Machine Learning Project

Death in 5 years prediction

By Stav Rahamim & Tal Rosner

Presentation Summary

- Introduction and approach
- Data preprocessing – main features conclusions
- Unsupervised learning
- Supervised learning
- Models measure methods & accuracy
- Conclusions

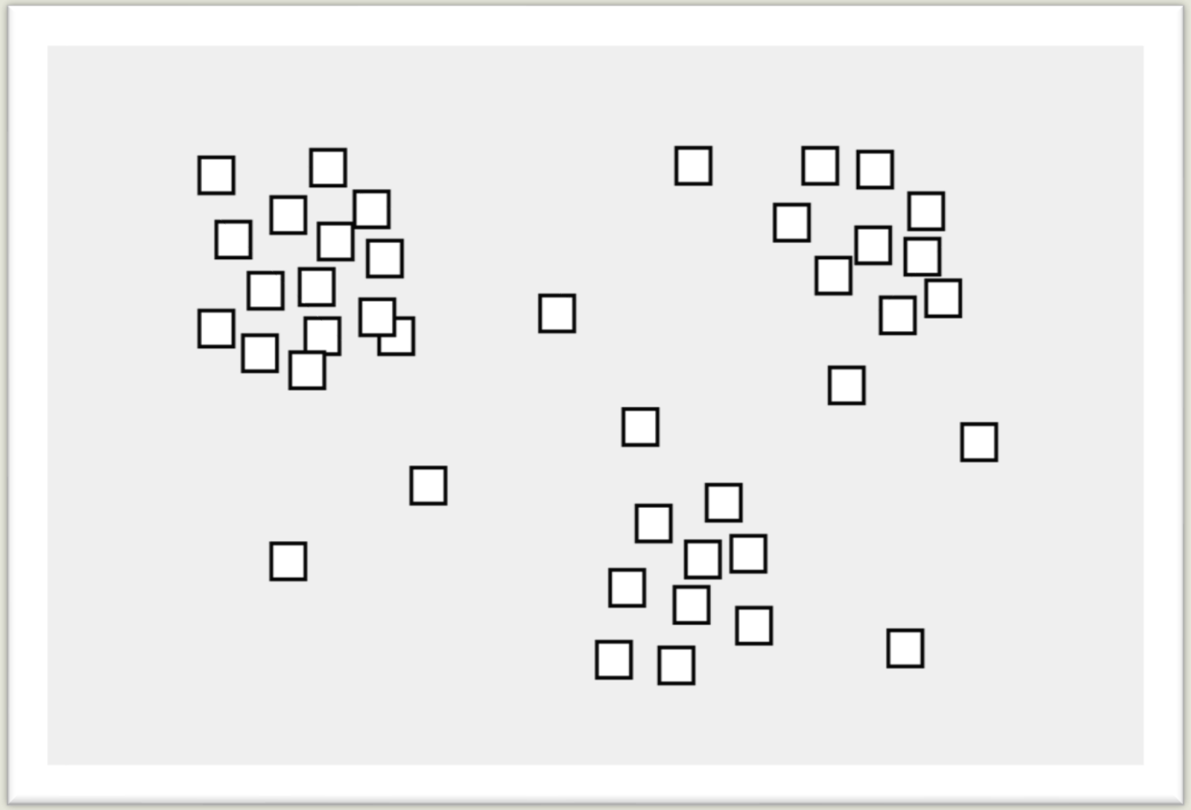
Introduction and approach

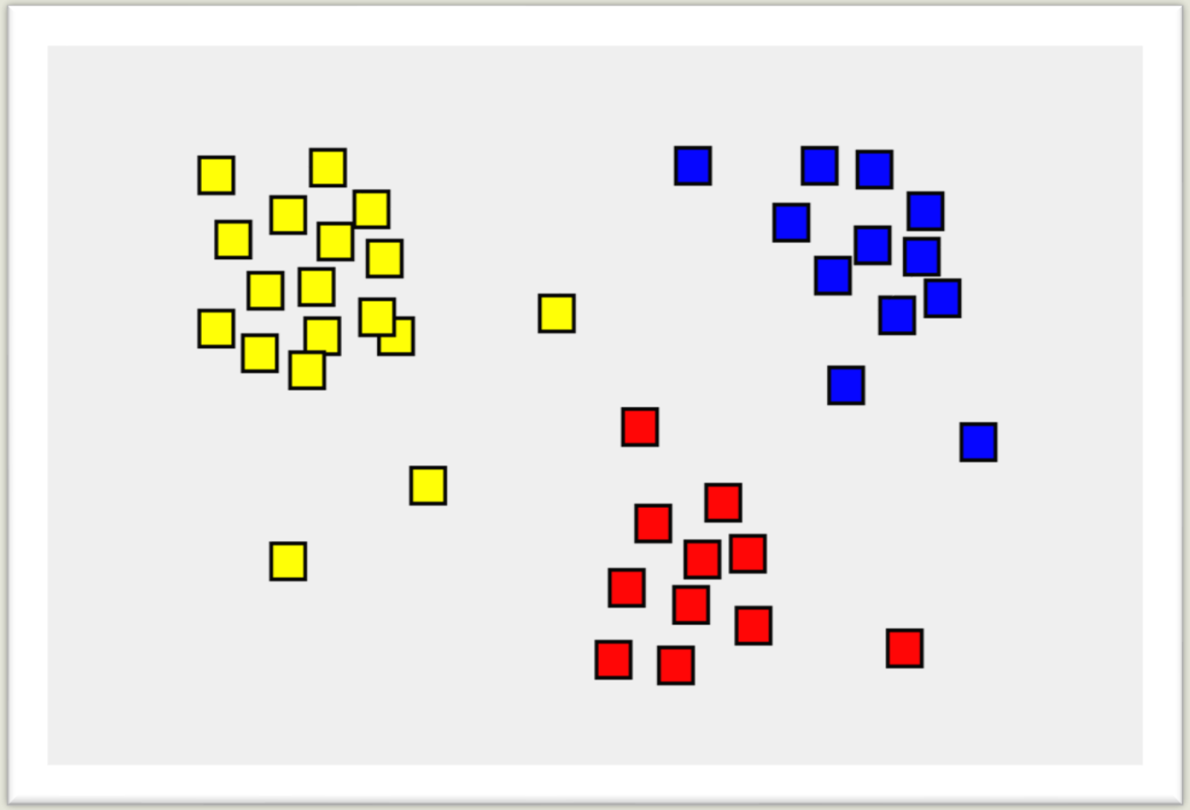
According to the Central Bureau of Statistics the data we are dealing with have high death rate.

We decided to use unsupervised learning to classify the risk groups. Unsupervised learning allows us to automatically identify patterns in the data and classify them into risk groups. This is a more efficient way of dealing with data that has a large rate of sick people, as it requires less manual intervention.

Year	Population	Death count	Death rate
2011	1,952,624	1704	0.0873%
2012	1,990,188	1699	0.0854%
2013	2,030,785	1694	0.0834%
2014	2,070,562	1733	0.0837%
2015	2,110,721	1697	0.0804%
2016	2,150,096	1798	0.0836%
2017	2,191,646	1780	0.0812%
2018	2,229,702	1770	0.0794%
Our data	**12,438**	**197**	**1.5839%**

