ML_practical_Q2 report

Pouria Yazdani
400243082

A.

Resampling Techniques:

- **Oversampling**: This involves increasing the number of samples in the minority class by duplicating or generating synthetic examples (e.g., using the SMOTE algorithm). This helps balance the dataset by adding more data to the underrepresented class.
- **Undersampling**: In this approach, samples are randomly removed from the majority class to reduce its size, aligning it more closely with the minority class. This helps balance the class distribution, although it risks losing information from the majority class.

Class Weighting:

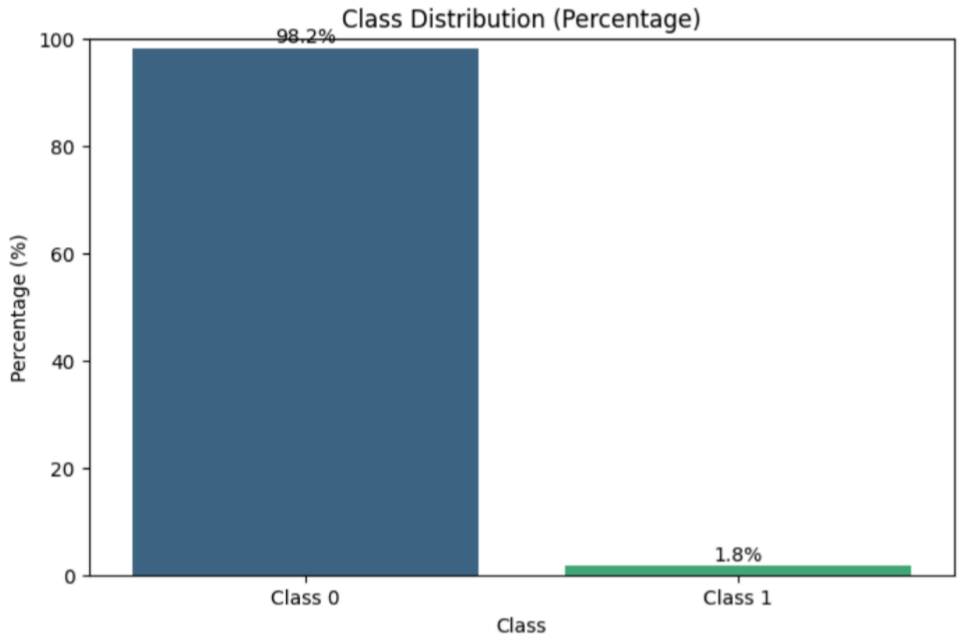
- **Assigning Weights to Classes**: When training a model, you can assign a higher weight to the minority class to penalize the model more heavily when it misclassifies instances of that class. This adjustment encourages the model to focus more on the minority class, improving its sensitivity to less frequent cases. Many algorithms, such as decision trees and support vector machines, allow class weighting directly as a parameter.

Algorithmic Modifications:

- **Specialized Algorithms**: Some machine learning algorithms are specifically designed to handle imbalanced data. For instance, anomaly detection algorithms like One-Class SVM or Isolation Forest are effective when one class is rare. Additionally, ensemble methods such as balanced random forests or EasyEnsemble can enhance performance by incorporating techniques specifically designed for imbalanced data scenarios.

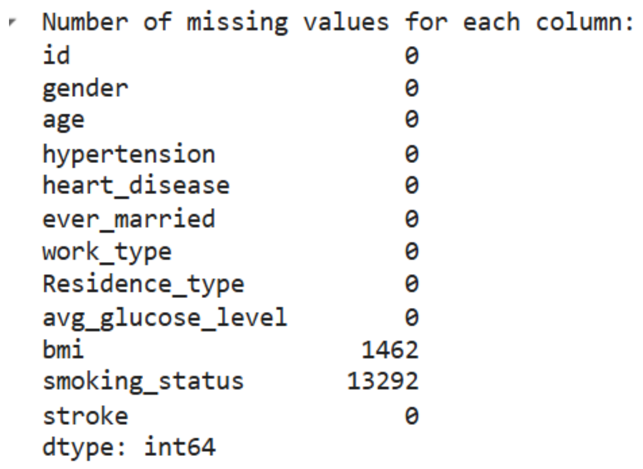
B&C&E.

