature and see whether there are outliers in each feature using box and whisker diagram. After that, we should handle the outliers carefully so that the data distribution is less biased. Some example techniques are trimming and median imputation.
6. Normalization: Note that different features may have different ranges of numerical values. It would be better if we could normalize all of them so that all numerical data can transform to a similar scale.
7. Encoding the categorical features: Since machine learning models are more able to handle mathematics related data, it will be better to use one-hot encoding on those categorical data, which makes the development of the model easier.
8. Analysis the correlation and select the better features for modeling: Some of the features of the dataset may have a better correlation with other features or the regression target. We may use a correlation heatmap to have a better picture on that. Then, we select those features with greater correlation value so that the trained model can have a better performance, accuracy and generalization.

Briefly discuss any specific challenges identified during exploration that are better addressed in preprocessing (those that do not affect running models, but might affect model performance):

In general, there are challenges that are better addressed in preprocessing.

1. Data cleaning: The collected dataset may be dirty when you first receive it, for example, missing values and duplicated data. For missing data, it would be a challenge to decide which technique should we use to fill in or remove those missing values. For duplicated data, it would be a time-consuming task to identify all duplicated data and then remove them accordingly.

2. Handling outliers: Outliers are harmful to build a precise model. In order to handle them properly, we need to test different techniques to deal with them, to see which one is the best.
3. Feature selection: Although we can visualize the correlation among the features and the target, it is quite controversial on the definition of high correlation. For example, what is the threshold for the correlation value to determine whether a pair of features are highly related.
4. Normalization: The data can be quite diverse, like the range of the data can differ a lot. It can be hard for us to choose a suitable normalization technique to ensure all the data and features are in fair and consistent comparison.

Part 2: Data Preprocessing Techniques

[Q7] Handling Missing Values

Discuss their impact on feature distribution and model performance:

For mean imputation, this imputation strategy only works on numerical features, but not categorical features. Therefore, I will just discuss its impact on numerical features only. Mean imputation fills in the missing values with the mean of the available value in each numerical feature column. Although this can keep the mean, maximum, minimum, range of each feature column unchanged, this decreases both the standard deviation and the variance. This may change the median as well. Since mean imputation assumes the missing values are missing in fully random manner, so if the missing values are not missing in fully random manner or the feature column contains reasonable number of outliers, mean imputation can make the feature column biased and worsen the model performance.

For median imputation, this imputation strategy only works on numerical features, but not categorical features. Therefore, I will just discuss its impact on numerical features only. Median imputation fills in the missing values with the median of the available value in each numerical feature column. Although this can keep the median, maximum, minimum, range of each feature column unchanged, this decreases both the standard deviation and the variance. This may change the mean as well. Since median imputation assumes the missing values are missing in fully random manner, so if the missing values are not missing in fully random manner or the feature column contains reasonable number of outliers, median imputation can make the feature column biased and worsen the model performance.

For mode imputation, this imputation strategy works on both numerical features and categorical features. Therefore, I will first discuss its impact on numerical features, and follow by that on categorical features. Talking about numerical features, mode imputation fills in the missing values with the mode (i.e. the most frequently appeared) of the available value in each numerical feature column. Although this can keep the mode, maximum, minimum, range of each feature column unchanged, this would bring uncertain change to mean, median, standard deviation, variance as this highly depend on the distribution of the features. Talking about categorical features, mode imputation fills in the missing values with the most frequently appeared

available value in each categorical feature column. This can keep the most frequently appeared values and the number of categories of each feature column unchanged. No matter numerical features or categorical features, since mode imputation assumes the missing values are most likely be that most frequently appeared value or the missing values are missing in fully random manner, so if the if the missing values are not behaving in such manner or the mode is biased or skewed or the missing values are not missing in fully random manner, mode imputation can make the feature column biased and worsen the model performance.

For constant imputation, this imputation works on both numerical features and categorical features. Therefore, I will first discuss its impact on numerical features, and follow by that on categorical features. Talking about numerical features, constant imputation fills in the missing values with a constant number (such as 0 or other special values) decided by the user who clean up the data. The impact on the data distribution will be uncertain if an inappropriate constant is chosen. This may greatly affect the mean, median, mode, minimum, maximum, range, standard deviation, variance as we cannot tell what constant will be used. Talking about categorical features, constant imputation fills in the missing values with a constant value (or we call strings if more precisely speaking) decided by the user who clean up the data. The impact on the data distribution will be uncertain if an inappropriate constant is chosen. No matter numerical features or categorical features, since constant imputation assumes the missing values are not useful and not informative, and pretend replacing the missing value with a constant would not affect the distribution greatly, so if the constant is inconsistent that it is far outside the range/domain of the existing available values in the feature column, or missing values are not missing in fully random manner, constant imputation can make the feature column biased and worsen the model performance.

Judge what imputation should be used and when:

For mean imputation, this should be used when the missing values are missing in fully random manner, or the missing values are very close to the mean of the distribution of that feature column. In addition, this strategy is used when users want to simply fill in those missing values in a simple and efficient way, which explains why this is one of the most preferred imputation strategies to use for numerical features. Moreover, this strategy should only be used for numerical features but not categorical features. Therefore, by taking the training dataset as an example, those numerical feature columns that contain missing value are suitable to use mean

imputation.

For median imputation, this should be used when the missing values are missing in fully random manner, or the missing values are very close to the median of the distribution of that feature column. In addition, this strategy is used when users want to simply fill in those missing values in a simple and efficient way, which explains why this is one of the most preferred imputation strategies to use for numerical features. Moreover, this strategy should only be used for numerical features but not categorical features. Therefore, by taking the training dataset as an example, those numerical feature columns that contain missing value are suitable to use median imputation.

For mode imputation, this should be used when the missing values are very close to, or even most likely to be the most frequently appeared values in the feature column. In addition, this is a simple and efficient way to fill in the missing values. This is highly preferred to use for categorical features as the imputation strategies available for categorical features are fewer than that for numerical features. This is one of the best imputation strategies for categorical features. However, this imputation strategy is less frequently used for numerical features as fill in those missing values with the mode would be more likely to introduce more biases than that of mean imputation and median imputation, not even talking about the uncertain effect on the distribution of the data is of higher degree. Therefore, by taking the training dataset as an example, those categorical feature columns that contain missing value are suitable to use mode imputation.