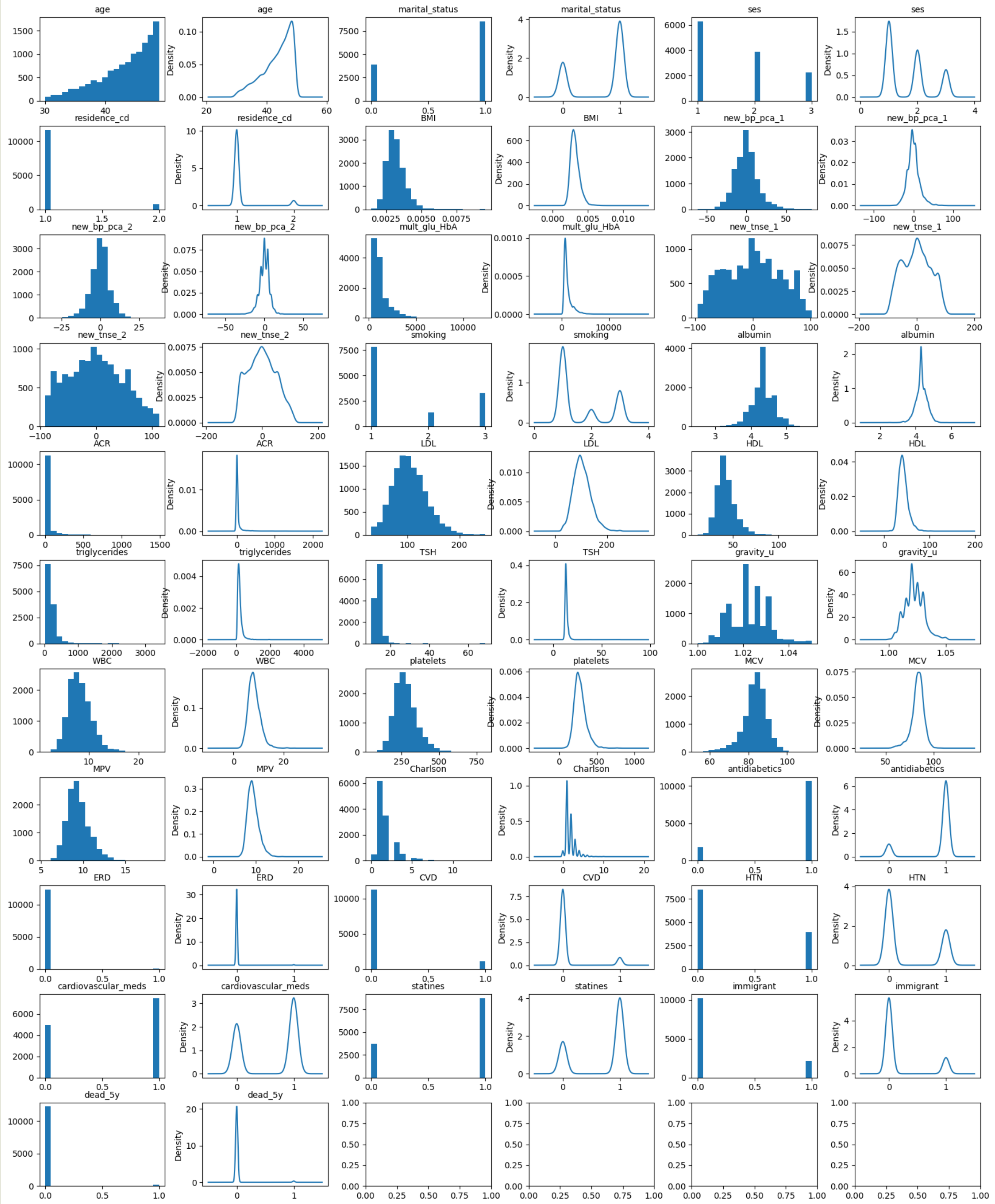
Supervised VS Unsupervised





Data preprocessing

- Null including features
- Categorical features (strings)
- Dimensions reduction using correlative features
- Normalize
- Data balancing


age
marital_status
ses
residence_cd
BMI
new_bp_pca_1
new_bp_pca_2
mult_glu_HbA
new_tnse_1
new_tnse_2
smoking
albumin
ACR
LDL
HDL
triglycerides
TSH
gravity_u
WBC
platelets
MCV
MPV
Charlson
antidiabetics
ERD
CVD
HTN
cardiovascular_meds
statines
immigrant
dead_5y
Density

Null including features

Discover features

- Finding the null rate in each feature contains null

Visualize distribution

- Value count
- Summary of the central tendency, dispersion and shape of a dataset’s distribution
- Histogram and density estimation

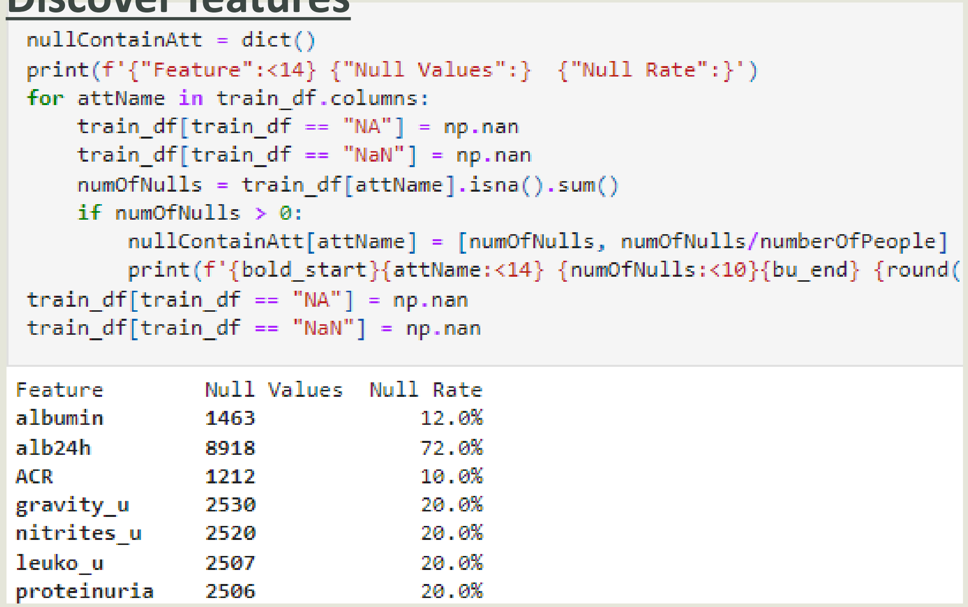
Feature removing

- More than 50% nulls
- More than 50% zeros

Feature filling

- The density estimation like normal – we filled with the median because it is noise insensitive
- Fill the previous values for non normal like density estimation

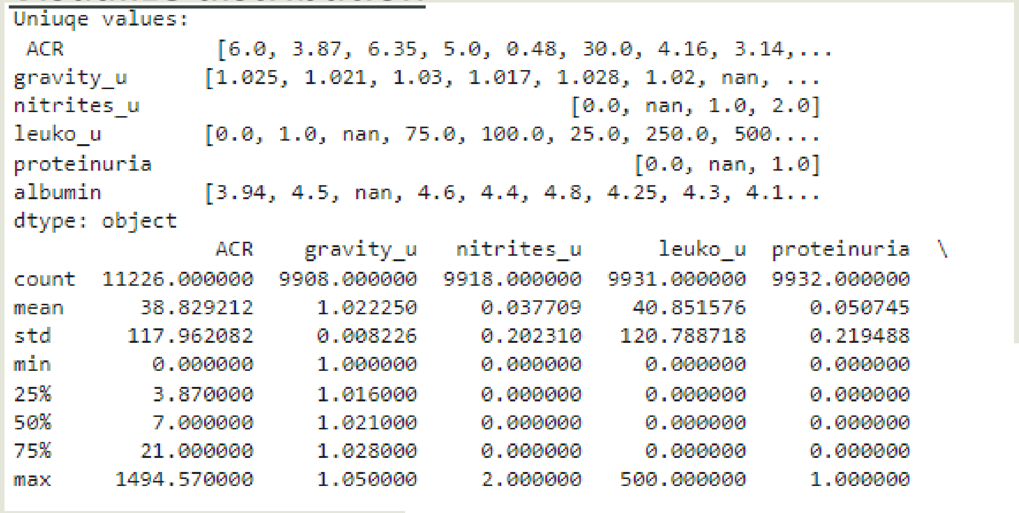
Discover features



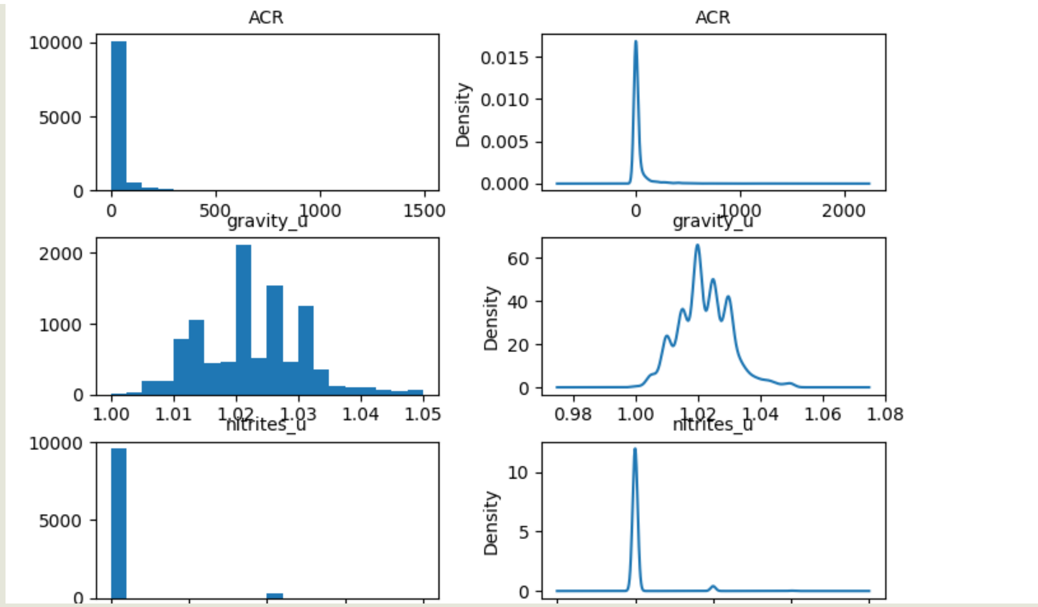
```
nullContainAtt = dict()
print(f'{"Feature":<14} {"Null Values":}  {"Null Rate":}')
for attName in train_df.columns:
    train_df[train_df == "NA"] = np.nan
    train_df[train_df == "NaN"] = np.nan
    numOfNulls = train_df[attName].isna().sum()
    if numOfNulls > 0:
        nullContainAtt[attName] = [numOfNulls, numOfNulls/numberOfPeople]
        print(f'{bold_start}{attName:<14} {numOfNulls:<10}{bu_end} {round(
train_df[train_df == "NA"] = np.nan
train_df[train_df == "NaN"] = np.nan
```

```
Feature        Null Values  Null Rate
albumin        1463            12.0%
alb24h         8918            72.0%
ACR            1212            10.0%
gravity_u      2530            20.0%
nitrites_u     2520            20.0%
leuko_u        2507            20.0%
proteinuria    2506            20.0%
```