In this ML problem we are facing several issues. **First** of all our data set is highly imbalanced:



Since we are to perform a **distance-based** algorithm (knn) we should not use oversampling to address this issue. Because in this way the problem will not be solved in correct manner, since each class 1 observation will get duplicated and the distances between the test and train samples could be highly corrupted.

Secondly , two of our predictors contain missing values including, *bmi* and *smoking_status*. One of them is numerical and the other is categorical.

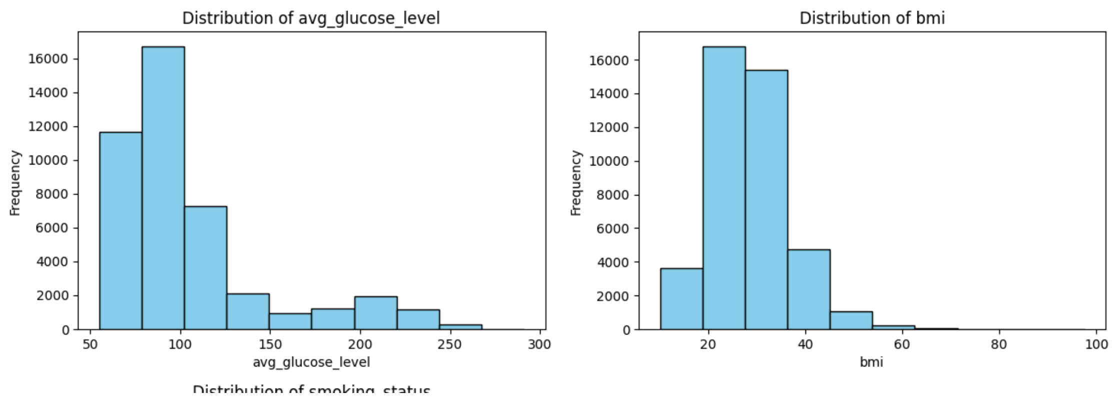


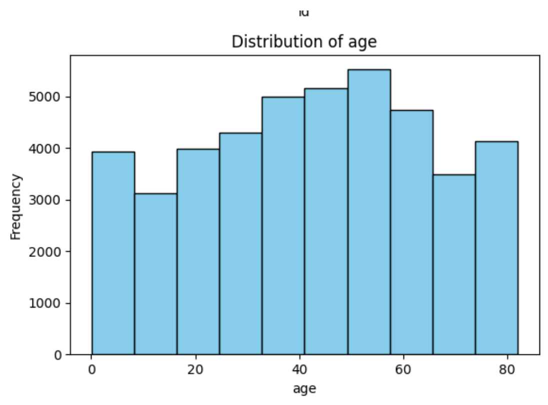
```
Number of missing values for each column:
id                       0
gender                   0
age                      0
hypertension             0
heart_disease            0
ever_married             0
work_type                0
Residence_type           0
avg_glucose_level        0
bmi                   1462
smoking_status       13292
stroke                   0
dtype: int64
```

To solve this problem we have several solutions:

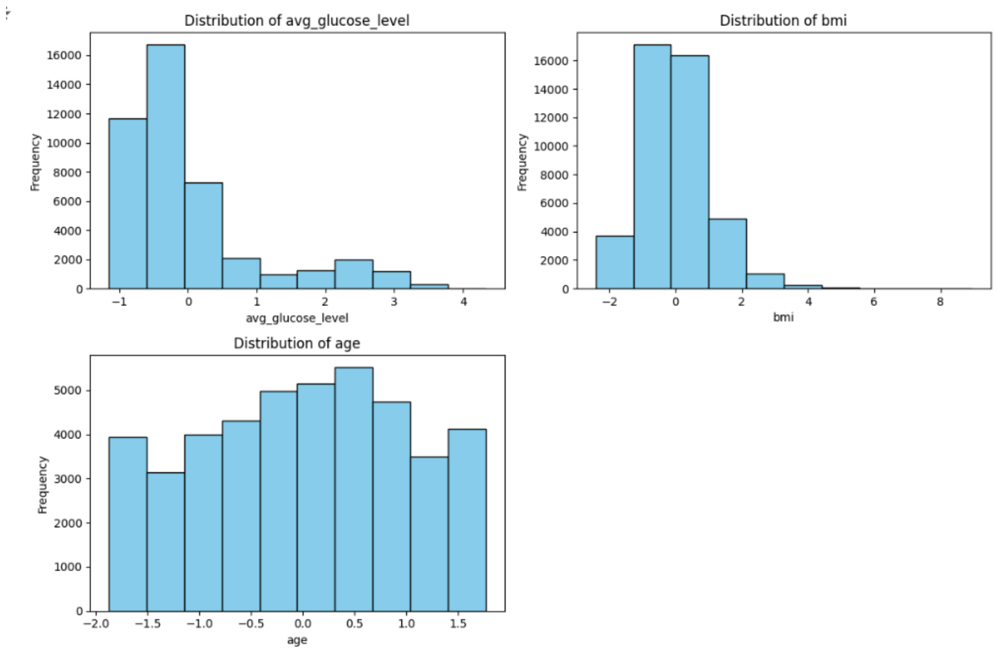
bmi: we can use imputation techniques to address this issue, both median and average filling are desirable. Other techniques such as kmeans clustering are also explored, but the results were **not** better.
Smoking_status: here the problem is more complicated since, median imputation techniques could result in serious corrupted results because we could generate **incorrect meaningful dependencies** which are not inherently in the dataset , two solutions are explored. First we could assign a new column for these missing smoking statuses, and define them as a new predictor, the other way is when performing one-hot encoding on them, just ignore them and all the columns of smoking_status_* will be False (zero).
Both of above mentioned solutions were explored but again **no** meaningful progress were observed.

Further delving into the dataset I observed that the orders of out numerical values were so different which would need **feature scaling**. I have performed z-score normalization in order to address this issue.



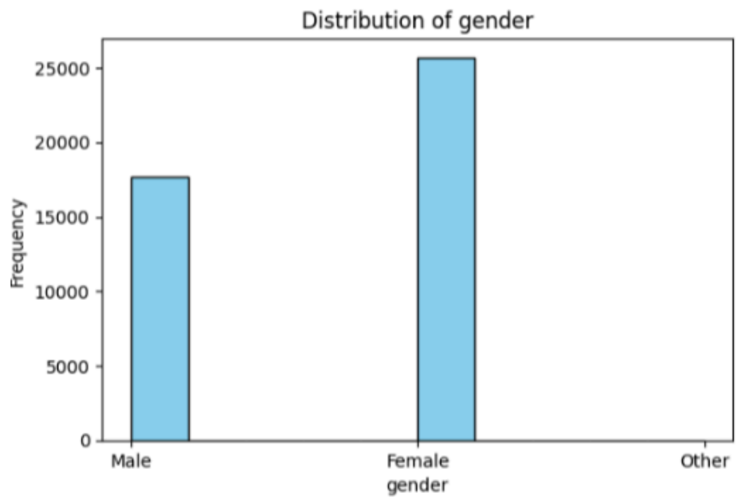


The above distributions are before performing z-score normalization and below is the results after z-score normalization.

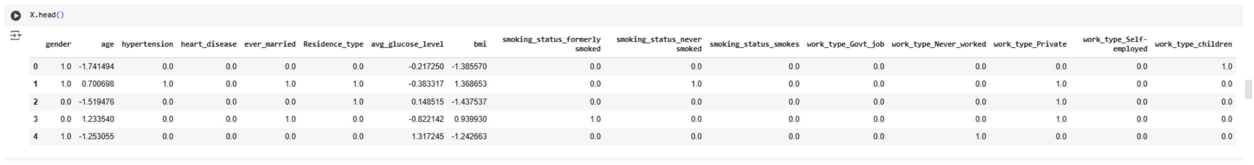


Another thing that I have observed was that when performing one-hot encoding lots of unnecessary columns were added to dataset, especially for 2 class categorical features, so I have presented them as binary values (0 and 1).