For constant imputation, this is the least preferred imputation strategy among these four mentioned strategies. This strategy is rarely used because constant imputation is used when the missing values are not useful and not informative and pretend replacing the missing value with a constant would not affect the distribution greatly. However, this situation rarely happens as most features should follow certain distribution. Therefore, this strategy is usually not used for both numerical features and categorical features even though it is simple to use. Therefore, by taking the training dataset as an example, constant imputation should not be the first consideration to fill in the missing values, no matter it is of numerical or categorical features.

[Q8] Normalization and Standardization

Report the first numerical feature column of the first 10 samples before and after processing:

Before any normalization and standardization process:

```
0    65.00
1    65.00
2    59.50
3    66.55
4    71.00
5    70.00
6    57.50
7    65.50
8    70.00
9    80.00
Name: C6, dtype: float64
```

After using StandardScaler:

```
0   -0.200135
1   -0.200135
2   -1.031074
3    0.034039
4    0.706344
5    0.555264
6   -1.333234
7   -0.124595
8    0.555264
9    2.066063
Name: C6, dtype: float64
```

After using MinMaxScaler:

```
0    0.368421
1    0.368421
2    0.252632
3    0.401053
4    0.494737
5    0.473684
6    0.210526
7    0.378947
8    0.473684
9    0.684211
Name: C6, dtype: float64
```

After using RobustScaler:

```
0   -0.206667
1   -0.206667
2   -0.940000
3    0.000000
4    0.593333
5    0.460000
6   -1.206667
7   -0.140000
8    0.460000
9    1.793333
Name: C6, dtype: float64
```

Discuss the difference between these techniques and when to use each:

For StandardScaler, it scales the numerical features such that the mean of the feature column is 0 and the standard deviation of the feature column is 1. In short, this technique scales the numerical features to the Z-score. Below is the formula used in this technique:

$$x_i' = \frac{x_i - \mu}{\sigma}$$

$where\ x_i'\ is\ the\ value\ after\ using\ StandardScaler,$
$x_i\ is\ the\ initial\ value,$
$\mu\ is\ the\ mean,$
$\sigma\ is\ the\ standard\ deviation$

StandardScaler should be used when the numerical data follows a normal distribution, or if we would like to use some algorithms that have an assumption that the data is following a normal distribution later.

For MinMaxScaler, it scales the numerical features to a given range. By default, people in general specify the range is between 0 and 1 (both inclusively). The minimum value will be scaled to 0 and the maximum value will be scaled to 1 for each numerical feature column if we follow the default setting. Otherwise, the minimum value will be scaled to the lower bound value defined by the user and the maximum value will be scaled to the upper bound value defined by the user. This

technique can preserve the "relative distance" between the values of each feature column. Below is the formula used in this technique:

$$x_i' = \frac{x_i - x_{min}}{x_{max} - x_{min}} \times (c_{max} - c_{min}) + c_{min}$$

$where\ x_i'\ is\ the\ value\ after\ using\ MinMaxScaler,$
$x_i\ is\ the\ initial\ value,$
$x_{max}\ is\ the\ maximum\ value\ in\ the\ feature\ column,$
$x_{min}\ is\ the\ minimum\ value\ in\ the\ feature\ column,$
$c_{max}\ is\ the\ upper\ bound\ of\ the\ feature\ range\ defined\ by\ user\ (by\ default\ this\ is\ 1),$
$c_{min}\ is\ the\ lower\ bound\ of\ the\ feature\ range\ defined\ by\ user\ (by\ default\ this\ is\ 0)$

MinMaxScaler should be used when the user would like to preserve the relativity between the numerical data points and linearly scale them to a user-defined range, or we would like to use some algorithms that are sensitive to the scaling of the numerical features later.

For RobustScaler, it scales the numerical features using median and quantiles. By default, people in general specify the upper limit of the quantile as the third quartile (i.e. 75th quantile) and the lower limit of the quantile as the first quartile (i.e. 25th quantile). Otherwise, the user's choices of the two quantiles should follow the following rule:

0th quantile<lower limit of the quantile<upper limit of the quantile<100th quantile

This technique is intentionally designed to resist the adverse effect from the extreme values and outliers in the data. Below is the formula used in this technique:

$$x_i' = \frac{x_i - x_{median}}{x_{upper} - x_{lower}}$$

$where\ x_i'\ is\ the\ value\ after\ using\ RobustScaler,$
$x_i\ is\ the\ initial\ value,$
$x_{median}\ is\ the\ median\ of\ the\ feature\ column,,$
$x_{upper}\ is\ the\ upper\ limit\ of\ the\ quantile\ defined\ by\ user\ (by\ default\ this\ is\ 75th\ quantile),$
$x_{lower}\ is\ the\ lower\ limit\ of\ the\ quantile\ defined\ by\ user\ (by\ default\ this\ is\ 25th\ quantile)$

RobustScaler should be used when the data contains reasonable number of outliers

and we do not want the outliers to affect the scaling process that much, or we would like to resist against the outliers while preserving the relativity between the data points.

[Q9] Encoding Categorical Variables

Report the first categorical feature column of the first 10 samples before and after processing:

Before any encoding process:

```
0    divorced
1      single
2      single
3     married
4      single
5      single
6      single
7      single
8      single
9      single
Name: C0, dtype: object
```

After using OneHotEncoder:

	C0_divorced	C0_facto union	C0_legally separated	C0_married	C0_single	C0_widower
0	1	0	0	0	0	0
1	0	0	0	0	1	0
2	0	0	0	0	1	0
3	0	0	0	1	0	0
4	0	0	0	0	1	0
5	0	0	0	0	1	0
6	0	0	0	0	1	0
7	0	0	0	0	1	0
8	0	0	0	0	1	0
9	0	0	0	0	1	0

After using OrdinalEncoder:

```
0    0
1    4
2    4
3    3
4    4
5    4
6    4
7    4
8    4
9    4
Name: C0, dtype: int64
```

Explain the scenarios where each encoding technique is preferred:

Since OneHotEncoder transform the input categorical feature into a binary matrix such that for each row of training data instance, there is only one of the choices (i.e. only one of the column) of each feature is 1, while the rest of them is 0, so one-hot encoding is preferred to use when we are handling the nominal data that has no obvious order, rank or priority among the choices for that feature.

Since OrdinalEncoder transform the input categorical feature such that for every choice of each feature, every choice is assigned a unique integer. If there are n categories for one feature, then the possible value of a choice (of each row of training data instance) for that feature is between 0 and (n-1). Hence, ordinal encoding is preferred to use when we are handling the ordinal data that has obvious order, rank or priority among the choices for that feature.