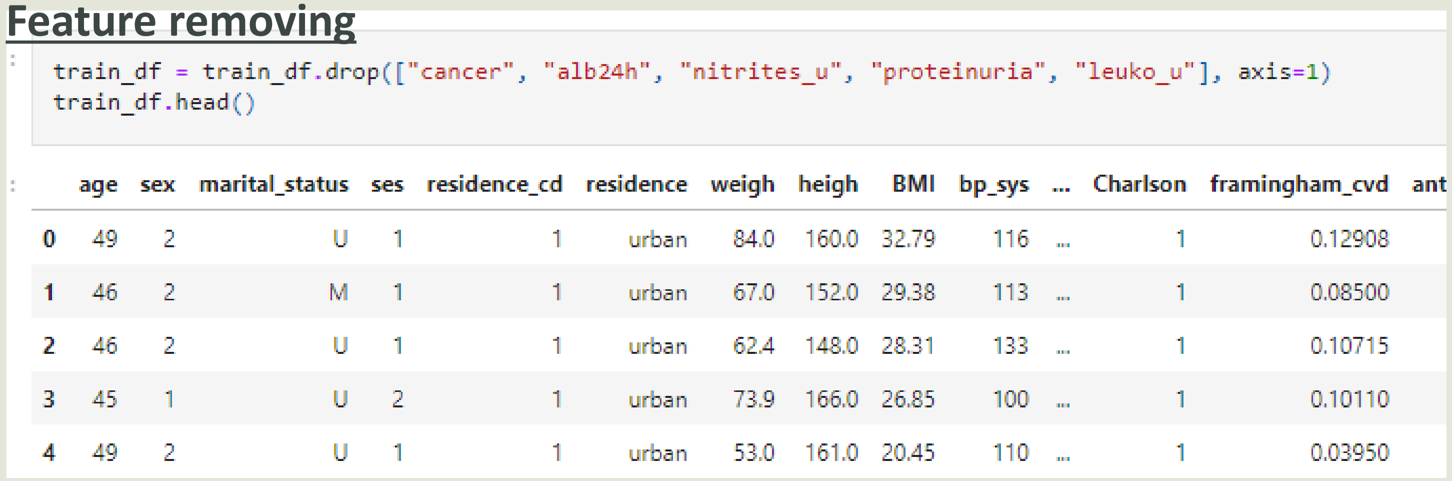
Visualize distribution



```
Uniuqe values:
 ACR            [6.0, 3.87, 6.35, 5.0, 0.48, 30.0, 4.16, 3.14,...
gravity_u      [1.025, 1.021, 1.03, 1.017, 1.028, 1.02, nan, ...
nitrites_u                                [0.0, nan, 1.0, 2.0]
leuko_u        [0.0, 1.0, nan, 75.0, 100.0, 25.0, 250.0, 500....
proteinuria                                    [0.0, nan, 1.0]
albumin        [3.94, 4.5, nan, 4.6, 4.4, 4.8, 4.25, 4.3, 4.1...
dtype: object
                ACR    gravity_u   nitrites_u      leuko_u  proteinuria  \
count  11226.000000  9908.000000  9918.000000  9931.000000  9932.000000
mean      38.829212     1.022250     0.037709    40.851576     0.050745
std      117.962082     0.008226     0.202310   120.788718     0.219488
min        0.000000     1.000000     0.000000     0.000000     0.000000
25%        3.870000     1.016000     0.000000     0.000000     0.000000
50%        7.000000     1.021000     0.000000     0.000000     0.000000
75%       21.000000     1.028000     0.000000     0.000000     0.000000
max     1494.570000     1.050000     2.000000   500.000000     1.000000
```



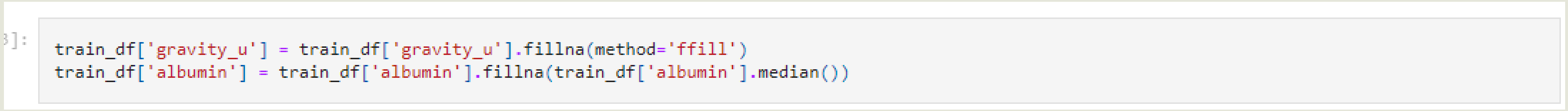
Feature removing



```
train_df = train_df.drop(["cancer", "alb24h", "nitrites_u", "proteinuria", "leuko_u"], axis=1)
train_df.head()
```

	age	sex	marital_status	ses	residence_cd	residence	weigh	heigh	BMI	bp_sys	...	Charlson	framingham_cvd	ant
0	49	2	U	1	1	urban	84.0	160.0	32.79	116	...	1	0.12908	
1	46	2	M	1	1	urban	67.0	152.0	29.38	113	...	1	0.08500	
2	46	2	U	1	1	urban	62.4	148.0	28.31	133	...	1	0.10715	
3	45	1	U	2	1	urban	73.9	166.0	26.85	100	...	1	0.10110	
4	49	2	U	1	1	urban	53.0	161.0	20.45	110	...	1	0.03950	

Feature filling



```
train_df['gravity_u'] = train_df['gravity_u'].fillna(method='ffill')
train_df['albumin'] = train_df['albumin'].fillna(train_df['albumin'].median())
```

Categorical features

Discover features

- Finding the string contain features

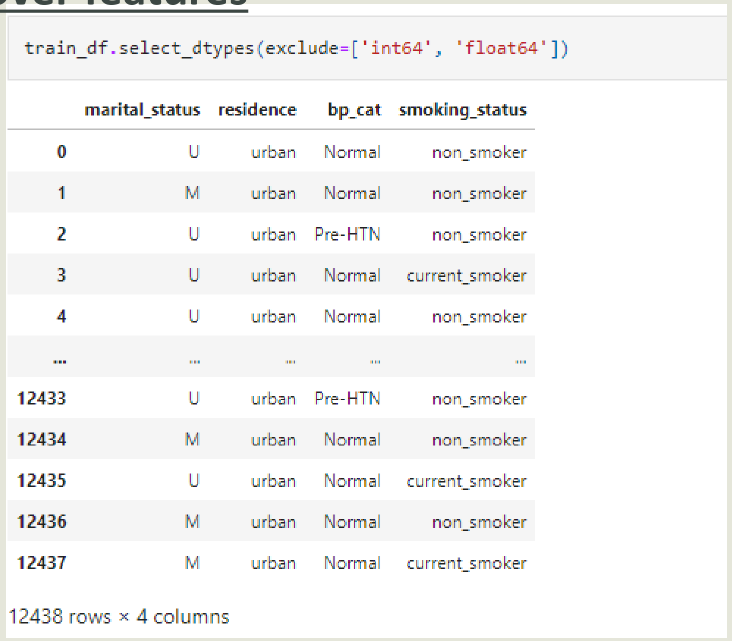
Look for numerical discrete mapping

- Looking for features that have the same information but in numerical form

Convert to numerical

- Understand what are the feature unique values.
- Binary features will map into {0,1}
- Ordered categories will map into discrete numbers from 1 to N groups

Discover features

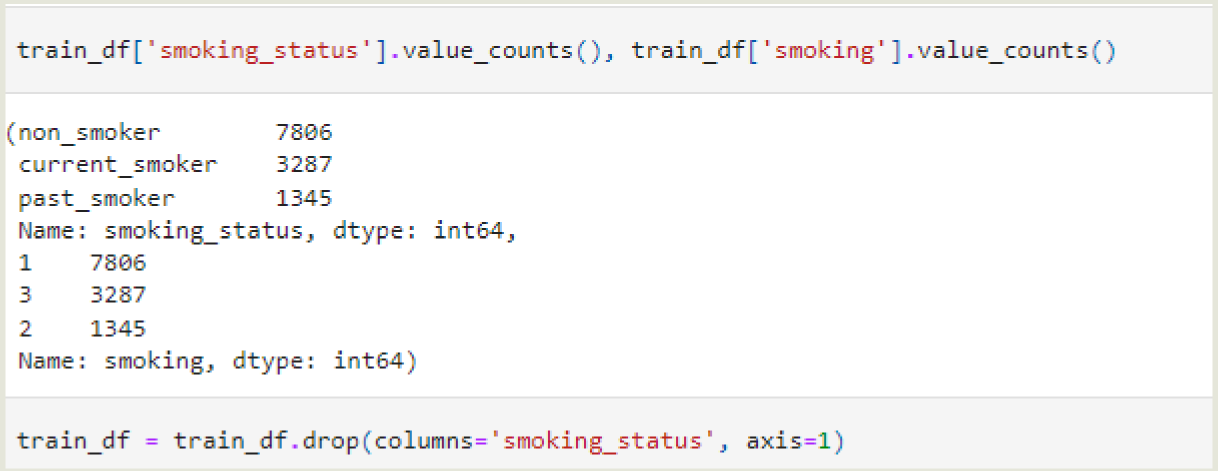


```
train_df.select_dtypes(exclude=['int64', 'float64'])
```

	marital_status	residence	bp_cat	smoking_status
0	U	urban	Normal	non_smoker
1	M	urban	Normal	non_smoker
2	U	urban	Pre-HTN	non_smoker
3	U	urban	Normal	current_smoker
4	U	urban	Normal	non_smoker
...	...	...	...	...
12433	U	urban	Pre-HTN	non_smoker
12434	M	urban	Normal	non_smoker
12435	U	urban	Normal	current_smoker
12436	M	urban	Normal	non_smoker
12437	M	urban	Normal	current_smoker

12438 rows × 4 columns

Look for numerical discrete mapping

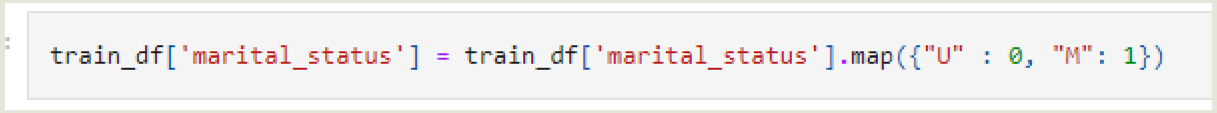


```
train_df['smoking_status'].value_counts(), train_df['smoking'].value_counts()
```