Also after **further investigation** I noticed, *gender* predictor has 3 values, and one of them had only 11 observations which had *gender*=”other”. So I simply removed those observations because none of them were of class *stroke=1*.



Further to handle the categorical data, I used one hot encoding using pandas builtin method get_dummies.
So the dataset looks something like this:



```
X.head()
```

	gender	age	hypertension	heart_disease	ever_married	Residence_type	avg_glucose_level	bmi	smoking_status_formerly smoked	smoking_status_never smoked	smoking_status_smokes	work_type_Govt_job	work_type_Never_worked	work_type_Private	work_type_Self-employed	work_type_children
0	1.0	-1.741494	0.0	0.0	0.0	0.0	-0.217250	-1.385570	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1.0
1	1.0	0.700698	1.0	0.0	1.0	1.0	-0.383317	1.368653	0.0	1.0	0.0	0.0	0.0	1.0	0.0	0.0
2	0.0	-1.519476	0.0	0.0	0.0	1.0	0.148515	-1.437537	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
3	0.0	1.233540	0.0	0.0	1.0	0.0	-0.822142	0.939930	1.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0
4	1.0	-1.253055	0.0	0.0	0.0	0.0	1.317245	-1.242663	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0

A subtle and yet very important issue we should consider is the order of performing the above solutions on out dataset in the preprocessing step. We should perform 3 main chores:

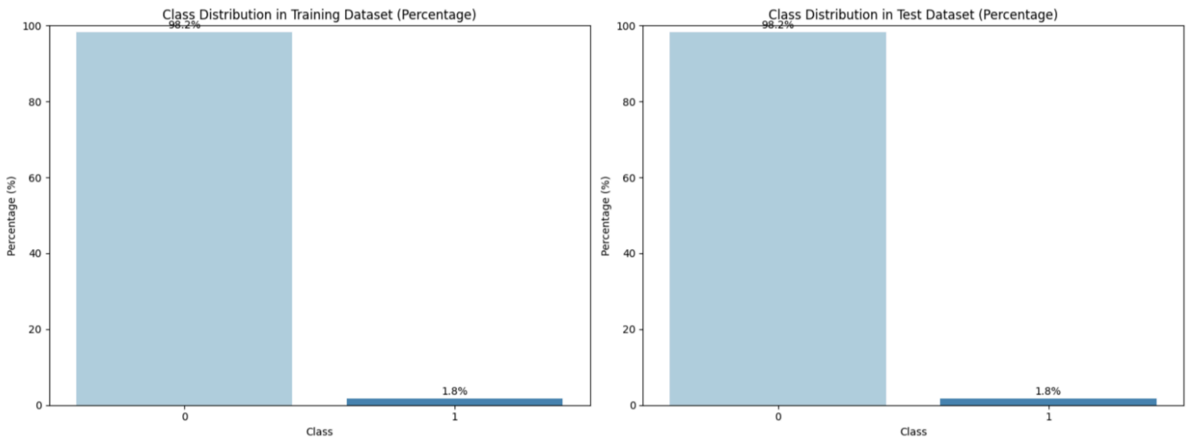
- Undersampling
- Filling nan values and Normalization
- Splitting dataset to train and test.

Based on best practices and mathematically correct practices, splitting the dataset should be the first step in performing any ML flow.
Further on the order of the other 2 chores depends on how do we want to fill the missing values. If we are to use more sophisticated solution like fitting a model (like a random forest) it is **necessary** to first apply the normalization and then perform undersampling(**normalization before undersampling**) but if we are to use typical under sampling methods like random undersampling the order is not that much of importance.
We have used random undersampling with different ratios and have tested the both mentioned orders and the results were not different.