[Q10] Feature Selection

How feature selection impacts the performance of your models:

There are a few advantages can be brough by feature selection if we select the most informative and related features to build our machine learning models.

First of all, this can lower the chance of overfitting. It is because by removing those less relevant features and the features that contain many noises (or outliers), this allows our models to learn the most relevant implicit patterns among those useful features and less disturbed by the noise. Thus, this allows our models generalize better when dealing with the unseen testing dataset.

Secondly, this facilitates the dimensionality reduction. Notice that sometimes we may be dealing with a lot of features when building our models, maybe hundreds, thousands or more. If we simply keep all the features and use them to build our models, this makes the building process more complicated and require more efforts. Hence, selecting the most relevant features to build our models can make the building process easier and reduce the efforts needed.

Thirdly, this helps improve the model performance by raising the model accuracy. It is

obvious to see that when we exclude the features that are less relevant and more noisy, the model can learn the most critical patterns and can generalize more better. Therefore, this improves the model accuracy.

Fourthly, this boosts the training speed. Since the number of features required to train the model has reduced, fewer computational resources are needed to train the models. This results in shorter training time and encourages building more models and faster prediction.

Report which features were removed for what reasons:

For using VarianceThreshold as the feature selection method, feature columns C0, C1, C3, C4, C7, C8, C9, C10, C11, C12, C13, C15 are removed. They are removed because these feature columns have a variance less than 0.5. Take the feature column C15 as an example, it is a binary nominal categorical feature. As I have applied one hot encoding on this feature column before using the feature selection method, so most of the values under the columns of C15_no and C15_yes are mostly filled with either 0s or 1s. This results in a very small variance under these feature columns.

For using SelectKBest as the feature selection method with the target being regression target, feature columns C0, partial of C1, C3, C4, C7, C8, C9, C10, C11, C12, C13, C15, C16, C17, C18, C19, C20, C21, C22, C24, C25, C27, C28, C29, C30 are removed. They are removed because these features are not within the top 9 highest scores with the regression target (Note that for the nominal categorical feature columns, if they are processed using one hot encoding, each choice of that feature is counted as 1 feature column; if there are N choices for that feature, it would count as having N feature columns but not 1). The scoring function I have used is mutual_info_regression, which measures the dependency between each feature and the regression target. If the dependency is higher, the greater the values. The lowest extreme case is 0, which implies the feature and the regression target is completely independent. Hence, these features are removed as the regression target are less dependent on these features.

For using SelectKBest as the feature selection method with the target being classification target, feature columns C0, C1, C2, C3, C4, C5, C6, C7, C8, C9, C10, C12, C13, C15, C16, C17, C21, C22, C23, C27, C28, C29, C30 are removed. They are removed because these features are not within the top 9 highest scores with the

classification target (Note that for the nominal categorical feature columns, if they are processed using one hot encoding, each choice of that feature is counted as 1 feature column; if there are N choices for that feature, it would count as having N feature columns but not 1). The scoring function I have used is mutual_info_classif, which measures the dependency between each feature and the classification target. If the dependency is higher, the greater the values. The lowest extreme case is 0, which implies the feature and the classification target is completely independent. Hence, these features are removed as the classification target are less dependent on these features.

[Q11] Feature Engineering

Suggest one feature engineering method that could help improve model performance:

I would suggest feature creation using data-driven approach to help improving the model performance. In particular, I would create a new feature column by multiplying the feature column C20 with the feature column C26 together for each row of dataset.

Justify the creation of this new feature based on your data exploration findings:

The reason for suggesting using the technique of feature creation using the data-driven approach is because I notice that when both feature column C20 and feature C26 are greater than 0, the regression target is tends to be bigger than other rows of dataset that when one of the columns or both of the columns are 0. This gives me an impression that if the newly created feature columns have a bigger value, the greater the regression target.

Part 3: Regression

Linear Regression

Model setting for the seven models I have built:

Model name	Feature(s) used to train the model
Model 1	C8 (binary categorical feature)
Model 2	C9 (binary categorical feature)
Model 3	C10 (binary categorical feature)
Model 4	C28 (continuous numerical feature)
Model 5	C29 (continuous numerical feature)
Model 6	C30 (continuous numerical feature)
Model 7	C8, C9, C10, C28, C29, C30

[Q12]

Report the validation R^2 score of each of the models to evaluate the relationship between different features and the regression target:

Model name	R^2 validation score
Model 1	-0.0017531141065381117
Model 2	-0.00032257500517074433
Model 3	-0.006686626115260941
Model 4	0.0025504461422434233
Model 5	0.0006419687473380176
Model 6	$3.8542612966985246\times10^{-5}$
Model 7	0.0015940979258570742

[Q13]

Report the mean squared error of each of the seven models in the validation set:

Model name	Mean squared error (MSE)

Model 1	0.004872941755801945
Model 2	0.004865983021536789
Model 3	0.004896940400110126
Model 4	0.004852007457580571
Model 5	0.004861291081516127
Model 6	0.0048642263959816495
Model 7	0.004856659531121537

Compare them based on the two performance metrics, i.e., R^2 score and mean squared error:

Based on the R^2 score, I notice that the models that are trained with continuous numerical data usually gives a better prediction on the regression target compared to the binary categorical data, while the models that are trained with the combination of both continuous numerical data and binary categorical data gives the best prediction on regression target. It is because the R^2 score of Model 1, Model 2, Model 3 (each of them are trained with one binary categorical feature) are negative, while that for Model 4, Model 5, Model 6 (each of them are trained with one continuous numerical feature) are positive. The model with highest R^2 score is Model 7 (which uses the six features in the first six models to train). But still, the R^2 score for all the models are very close to 0, which implies none of them can have a good prediction on regression target based on these features.

Based on the mean squared error (MSE), I can see that all seven models are having similar MSE at around 0.0048, which implies the average square difference between the predicted regression value and the actual regression value are very small. In general, the models that are trained with one continuous numerical feature (Model 4, Model 5, Model 6) have slightly smaller MSE than those trained with one binary categorical feature (Model 1, Model 2, Model 3), while Model 7 (which takes six features in previous six models to train) has its MSE similar to Model 4.

Hence, this shows that there is an implicit implication that if a model has a smaller R^2 score than another model, then this model should usually have a smaller MSE than another model.