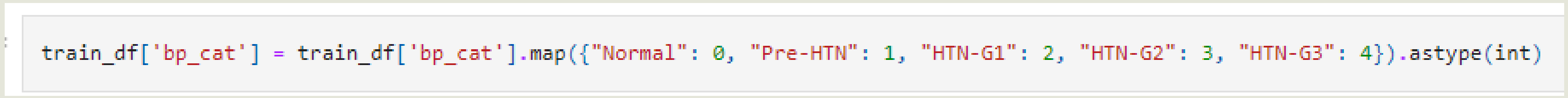
```
(non_smoker        7806
 current_smoker    3287
 past_smoker       1345
 Name: smoking_status, dtype: int64,
 1    7806
 3    3287
 2    1345
 Name: smoking, dtype: int64)
```

```
train_df = train_df.drop(columns='smoking_status', axis=1)
```

Convert to numerical



```
train_df['marital_status'] = train_df['marital_status'].map({"U" : 0, "M": 1})
```



```
train_df['bp_cat'] = train_df['bp_cat'].map({"Normal": 0, "Pre-HTN": 1, "HTN-G1": 2, "HTN-G2": 3, "HTN-G3": 4}).astype(int)
```

Dimensions reduction using correlative features

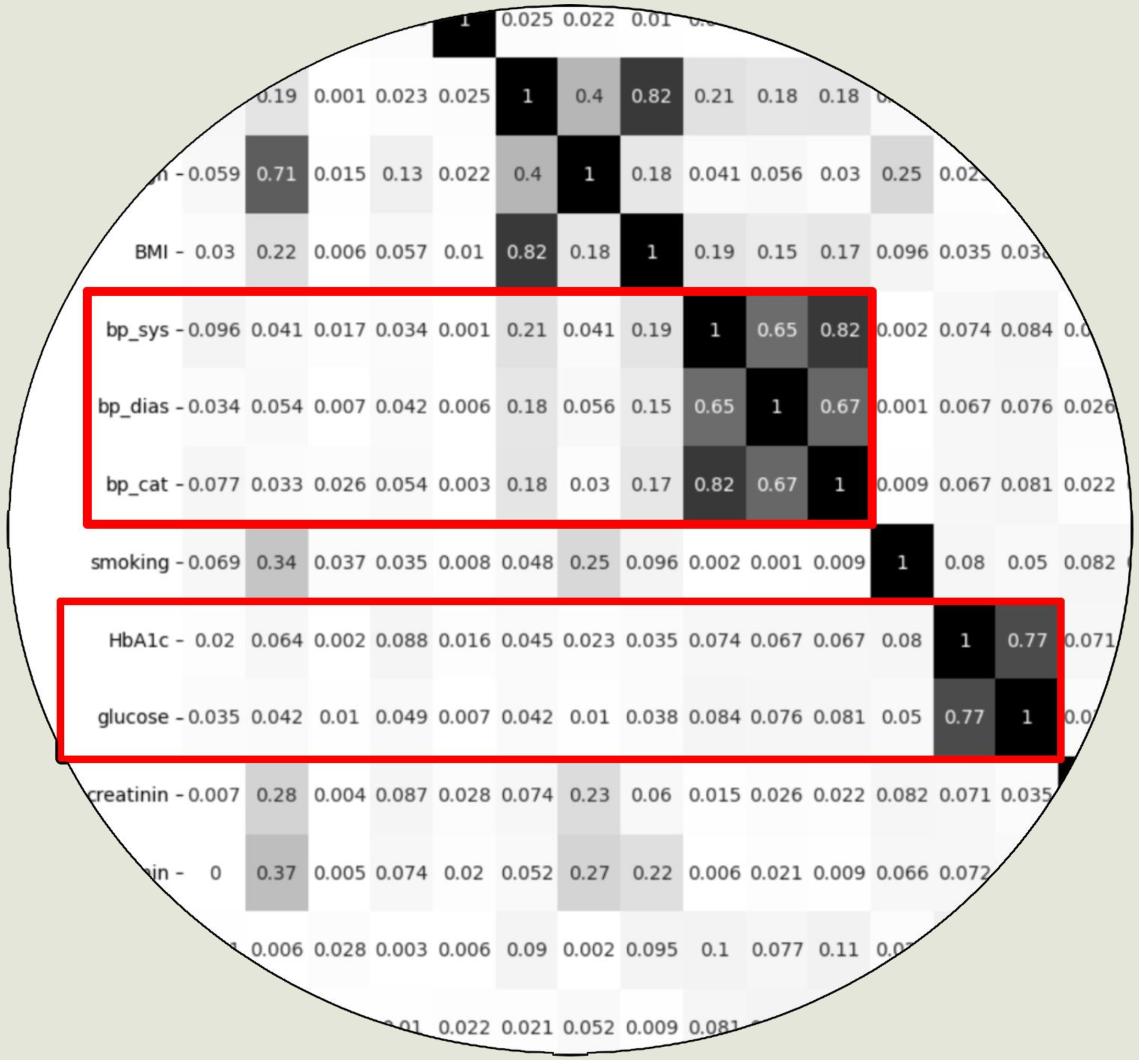
Correlation matrices

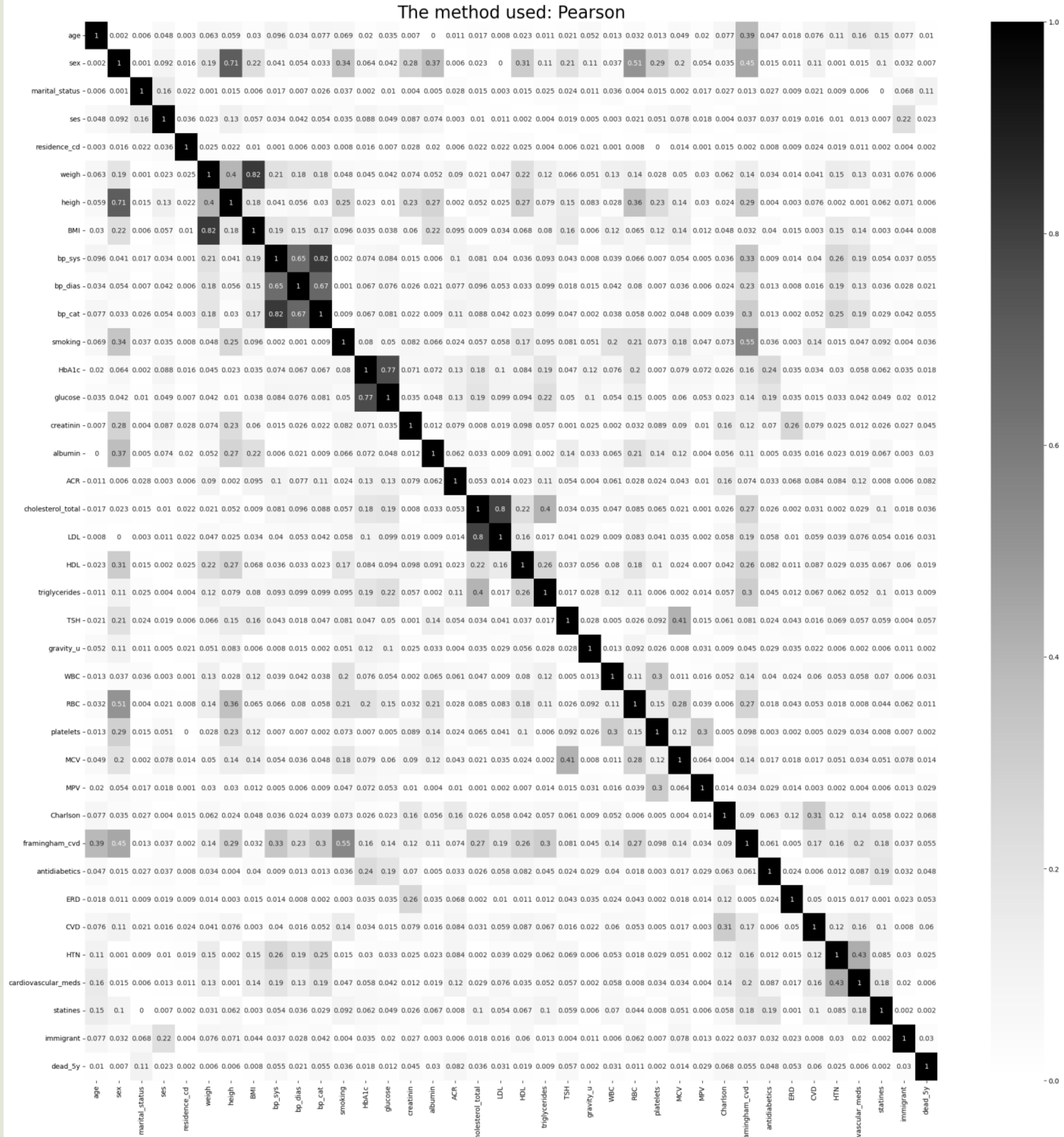
Pearson

- Measures the **linear** association between two variables and is calculated as the covariance of the two variables divided by the product of their standard deviations.
- Used for continuous feature
- -1 to 1 range.
  -1: perfect negative linear relationship
  0: no linear relationship
  1: perfect positive linear relationship.

Spearman

- Rank-based measure of association and is calculated as the Pearson correlation between the ranks of the two variables. It is used when the relationship between the two variables is **not linear.**
- Used when the data is ordinal or has outliers
- -1 to 1 range.
  -1: perfect negative monotonic relationship
  0: no monotonic relationship
  1: perfect positive monotonic relationship.