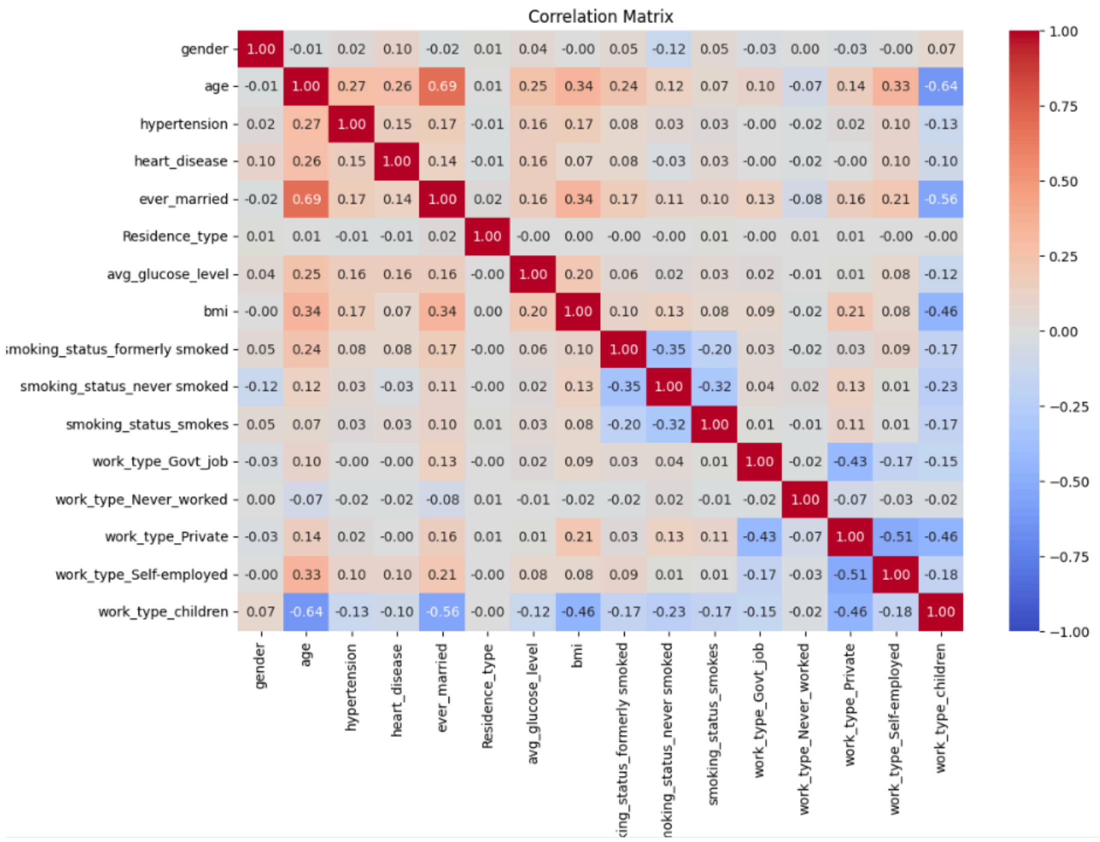
```
Undersample Ratio 2.5:
Class 0 count: 1565
Class 1 count: 626

Undersample Ratio 3.0:
Class 0 count: 1878
Class 1 count: 626

Undersample Ratio 4.0:
Class 0 count: 2504
Class 1 count: 626

Undersample Ratio 5.0:
Class 0 count: 3130
Class 1 count: 626

Undersample Ratio 6.0:
Class 0 count: 3756
Class 1 count: 626
```