In a nutshell, training the model with continuous numerical features tends to give better prediction performance than that of binary categorical features, while the best practice is to use the mix of both two kinds of features (and possibly also include

categorical features with more than 2 possible values) due to the introduction of variety.

[Q14]

Discuss the mathematical meaning of the weight (in brief) for a binary categorical independent variable:

The mathematical meaning of the weight for a binary categorical independent variable means the relative importance of each choice of that binary categorical independent variable to predict the regression target. In other words, this is a way of quantifying the relative contribution of each choice of that binary categorical independent variable to the regression target.

How can a categorical feature with more than 2 possible values be formulated as the independent variable of a linear model:

If a categorical feature with more than 2 possible values, if it is preprocessed with one hot encoding, we can just simply treat each possible value (i.e. each choice) as an independent variable and train the linear regression model with the regression target.

If a categorical feature with more than 2 possible values, if it is preprocessed with ordinal encoding, we can just simply treat the only one feature column as independent variable and train the linear regression model with the regression target, just like what we have done with the continuous numerical feature.

Pick a categorical feature with more than 2 possible values for the linear regression model and repeat [Q12-13]:

Model name	Feature(s) used to train the model
Model 8	C0 (nominal categorical feature with 6 possible values)

Model name	R^2 validation score
Model 8	0.0027947565607518987

Model name	Mean squared error (MSE)
Model 8	0.004850819030588966

Feedforward Neural Networks

[Q15]

Report the model setting, training time, and performance of the neural network model for each value of H:

Model setting

Model name	Trial	Model setting
Model i (H = 1)	Trial 1	hidden_layer_sizes = (1, 1, 1), activation = "identity", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (1, 1, 1), activation = "identity", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (1, 1, 1), activation = "identity", random_state = 3, early_stopping = True, the rest of the parameters set as default
Model ii (H = 8)	Trial 1	hidden_layer_sizes = (8, 8, 8), activation = "identity", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (8, 8, 8), activation = "identity", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (8, 8, 8), activation = "identity",

		random_state = 3, early_stopping = True, the rest of the parameters set as default
Model iii (H = 32)	Trial 1	hidden_layer_sizes = (32, 32, 32), activation = "identity", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (32, 32, 32), activation = "identity", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (32, 32, 32), activation = "identity", random_state = 3, early_stopping = True, the rest of the parameters set as default
Model iv (H = 128)	Trial 1	hidden_layer_sizes = (128, 128, 128), activation = "identity", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (128, 128, 128), activation = "identity", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (128, 128, 128), activation = "identity", random_state = 3, early_stopping = True, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model i (H = 1)	Trial 1	1.781468391418457
	Trial 2	2.029357671737671

	Trial 3	1.5840437412261963
Model ii (H = 8)	Trial 1	0.33663296699523926
	Trial 2	0.5345098972320557
	Trial 3	0.5887799263000488
Model iii (H = 32)	Trial 1	0.42204880714416504
	Trial 2	0.5993216037750244
	Trial 3	0.5758984088897705
Model iv (H = 128)	Trial 1	0.8565599918365479
	Trial 2	1.3544895648956299
	Trial 3	1.0893800258636475

Performance (i.e. R^2 score)

Model name	Trial	Performance (i.e. R^2 score)
Model i (H = 1)	Trial 1	-0.003832471316783659
	Trial 2	-0.003660080008091482
	Trial 3	-0.0072860613504115435
Model ii (H = 8)	Trial 1	-0.021321656740742156
	Trial 2	-0.005825527775841444
	Trial 3	-0.0026648236423454374
Model iii (H = 32)	Trial 1	-0.012956882603078146
	Trial 2	-0.0076944434512895565
	Trial 3	-0.01283064110977361
Model iv (H = 128)	Trial 1	-0.024201294127325967
	Trial 2	-0.015612955972238218
	Trial 3	-0.02241695704077462

Report the mean and standard deviation of the training time and R^2 score for each setting:

Mean and standard deviation of the training time

Model name	Statistics	Training time (seconds)
Model i (H = 1)	mean	1.7982899347941081
	standard deviation	0.18218735305183104
Model ii (H = 8)	mean	0.48664093017578125
	standard deviation	0.10836081941655645
Model iii (H = 32)	mean	0.53242293993632
	standard deviation	0.0786299281735792

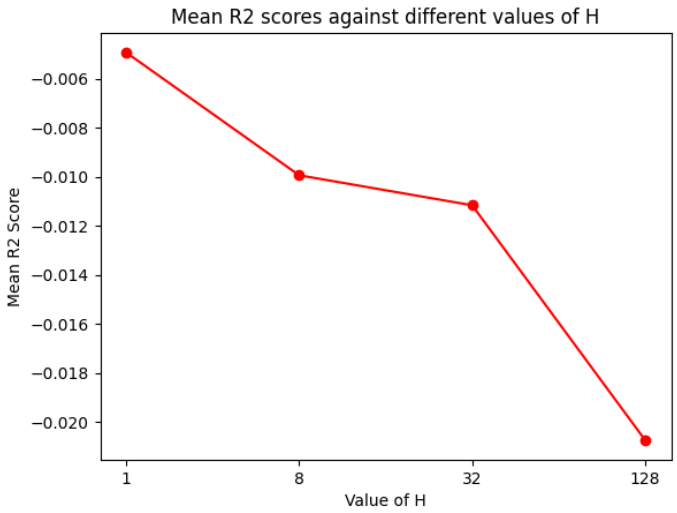
Model iv (H = 128)	mean	1.1001431941986084
	standard deviation	0.20342131857494675

Mean and standard deviation of R^2 score

Model name	Statistics	R^2 score
Model i (H = 1)	mean	-0.004926204225095561
	standard deviation	0.0016701544698533343
Model ii (H = 8)	mean	-0.009937336052976345
	standard deviation	0.008152692021506114
Model iii (H = 32)	mean	-0.011160655721380438
	standard deviation	0.002451523996028372
Model iv (H = 128)	mean	-0.02074373571344627
	standard deviation	0.0037004180191075576

[Q16]

Report the R^2 score for each value of H by plotting them using matplotlib:



Compare the training time and R^2 score of the linear regression model and the best neural network model:

Note that in Trial 2 of the training of Model i (H = 1), it shows a warning that the optimization has not converged yet. There are some reasons that lead to that. The first reason is that the maximum number of iteration allowed to train the neural network is too small, which means more number of iterations should be allowed to

converge. The second reason is that random state to generates the weight and initializes the bias is not a good seed, which explains why we should try different seeds (i.e. repeat 3 times) as the random state even the architecture of the neural network is the same. The third reason is that the number of artificial neurons in each hidden layer is too few, given that there are only three hidden layers, we should either increase the number of artificial neurons or increase the number of layers to improve the convergence issue.