BMI
bp_sys
bp_dias
bp_cat
smoking
HbA1c
glucose
creatinin


The method used: Pearson
age
sex
marital_status
ses
residence_cd
weigh
heigh
BMI
bp_sys
bp_dias
bp_cat
smoking
HbA1c
glucose
creatinin
albumin
ACR
cholesterol_total
LDL
HDL
triglycerides
TSH
gravity_u
WBC
RBC
platelets
MCV
MPV
Charlson
framingham_cvd
antidiabetics
ERD
CVD
HTN
cardiovascular_meds
statines
immigrant
dead_5y
1.0
0.8
0.6
0.4
0.2
0.0

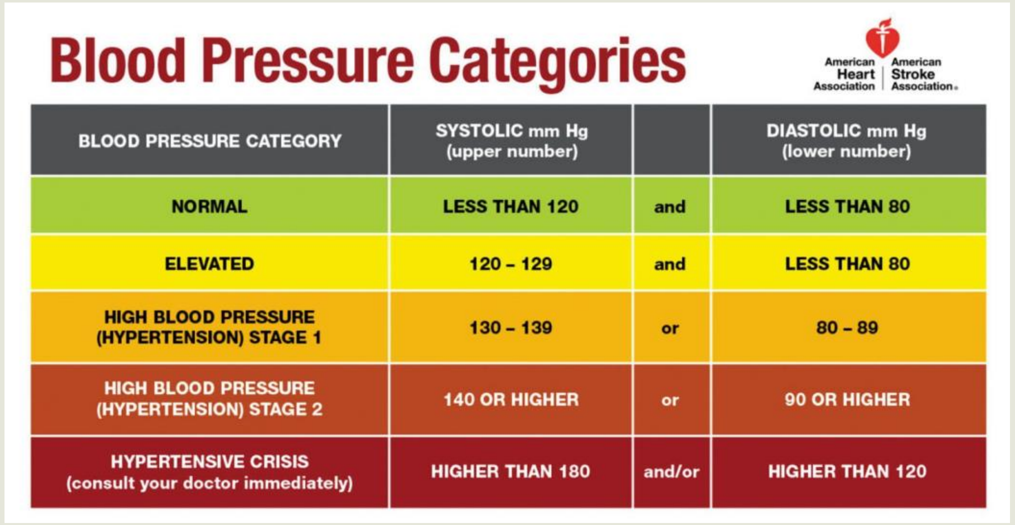
Continuous Dimension Reduction examples

BP features: we can see high correlation between \`bp_sys\`, \`bp_dias\` and \`bp_cat\`.

Why we used PCA:

1. Dimension reduction – less features == less computing power.
2. Creating new two orthogonal features – the new features have no correlation.

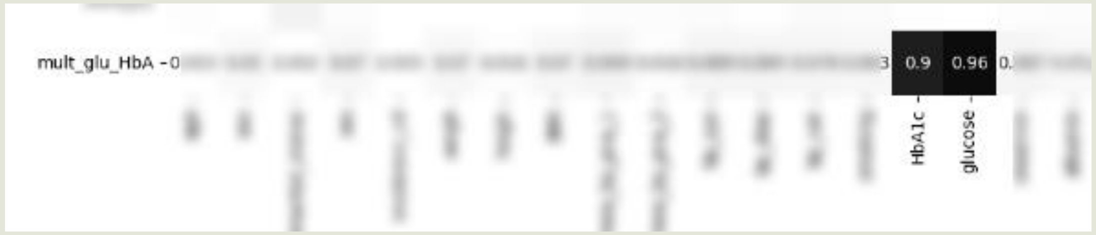
Trail and Error: \`glucose\` and \`HbA1c\` features have 0.77 correlation rate, by using trail and error approach we found out that multiply those features correlates well with both features and we can drop both and use the new feature

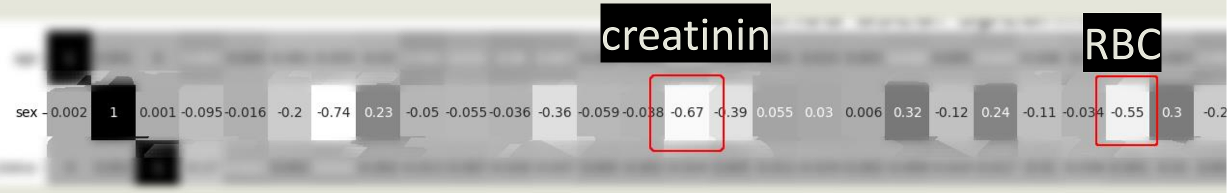


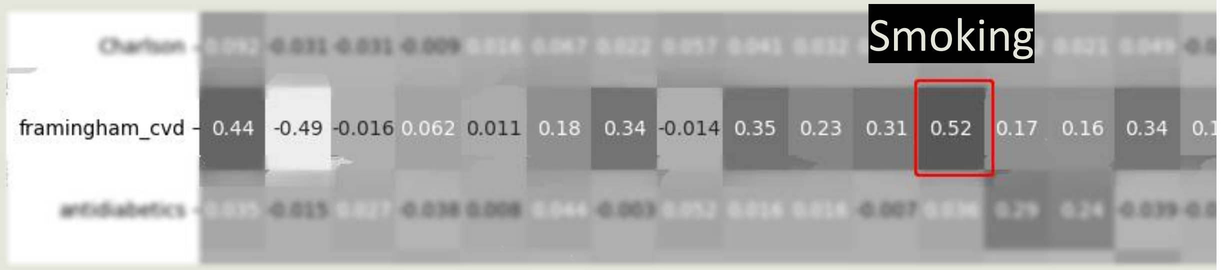
Blood Pressure Categories

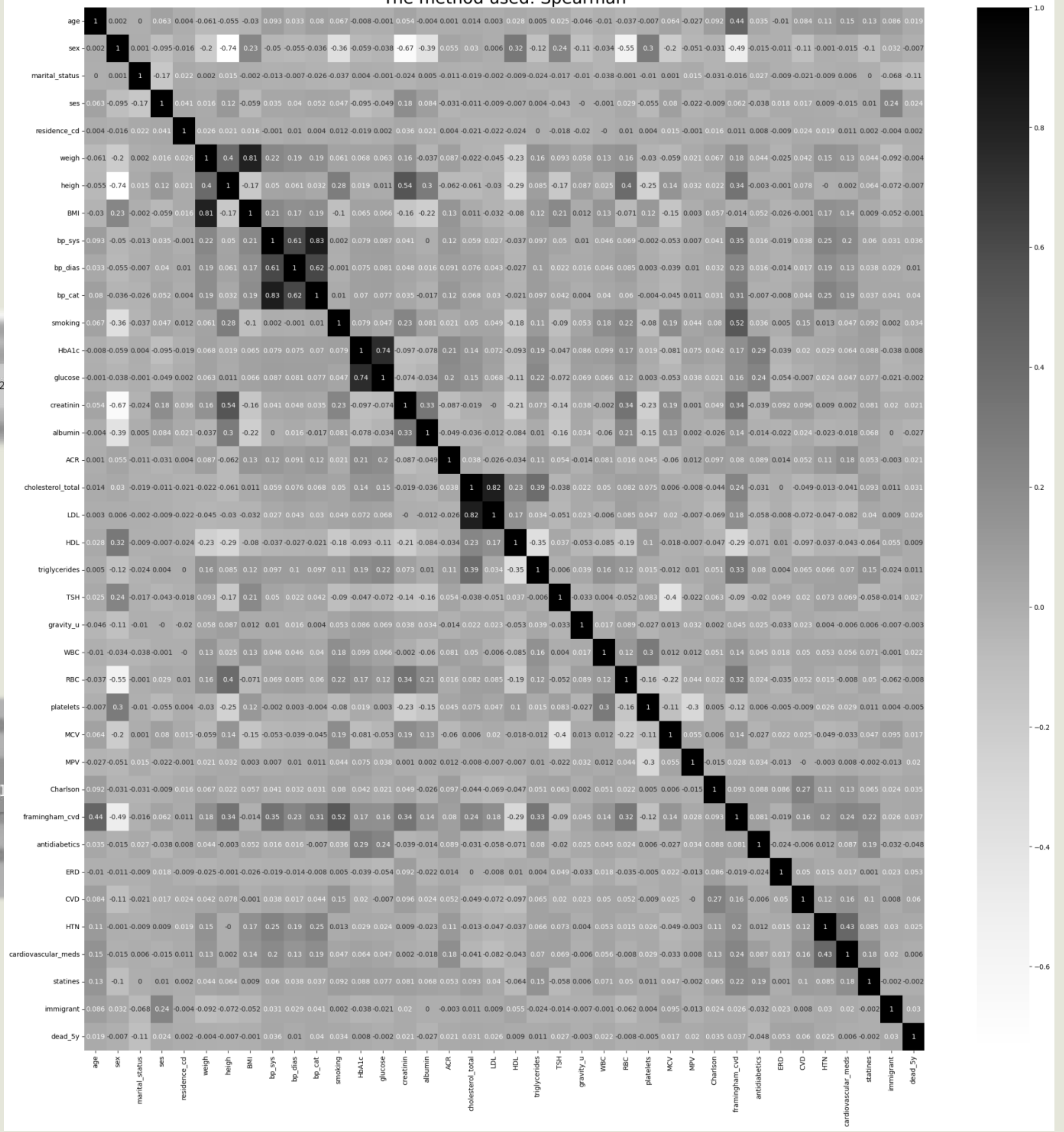
American Heart Association | American Stroke Association.