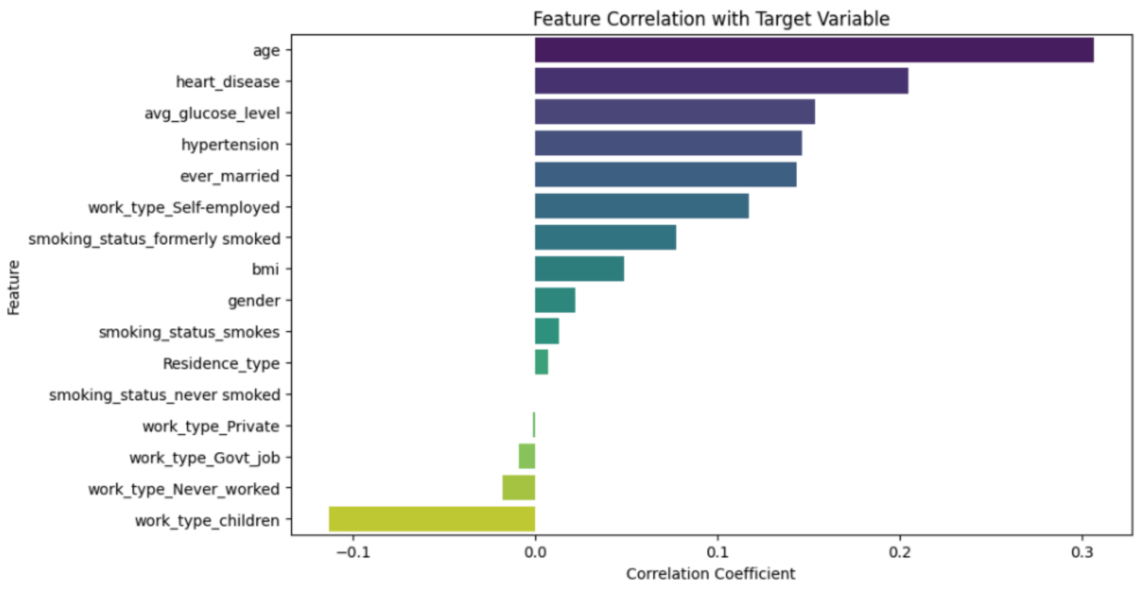
All the tested ratios are present in the notebook.

Another subtle point is that by setting *stratify=y* while splitting the dataset to train and test we are splitting it in a way that the ratio in both test and train are **preserved** and are **identical**.



I have also inspected other feature engineering techniques to further improve my results such as performing **PCA** for dimension reduction and **correlation analysis** for pruning any unhelpful feature which is presented in notebooks.





The results are represented in notebooks

D. is represented in the notebooks

G.

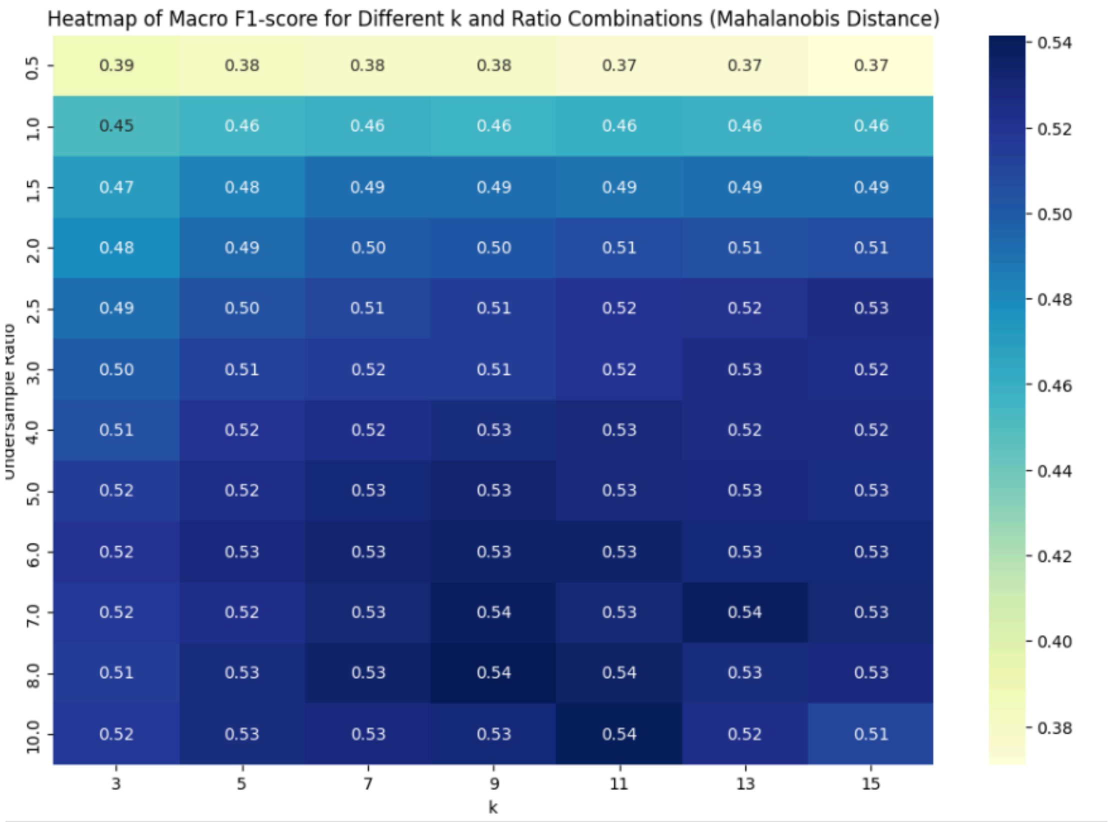
For this part I have used 3 distances:

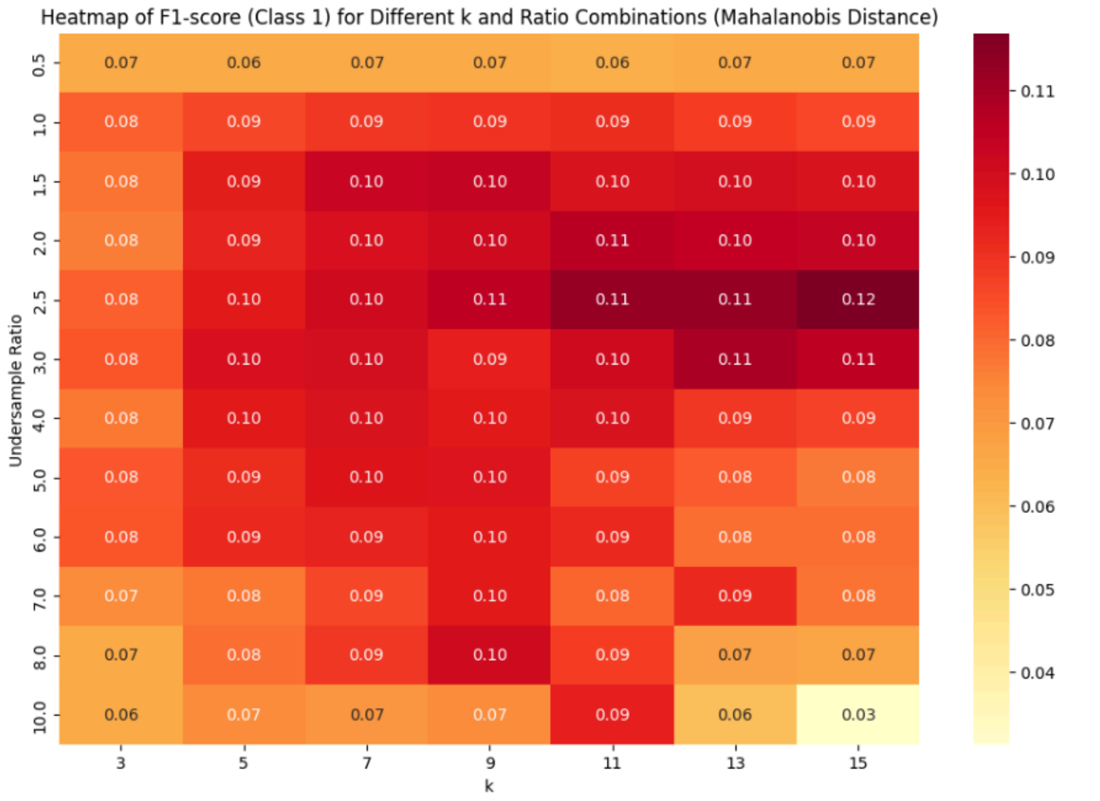
- Euclidean (minkowski where p=2)
- Cosine distance
- Mahalanobis distance

Also I have performed grid search on undersampling ratio (the ratio between stroke=1 and storke2) to extract the best parameters for KNN.