Overall speaking, the training time is in increasing trend when the value of H increases. The mean training time first decreases when H is increasing from 1 to 8, but then it increases when H is increasing from 8 to 128. The lowest mean training time is when H is equal to 8, while the highest mean training time is when H is equal to 1.

Overall speaking, the R^2 score is in decreasing trend when the value of H increases. The mean R^2 score keeps decreasing when H is increasing from 1 to 128. The lowest mean R^2 score is when H is equal to 128, while the highest mean R^2 score is when H is equal to 1.

By taking the training time and the R^2 score into account, if we treat the training time as a more important factor, the best neural network model would be when H is equal to 8, otherwise, if we treat the R^2 score as a more important factor, the best neural network model would be when H is equal to 1. Since we would usually like to have a model that perform as good as it can, so I think the neural network with H is equal to 1 is the best neural network model.

Part 4: Classification

Logistic Regression

Features selected to build the logistic regression model (stochastic gradient descent based): C11, C18, C19, C20, C23, C24, C25, C26

Reason for not selecting other features as selected in Q10:
It is because these features are selected using SelectKBest with the target being the classification target, which matches what we are doing in this part. I do not select other features as selected in Q10 because firstly, VarianceThreshold does not consider the target labels, which makes it cannot filter out all the less related features that can predict the classification target. Secondly another set of selected feature using SelectKBest is being pair with the target being the regression label, which means this set of selected features may be inconsistent to use in this part and result in poor performance.

[Q17]

Report the model setting, training time, and performance of the logistic regression model:

Model setting

Model name	Trial	Model setting
Model 1 (η = 0.25)	Trial 1	loss = "log_loss", random_state = 1, learning_rate = "constant", eta0 = 0.25, the rest of the parameters set as default
	Trial 2	loss = "log_loss", random_state = 2, learning_rate = "constant", eta0 = 0.25, the rest of the parameters set as default
	Trial 3	loss = "log_loss",

		random_state = 3, learning_rate = "constant", eta0 = 0.25, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model 1 (η = 0.25)	Trial 1	0.007850408554077148
	Trial 2	0.009112119674682617
	Trial 3	0.009345293045043945

Validation accuracy

Model name	Trial	Validation accuracy
Model 1 (η = 0.25)	Trial 1	0.8700564971751412
	Trial 2	0.8757062146892656
	Trial 3	0.8742937853107344

F1 score

Model name	Trial	F1 score
Model 1 (η = 0.25)	Trial 1	0.9127134724857684
	Trial 2	0.9163498098859316
	Trial 3	0.9138431752178122

Report the corresponding mean and standard deviation of the training time, accuracy, and the F1 score for each setting:

Mean and standard deviation of the training time

Model name	Statistics	Training time (seconds)
Model 1 (η = 0.25)	mean	0.00876927375793457
	standard deviation	0.0006566721162073562

Mean and standard deviation of the validation accuracy

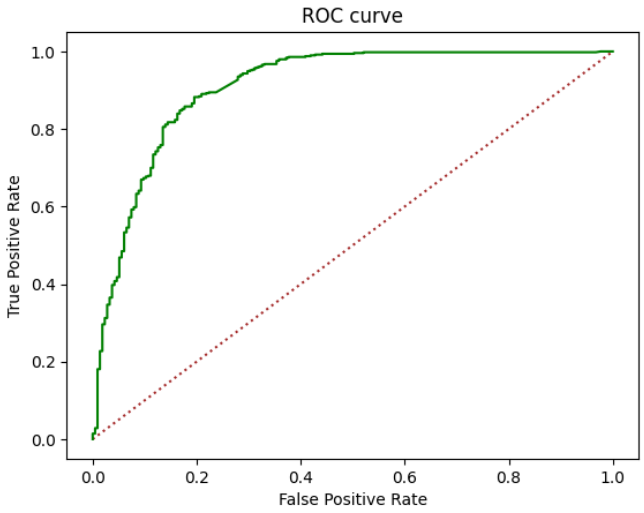
Model name	Statistics	Validation accuracy
Model 1 (η = 0.25)	mean	0.873352165725047
	standard deviation	0.0024006683209005755

Mean and standard deviation of the F1 score

Model name	Statistics	F1 score
Model 1 (η = 0.25)	mean	0.9143021525298374
	standard deviation	0.0015195904166741476

[Q18]

Plot the ROC curve calculated on the validation set with the last model in [Q17] and report the AUC value:



AUC value = 0.908184348318317

Give one reason why we need to examine the ROC curve as well:

It is because ROC curve (full name: Receiver Operating Characteristic curve) can provide us the performance measurement on classification problem (usually binary classification problem, but also applicable to multi-class classification problem), which tells us about the trade-off between true positive rate and false positive rate at different threshold settings. Since ROC curve is a probability curve, with y-axis being true positive rate and x-axis being false positive rate, we can understand how well the classification model in classifying multiple classes.

[Q19]

Report the accuracy and F1 score on the validation set using different learning rates:

Model setting

Model name	Trial	Model setting
Model 2 (η = 0.50)	Trial 1	loss = "log_loss", random_state = 1, learning_rate = "constant", eta0 = 0.50, the rest of the parameters set as default
	Trial 2	loss = "log_loss", random_state = 2, learning_rate = "constant", eta0 = 0.50, the rest of the parameters set as default
	Trial 3	loss = "log_loss", random_state = 3, learning_rate = "constant", eta0 = 0.50, the rest of the parameters set as default
Model 3 (η = 0.75)	Trial 1	loss = "log_loss", random_state = 1, learning_rate = "constant", eta0 = 0.75, the rest of the parameters set as default
	Trial 2	loss = "log_loss", random_state = 2, learning_rate = "constant", eta0 = 0.75, the rest of the parameters set as default
	Trial 3	loss = "log_loss", random_state = 3, learning_rate = "constant", eta0 = 0.75, the rest of the parameters set as default