BLOOD PRESSURE CATEGORY	SYSTOLIC mm Hg (upper number)		DIASTOLIC mm Hg (lower number)
NORMAL	LESS THAN 120	and	LESS THAN 80
ELEVATED	120 – 129	and	LESS THAN 80
HIGH BLOOD PRESSURE (HYPERTENSION) STAGE 1	130 – 139	or	80 – 89
HIGH BLOOD PRESSURE (HYPERTENSION) STAGE 2	140 OR HIGHER	or	90 OR HIGHER
HYPERTENSIVE CRISIS (consult your doctor immediately)	HIGHER THAN 180	and/or	HIGHER THAN 120



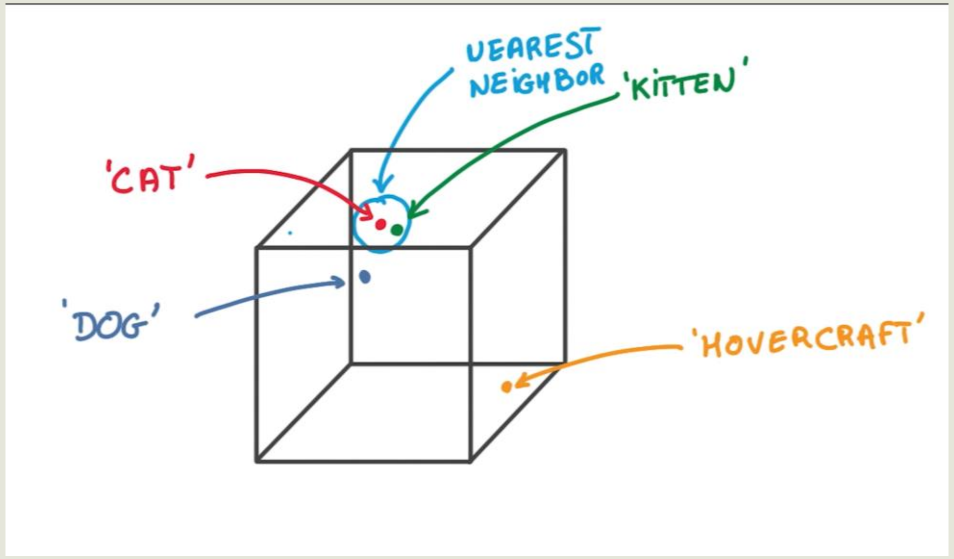

creatinin
RBC
sex
-0.002 1 0.001 -0.095 -0.016 -0.2 -0.74 0.23 -0.05 -0.055 -0.036 -0.36 -0.059 -0.038 -0.67 -0.39 0.055 0.03 0.006 0.32 -0.12 0.24 -0.11 -0.034 -0.55 0.3 -0.2

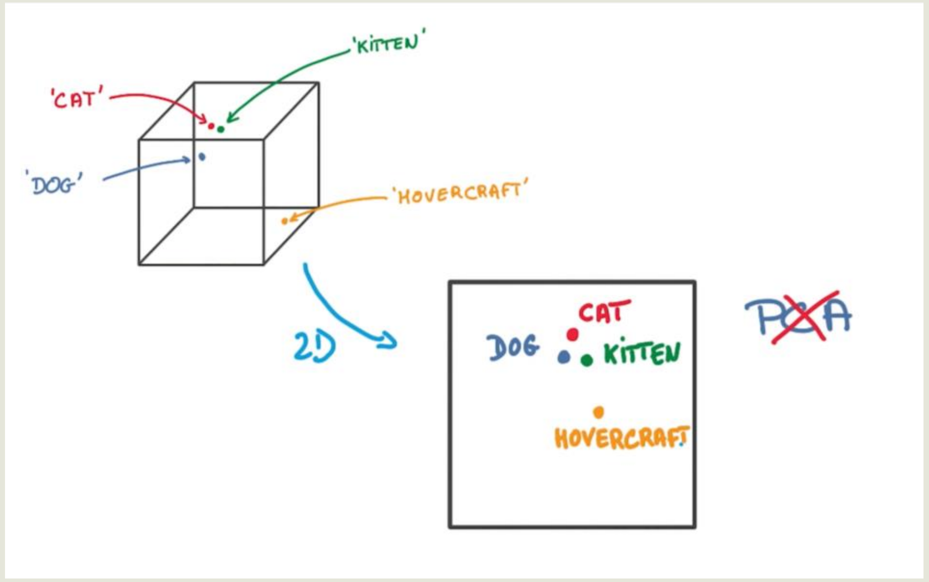

Smoking
Charlson
framingham_cvd
0.44 -0.49 -0.016 0.062 0.011 0.18 0.34 -0.014 0.35 0.23 0.31 0.52 0.17 0.16 0.34 0.1
antidiabetics

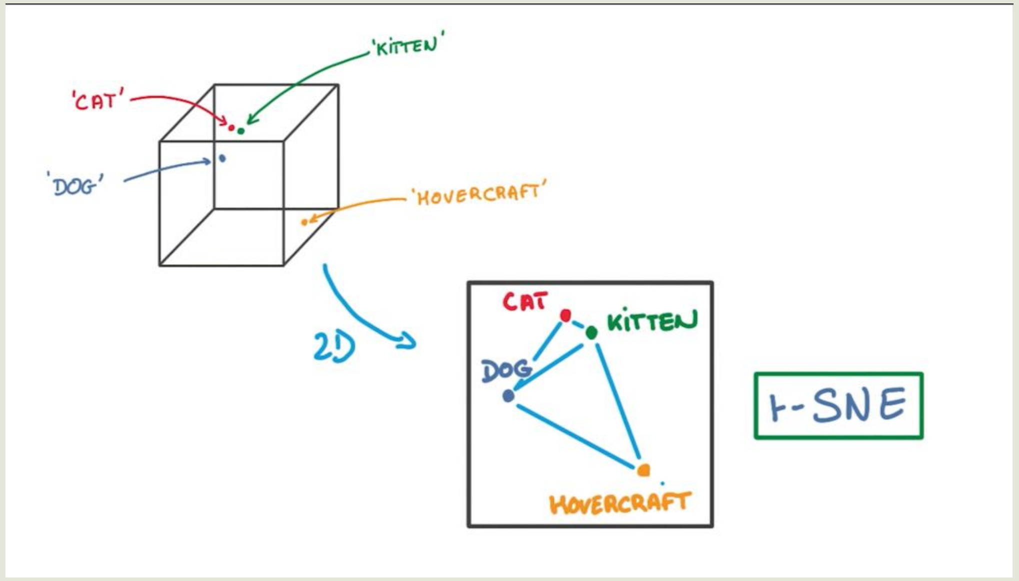

The method used: Spearman
age
sex
marital_status
ses
residence_cd
weigh
heigh
BMI
bp_sys
bp_dias
bp_cat
smoking
HbA1c
glucose
creatinin
albumin
ACR
cholesterol_total
LDL
HDL
triglycerides
TSH
gravity_u
WBC
RBC
platelets
MCV
MPV
Charlson
framingham_cvd
antidiabetics
ERD
CVD
HTN
cardiovascular_meds
statines
immigrant
dead_5y
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

District Dimension Reduction examples

- **TSNE (t-distributed Stochastic Neighbor Embedding):**
  1. **Compute pairwise similarities -** calculate the pairwise similarities between all data points in the high-dimensional space. This can be done using a similarity measure such as the Euclidean or Gaussian kernel.
  2. **Initialize the low-dimensional representation** - The next step is to randomly initialize the low-dimensional representation of the data points. This is typically done in a 2D or 3D plot.
  3. **Optimize the cost function** - The main objective of t-SNE is to minimize a cost function that measures the difference between the pairwise similarities in the high-dimensional space and the pairwise similarities in the low-dimensional space. The cost function is optimized using gradient descent or similar optimization techniques.
  4. **Compute the low-dimensional representation** - After the cost function is minimized, the final low-dimensional representation of the data is obtained. This representation can then be plotted in a 2D or 3D scatter plot to visualize the relationships between the data points.

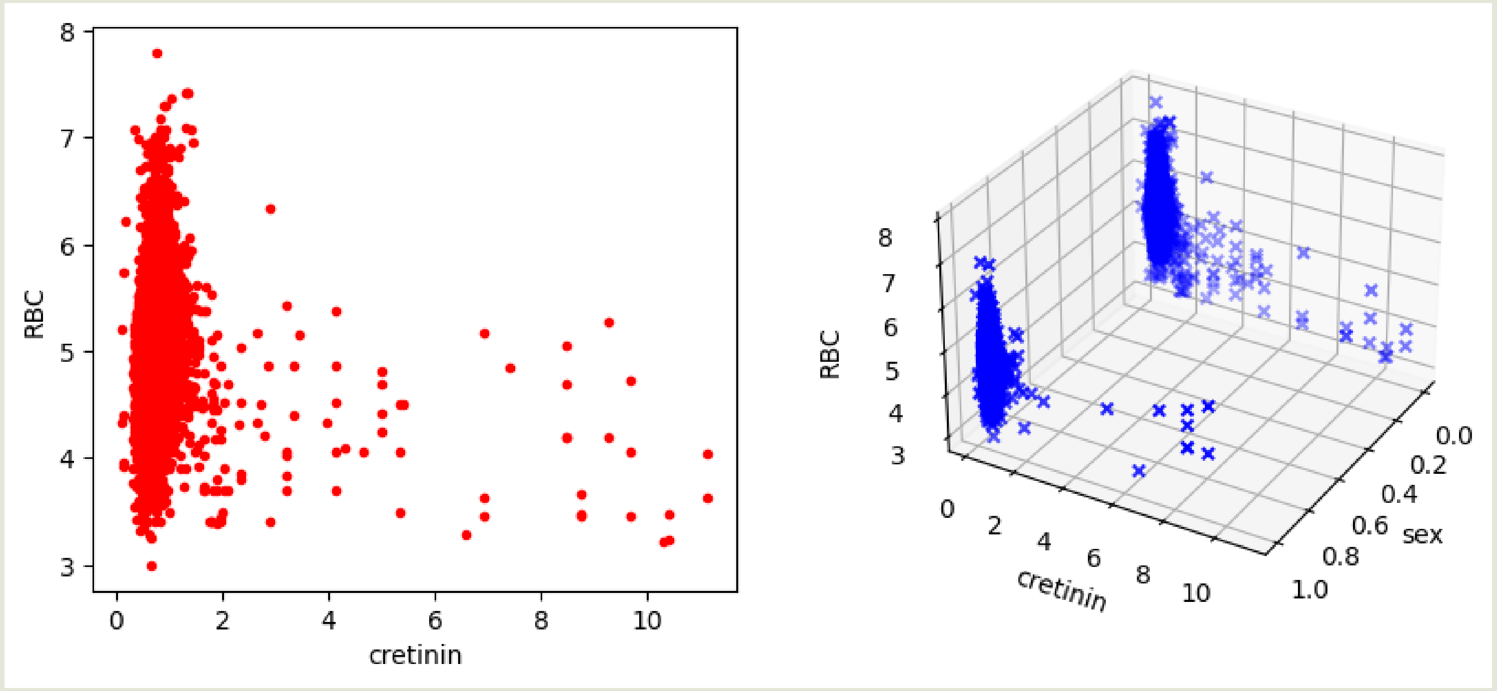

NEAREST NEIGHBOR
'KITTEN'
'CAT'
'DOG'
'HOVERCRAFT'