Further I have extracted top 5 models based on **f1 for stroke class** and **macro f1**.(since the data is highly imbalanced)

For example in below the heatmaps and the best models for each category is presented using **Mahalanobis distance. (no significant difference were found using different distance metrics.)**


Heatmap of Macro F1-score for Different k and Ratio Combinations (Mahalanobis Distance)
Undersample Ratio
k
3 5 7 9 11 13 15
0.5: 0.39 0.38 0.38 0.38 0.37 0.37 0.37
1.0: 0.45 0.46 0.46 0.46 0.46 0.46 0.46
1.5: 0.47 0.48 0.49 0.49 0.49 0.49 0.49
2.0: 0.48 0.49 0.50 0.50 0.51 0.51 0.51
2.5: 0.49 0.50 0.51 0.51 0.52 0.52 0.53
3.0: 0.50 0.51 0.52 0.51 0.52 0.53 0.52
4.0: 0.51 0.52 0.52 0.53 0.53 0.52 0.52
5.0: 0.52 0.52 0.53 0.53 0.53 0.53 0.53
6.0: 0.52 0.53 0.53 0.53 0.53 0.53 0.53
7.0: 0.52 0.52 0.53 0.54 0.53 0.54 0.53
8.0: 0.51 0.53 0.53 0.54 0.54 0.53 0.53
10.0: 0.52 0.53 0.53 0.53 0.54 0.52 0.51
0.54 0.52 0.50 0.48 0.46 0.44 0.42 0.40 0.38


Heatmap of F1-score (Class 1) for Different k and Ratio Combinations (Mahalanobis Distance)
Undersample Ratio
k
3 5 7 9 11 13 15
0.5: 0.07 0.06 0.07 0.07 0.06 0.07 0.07
1.0: 0.08 0.09 0.09 0.09 0.09 0.09 0.09
1.5: 0.08 0.09 0.10 0.10 0.10 0.10 0.10
2.0: 0.08 0.09 0.10 0.10 0.11 0.10 0.10
2.5: 0.08 0.10 0.10 0.11 0.11 0.11 0.12
3.0: 0.08 0.10 0.10 0.09 0.10 0.11 0.11
4.0: 0.08 0.10 0.10 0.10 0.10 0.09 0.09
5.0: 0.08 0.09 0.10 0.10 0.09 0.08 0.08
6.0: 0.08 0.09 0.09 0.10 0.09 0.08 0.08
7.0: 0.07 0.08 0.09 0.10 0.08 0.09 0.08
8.0: 0.07 0.08 0.09 0.10 0.09 0.07 0.07
10.0: 0.06 0.07 0.07 0.07 0.09 0.06 0.03
0.11 0.10 0.09 0.08 0.07 0.06 0.05 0.04

```
         Top 5 Models by Macro F1-score:
      k  ratio  f1_class_1  macro_f1  weighted_f1
73    9    8.0    0.100890  0.541543     0.966252
81   11   10.0    0.094017  0.540818     0.971451
68   13    7.0    0.091503  0.537599     0.967554
66    9    7.0    0.095960  0.537426     0.962918
74   11    8.0    0.087542  0.535828     0.967893

      Top 5 Models by F1-score for Class 1:
      k  ratio  f1_class_1  macro_f1  weighted_f1
34   15    2.5    0.116872  0.525198     0.918749
32   11    2.5    0.112719  0.520051     0.912644
33   13    2.5    0.112342  0.521309     0.915478
40   13    3.0    0.109361  0.525160     0.925915
25   11    2.0    0.106033  0.506341     0.892165
```