Validation accuracy

Model name	Trial	Validation accuracy

Model 2 (η = 0.50)	Trial 1	0.8700564971751412
	Trial 2	0.7754237288135594
	Trial 3	0.8714689265536724
Model 3 (η = 0.75)	Trial 1	0.8531073446327684
	Trial 2	0.8305084745762712
	Trial 3	0.867231638418079

F1 score

Model name	Trial	F1 score
Model 2 (η = 0.50)	Trial 1	0.9125475285171103
	Trial 2	0.822742474916388
	Trial 3	0.9130850047755491
Model 3 (η = 0.75)	Trial 1	0.8955823293172691
	Trial 2	0.8795180722891566
	Trial 3	0.9106463878326996

Mean and standard deviation of the validation accuracy

Model name	Statistics	Validation accuracy
Model 2 (η = 0.50)	mean	0.8389830508474576
	standard deviation	0.044946926496657706
Model 3 (η = 0.75)	mean	0.8502824858757062
	standard deviation	0.015124650098125223

Mean and standard deviation of the F1 score

Model name	Statistics	F1 score
Model 2 (η = 0.50)	mean	0.8827916694030158
	standard deviation	0.042461759572990976
Model 3 (η = 0.75)	mean	0.8952489298130417
	standard deviation	0.012710268121133652

Feedforward Neural Networks

[Q20]

Report the model setting, training time, and performance of the neural networks for each value of H:

Model setting

Model name	Trial	Model setting
Model i (H = 1)	Trial 1	hidden_layer_sizes = (1, 1, 1), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (1, 1, 1), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (1, 1, 1), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default
Model ii (H = 8)	Trial 1	hidden_layer_sizes = (8, 8, 8), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (8, 8, 8), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (8, 8, 8), activation = "logistic",

		random_state = 3, early_stopping = True, the rest of the parameters set as default
Model iii (H = 32)	Trial 1	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default
Model iv (H = 128)	Trial 1	hidden_layer_sizes = (128, 128, 128), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (128, 128, 128), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (128, 128, 128), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model i (H = 1)	Trial 1	0.3348367214202881
	Trial 2	0.28469228744506836

	Trial 3	0.38020801544189453
Model ii (H = 8)	Trial 1	0.36047935485839844
	Trial 2	0.3013737201690674
	Trial 3	0.4588038921356201
Model iii (H = 32)	Trial 1	0.6956384181976318
	Trial 2	3.542210340499878
	Trial 3	1.021789312362671
Model iv (H = 128)	Trial 1	9.560025930404663
	Trial 2	7.503983974456787
	Trial 3	9.087588548660278

Validation accuracy

Model name	Trial	Validation accuracy
Model i (H = 1)	Trial 1	0.6963276836158192
	Trial 2	0.6963276836158192
	Trial 3	0.6963276836158192
Model ii (H = 8)	Trial 1	0.6963276836158192
	Trial 2	0.6963276836158192
	Trial 3	0.809322033898305
Model iii (H = 32)	Trial 1	0.6963276836158192
	Trial 2	0.8714689265536724
	Trial 3	0.6963276836158192
Model iv (H = 128)	Trial 1	0.8785310734463276
	Trial 2	0.867231638418079
	Trial 3	0.8757062146892656

F1 score

Model name	Trial	F1 score
Model i (H = 1)	Trial 1	0.8209825145711906
	Trial 2	0.8209825145711906
	Trial 3	0.8209825145711906
Model ii (H = 8)	Trial 1	0.8209825145711906
	Trial 2	0.8209825145711906
	Trial 3	0.8746518105849582
Model iii (H = 32)	Trial 1	0.8209825145711906
	Trial 2	0.9099901088031652
	Trial 3	0.8209825145711906

Model iv (H = 128)	Trial 1	0.9171483622350676
	Trial 2	0.9076620825147348
	Trial 3	0.9147286821705426

Report the mean and standard deviation of the training time, accuracy, and the F1 score for each setting:

Mean and standard deviation of the training time

Model name	Statistics	Training time (seconds)
Model i (H = 1)	mean	0.3332456747690837
	standard deviation	0.039010358844591114
Model ii (H = 8)	mean	0.3735523223876953
	standard deviation	0.06493197271694498
Model iii (H = 32)	mean	1.7532126903533936
	standard deviation	1.2720005360620976
Model iv (H = 128)	mean	8.717199484507242
	standard deviation	0.8792868986808242

Mean and standard deviation of the validation accuracy

Model name	Statistics	Validation accuracy
Model i (H = 1)	mean	0.6963276836158192
	standard deviation	0.0
Model ii (H = 8)	mean	0.7339924670433144
	standard deviation	0.053266047547009206
Model iii (H = 32)	mean	0.7547080979284368
	standard deviation	0.08256237369786433
Model iv (H = 128)	mean	0.8738229755178907
	standard deviation	0.00480133664180113

Mean and standard deviation of the F1 score

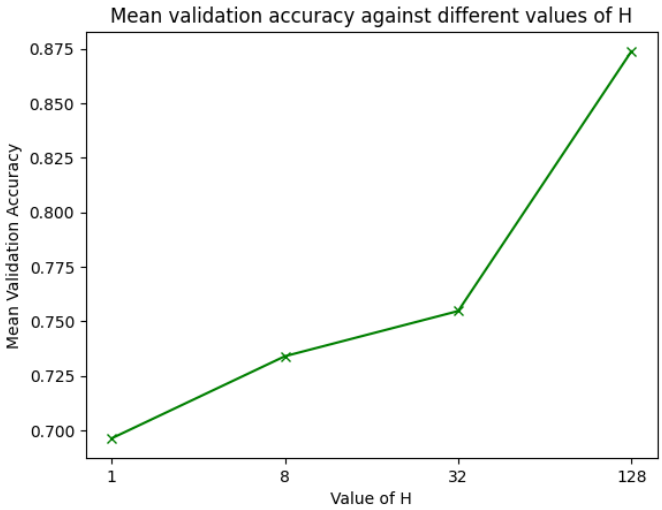
Model name	Statistics	F1 score
Model i (H = 1)	mean	0.8209825145711905
	standard deviation	$1.1102230246251565 \times 10^{-16}$
Model ii (H = 8)	mean	0.8388722799091132
	standard deviation	0.02529994876856213
Model iii (H = 32)	mean	0.8506517126485155
	standard deviation	0.04195858230568657

Model iv (H = 128)	mean	0.9131797089734484
	standard deviation	0.004024662653116539