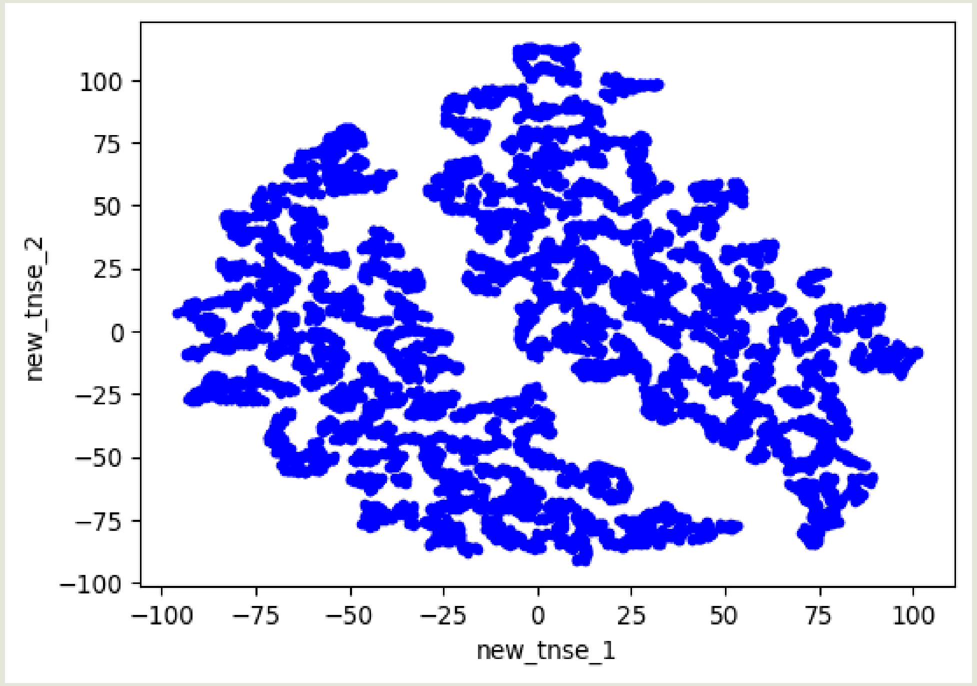

'KITTEN'
'CAT'
'DOG'
'HOVERCRAFT'
2D
CAT
DOG
KITTEN
HOVERCRAFT
~~PCA~~

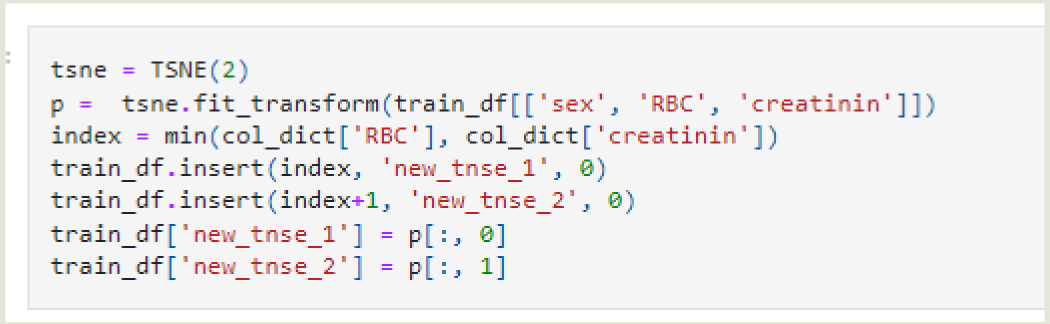

'KITTEN'
'CAT'
'DOG'
'HOVERCRAFT'
2D
CAT
KITTEN
DOG
HOVERCRAFT
t-SNE

Original features

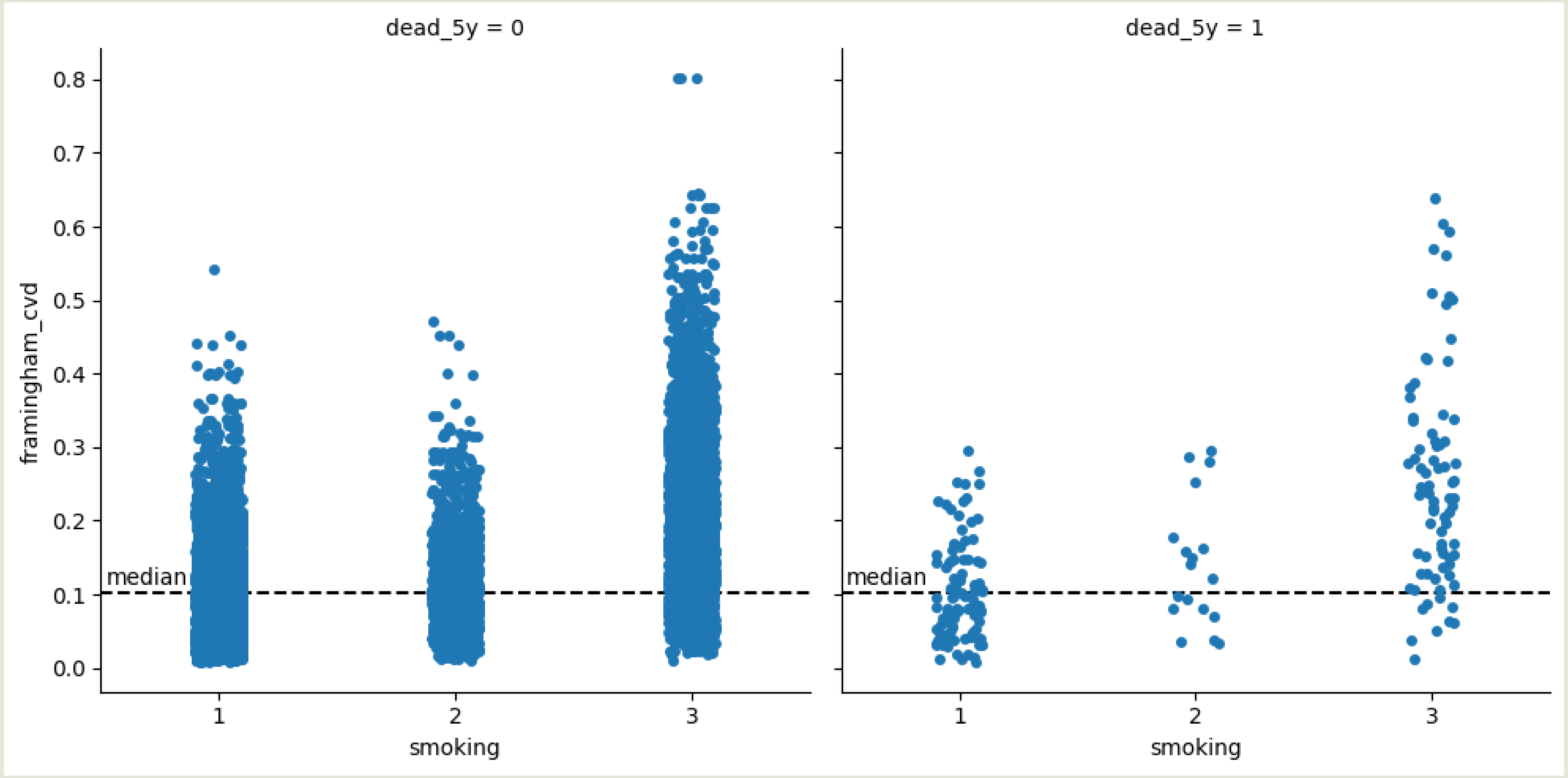
TNSE new features







```
tsne = TSNE(2)
p =  tsne.fit_transform(train_df[['sex', 'RBC', 'creatinin']])
index = min(col_dict['RBC'], col_dict['creatinin'])
train_df.insert(index, 'new_tnse_1', 0)
train_df.insert(index+1, 'new_tnse_2', 0)
train_df['new_tnse_1'] = p[:, 0]
train_df['new_tnse_2'] = p[:, 1]
```


dead_5y = 0
dead_5y = 1
framingham_cvd
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
median
median
1
2
3
smoking
smoking

Normalization

Min-Max Normalization

- Also referred as **Feature Scaling,** performs linear transformation on the data.
- This technique gets all the scaled data in the range (0, 1).
- Disadvantage: Does not handle outliers very well.
  For example: if you have 99 values between 0 and 40, and one value is 100, then the 99 values will all be transformed to a value between 0 and 0.4. That data is just as squished as before!
- We used the method for normal distribution like features.
- Formula:

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

Data Balancing

SMOTE-NC

- SMOTE-NC (Synthetic Minority Over-sampling Technique for Nominal and Continuous) is a technique for oversampling imbalanced datasets in machine learning.