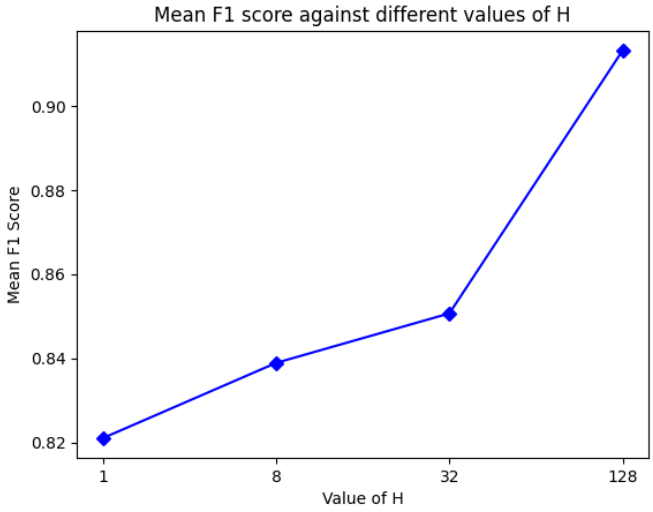
[Q21]

Plot the accuracy and F1 score for each value of H:

Validation accuracy for each value of H



F1 score for each value of H



Suggest a possible reason for the gap between the accuracy and F1 score:

One possible reason that F1 score is higher than accuracy in the above situation is due to uneven distribution of the binary classes in the training dataset. Recall that

accuracy emphasize on number of true positive and true negative instances, while F1 score emphasize on precision and recall (and hence emphasize on false positive and false negative instances).

It is possible that F1 score is higher than accuracy is due to that a huge portion of the classification results are true positive and false positive, while very few of them are true negative or false negative. This can be verified that nearly 70% of the classification target is “success” in the training dataset, which means there are many positive labels in the training dataset. Due to such imbalance distribution between positive labels (“success”) and negative labels (“failure”), it is quite easy for the classification feedforward neural network are more bias to positive label and thus there are more true positive and false positive classification results.

[Q22]

Compare the training time, accuracy, and the F1 score of the logistic regression model and the best neural network model:

Overall speaking, the training time is increasing as the value of H increases. The shortest mean training time is Model i (H = 1), which is around 0.333 seconds, while the longest mean training time is Model iv (H = 128), which is around 8.72 seconds. The mean training time keeps increasing from 0.333 seconds when H = 1 to 8.72 seconds when H = 128, which is around 2520% increase.

Overall speaking, the validation accuracy is increasing as the value of H increases. The lowest mean validation accuracy is Model i (H = 1), which is around 0.696, while the highest mean validation accuracy is Model iv (H = 128), which is around 0.874. The mean validation accuracy keeps increasing from 0.696 when H = 1 to 0.874 when H = 128, which is around 25.5% increase.

Overall speaking, the F1 score is increasing as the value of H increases. The lowest mean F1 score is Model i (H = 1), which is around 0.821, while the highest mean F1 score is Model iv, which is around 0.913. The mean F1 score keeps increasing from 0.821 when H = 1 to 0.913 when H = 128, which is around 11.2% increase.

By taking training time, validation accuracy and F1 score into account, since we mostly value the performance rather than the training time, so the best neural network model would be Model iv with H = 128.

[Q23]

Do you notice any trend when you increase the hidden layer size from 1 to 128? If so, please describe what the trend is and suggest a reason for your observation:

Yes, I notice there is an increasing trend in training time, validation accuracy and F1 score when the hidden layer size increases from 1 to 128. I have already mentioned that in Q22 as shown above.

First of all, there is an increasing trend in training time (and also the mean training time) when the value of H increases from 1 to 128. The reason for this is quite natural because when the hidden layer sizes increases from 1 to 128, the complexity of the neural network model becomes higher. By assuming the computational power being constant, the computational resources to train a more complicated neural network model would be more demanding, thus needing more time to train.

Secondly, there is an increasing trend in validation accuracy (and also the mean validation accuracy) when the value of H increases from 1 to 128. The reason for this is because when the hidden layer size increases from 1 to 128, the neural network becomes more and more complicated, and is more able to learn more characteristics, patterns, relationships between the training features and the training classification labels. Thus, more and more prediction on the validation dataset is true positive and true negative, leading to higher validation accuracy.

Thirdly, there is an increasing trend in F1 score (and also the mean F1 score) when the value of H increases from 1 to 128. The reason for this is quite similar to the explanation for the increasing trend in validation accuracy, because when the hidden layer size increases from 1 to 128, the neural network becomes more and more complicated, and is more able to learn more characteristics, patterns, relationships between the training features and the training classification labels. Thus, fewer and fewer prediction on the validation dataset is false positive and false negative, leading to higher precision and higher recall. This results in higher F1 score.

Part 5: Performance Enhancement

Preprocessing Validation

I find Pipeline and ColumnTransformer too hard to use, so I implement them in the simple way.

[Q24] Combination A

Validate this combination with a neural network model:

Model setting

Model name	Trial	Model setting
Model A	Trial 1	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model A	Trial 1	0.4250612258911133
	Trial 2	0.42458438873291016
	Trial 3	1.1732749938964844

Validation accuracy

Model name	Trial	Validation Accuracy
Model A	Trial 1	0.6963276836158192
	Trial 2	0.6963276836158192
	Trial 3	0.8898305084745762

F1 score

Model name	Trial	F1 score
Model A	Trial 1	0.8209825145711906
	Trial 2	0.8209825145711906
	Trial 3	0.9244186046511628

Mean and standard deviation of training time

Model name	Statistics	Training time (seconds)
Model A	mean	0.6743068695068359
	standard deviation	0.3528237980551686

Mean and standard deviation of validation accuracy

Model name	Statistics	Validation accuracy
Model A	mean	0.7608286252354048
	standard deviation	0.09121810642425328

Mean and standard deviation of F1 score

Model name	Statistics	F1 score
Model A	mean	0.8554612112645147
	standard deviation	0.048760240476647246

[Q25] Combination B

Validate this combination with a neural network model:

Model setting

Model name	Trial	Model setting
Model B	Trial 1	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 1,

		early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 2, early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model B	Trial 1	0.4071986675262451
	Trial 2	0.3943488597869873
	Trial 3	0.46668171882629395

Validation accuracy

Model name	Trial	Validation Accuracy
Model B	Trial 1	0.6963276836158192
	Trial 2	0.6963276836158192
	Trial 3	0.6963276836158192

F1 score

Model name	Trial	F1 score
Model B	Trial 1	0.8209825145711906
	Trial 2	0.8209825145711906
	Trial 3	0.8209825145711906

Mean and standard deviation of training time

Model name	Statistics	Training time (seconds)
Model B	mean	0.4227430820465088
	standard deviation	0.03150907003053131

Mean and standard deviation of validation accuracy

Model name	Statistics	Validation accuracy
Model B	mean	0.6963276836158192
	standard deviation	0.0

Mean and standard deviation of F1 score

Model name	Statistics	F1 score
Model B	mean	0.8209825145711905
	standard deviation	$1.1102230246251565 \times 10^{-16}$

[Q26] Combination C

Validate this combination with a neural network model:

Preprocessing on the features
First, for numerical features, I apply SimpleImputer with median strategy, and RobustScaler on them. Secondly, for ordinal categorical features, I apply SimpleImputer with mode strategy, and OrdinalEncoder on them. Thirdly, for nominal categorical features, I apply SimpleImputer with mode strategy, and OneHotEncoder on them. Fourthly, after combining numerical features, ordinal categorical features and nominal categorical features into a single Pandas DataFrame, I apply SelectKBest with the target being classification target to select the best 12 features for training.

Model setting

Model name	Trial	Model setting
Model C	Trial 1	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 1, early_stopping = True, the rest of the parameters set as default
	Trial 2	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 2,

		early_stopping = True, the rest of the parameters set as default
	Trial 3	hidden_layer_sizes = (32, 32, 32), activation = "logistic", random_state = 3, early_stopping = True, the rest of the parameters set as default

Training time

Model name	Trial	Training time (seconds)
Model C	Trial 1	2.106133222579956
	Trial 2	1.4577064514160156
	Trial 3	0.34084439277648926

Validation accuracy

Model name	Trial	Validation Accuracy
Model C	Trial 1	0.8742937853107344
	Trial 2	0.8700564971751412
	Trial 3	0.6963276836158192

F1 score

Model name	Trial	F1 score
Model C	Trial 1	0.9128305582761997
	Trial 2	0.9099804305283756
	Trial 3	0.8209825145711906

Mean and standard deviation of training time

Model name	Statistics	Training time (seconds)
Model C	mean	1.3015613555908203
	standard deviation	0.7290848745150401

Mean and standard deviation of validation accuracy

Model name	Statistics	Validation accuracy
Model C	mean	0.8135593220338982
	standard deviation	0.08291333401943034

Mean and standard deviation of F1 score

Model name	Statistics	F1 score
Model C	mean	0.8812645011252553
	standard deviation	0.04264167940894649

Hyperparameter Tuning

You are expected to try at least 10 combinations of the hyperparameter setting:

```
parameters_grid = {"hidden_layer_sizes": [(2, 4, 4), (4, 8, 4), (32, 32, 16), (64, 128, 64), (144, 144, 80),
                                          (128, 192, 192), (128, 256, 128), (96, 96, 96), (128, 128, 128), (256, 256, 256)],

                   "activation": ["logistic"],

                   "solver": ["sgd", "adam"],

                   "learning_rate": ["adaptive"],

                   "learning_rate_init": [0.2],

                   "max_iter": [1000],

                   "random_state": [4211],

                   "early_stopping": [True]
                   }
```

As you can see above, there are 10×1×2×1×1×1×1×1=20 possible combinations.

[Q27]

Note that Model 1 is the best model that can achieve the highest validation accuracy.

Report five combinations of the hyperparameter setting and highlight the best hyperparameter setting in terms of accuracy:

Model name	Hyperparameter setting
Model 1 (This is the best model)	hidden_layer_sizes = (4, 8, 4), activation = "logistic", solver = "adam", learning_rate = "adaptive", learning_rate_init = 0.2, max_iter = 1000, random_state = 4211, early_stopping = True, the rest of the parameters set as default
Model 2	hidden_layer_sizes = (64, 128, 64), activation = "logistic", solver = "adam",

	learning_rate = "adaptive", learning_rate_init = 0.2, max_iter = 1000, random_state = 4211, early_stopping = True, the rest of the parameters set as default
Model 3	hidden_layer_sizes = (144, 144, 80), activation = "logistic", solver = "adam", learning_rate = "adaptive", learning_rate_init = 0.2, max_iter = 1000, random_state = 4211, early_stopping = True, the rest of the parameters set as default
Model 4	hidden_layer_sizes = (128, 192, 192), activation = "logistic", solver = "sgd", learning_rate = "adaptive", learning_rate_init = 0.2, max_iter = 1000, random_state = 4211, early_stopping = True, the rest of the parameters set as default
Model 5	hidden_layer_sizes = (256, 256, 256), activation = "logistic", solver = "sgd", learning_rate = "adaptive", learning_rate_init = 0.2, max_iter = 1000, random_state = 4211, early_stopping = True, the rest of the parameters set as default

Report the mean and standard deviation of the validation accuracy of the five random data splits for each hyperparameter setting:

Validation accuracy

Model name	Random data split trial	Validation accuracy
Model 1 (This is the best model)	I	0.8601694915254238
	II	0.8418079096045198
	III	0.8545197740112994
	IV	0.8813559322033898
	V	0.8543140028288543
Model 2	I	0.8502824858757062
	II	0.8347457627118644
	III	0.8347457627118644
	IV	0.865819209039548
	V	0.8444130127298444
Model 3	I	0.6878531073446328
	II	0.6878531073446328
	III	0.6878531073446328
	IV	0.6878531073446328
	V	0.6874115983026874
Model 4	I	0.867231638418079
	II	0.846045197740113
	III	0.847457627118644
	IV	0.867231638418079
	V	0.8557284299858557
Model 5	I	0.8587570621468926
	II	0.8389830508474576
	III	0.8389830508474576
	IV	0.864406779661017
	V	0.8543140028288543

Mean and standard deviation of validation accuracy

Model name	Statistics	Validation accuracy
Model 1 (This is the best model)	mean	0.8584334220346974
	standard deviation	0.012939469299171
Model 2	mean	0.8460012466137655
	standard deviation	0.011549675033780429
Model 3	mean	0.6877648055362438
	standard deviation	0.00017660361677815928
Model 4	mean	0.8567389063361542
	standard deviation	0.009183827774855689

Model 5	mean	0.8510887892663359
	standard deviation	0.010389139377060673