- It combines the Synthetic Minority Over-sampling Technique (SMOTE) with the nearest neighbors correction for continuous features (NC).

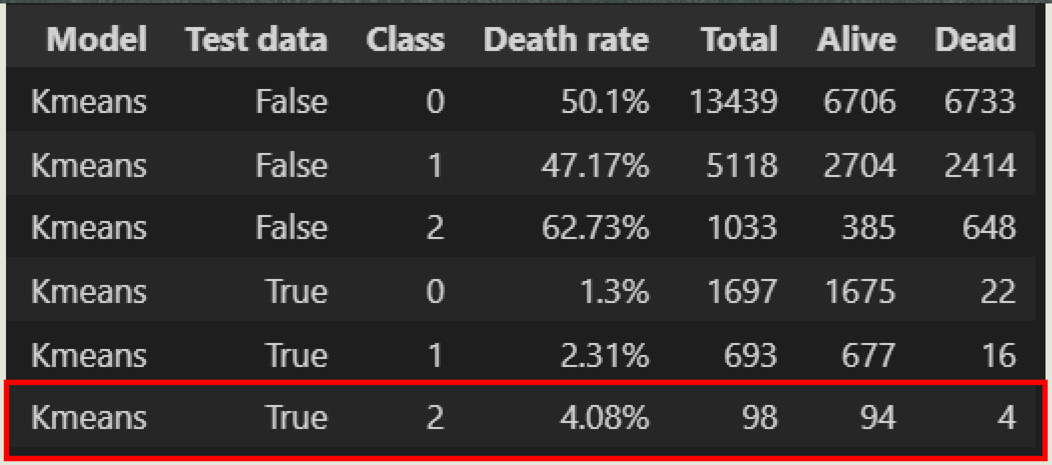
  Let $\boldsymbol{x_i}$ *be a minority class sample and* $\boldsymbol{N_i}$ *be its* $\boldsymbol{k}$ *nearest neighbors.*
  Let $\boldsymbol{d_{ij}}$ *be the Euclidean distance between* $\boldsymbol{x_i}$ *and* $\boldsymbol{x_j}$ *for* $\boldsymbol{j}$ *in* $\boldsymbol{N_i}$*.*
  Then, the weighted average of $\boldsymbol{x_i}$ *and its nearest neighbors is computed as follows:*
  $$\boldsymbol{x'_i} = \boldsymbol{x_i} + \boldsymbol{r} * (\boldsymbol{x_j} - \boldsymbol{x_i})$$
  where $\boldsymbol{r}$ *is a random number between* 0 *and* 1 *and* $\boldsymbol{j}$ *is randomly selected from* $\boldsymbol{N_i}$*.*

- SMOTE-NC generates synthetic samples for the minority class by interpolating between minority class samples and their nearest neighbors. The nearest neighbors correction is applied to ensure the generated samples are consistent with the distribution of the continuous features in the minority class.

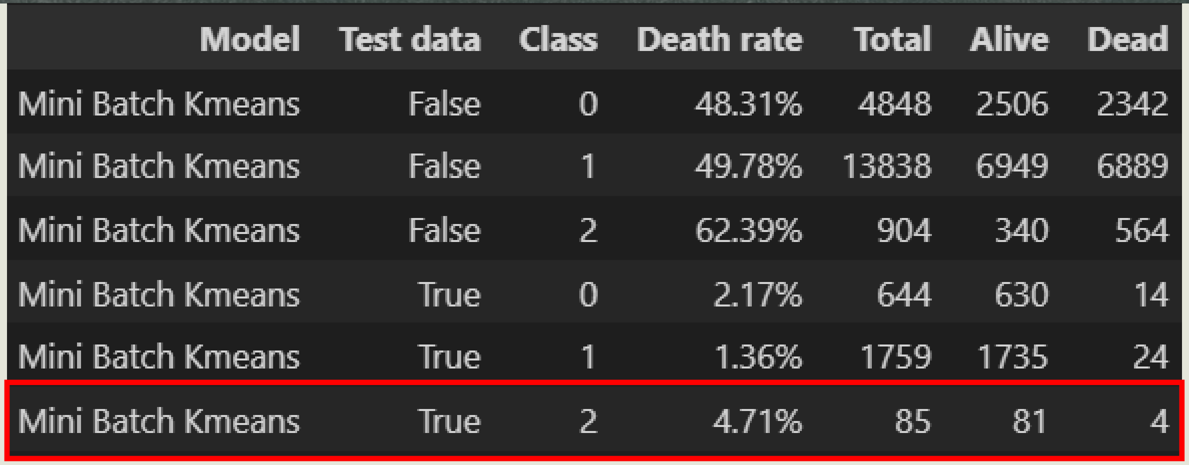
Unsupervised Learning

Cluster our data into risk groups before the supervised learning

K Means & Mini Batch K Means



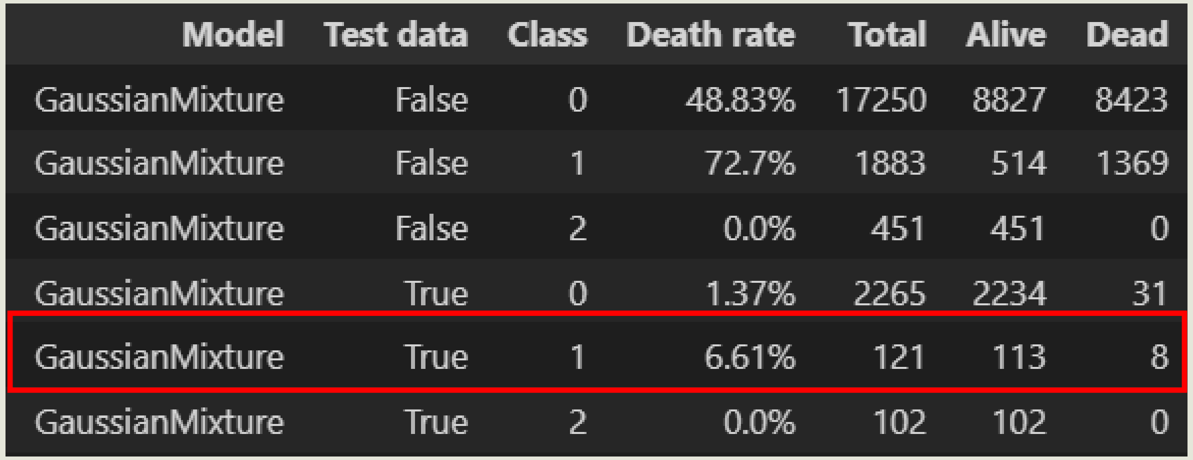
Model	Test data	Class	Death rate	Total	Alive	Dead
Kmeans	False	0	50.1%	13439	6706	6733
Kmeans	False	1	47.17%	5118	2704	2414
Kmeans	False	2	62.73%	1033	385	648
Kmeans	True	0	1.3%	1697	1675	22
Kmeans	True	1	2.31%	693	677	16
Kmeans	True	2	4.08%	98	94	4



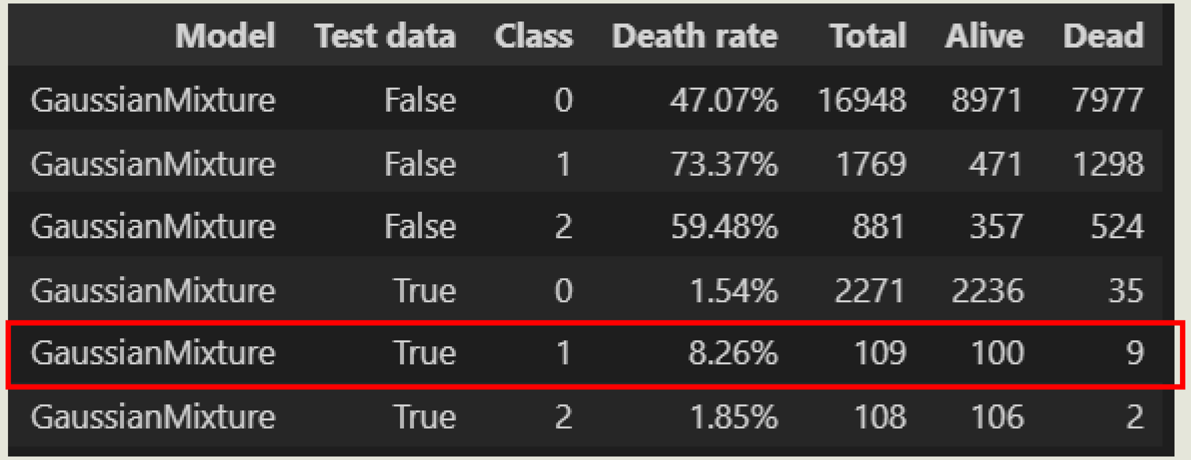
Model	Test data	Class	Death rate	Total	Alive	Dead
Mini Batch Kmeans	False	0	48.31%	4848	2506	2342
Mini Batch Kmeans	False	1	49.78%	13838	6949	6889
Mini Batch Kmeans	False	2	62.39%	904	340	564
Mini Batch Kmeans	True	0	2.17%	644	630	14
Mini Batch Kmeans	True	1	1.36%	1759	1735	24
Mini Batch Kmeans	True	2	4.71%	85	81	4

- We can easily notice that both models created new risk group with much higher risk than the other and the original data.
- It seems like a good direction, so we tested some more models, the following model works best form our perspective.

Gaussian Mixture Model



Model	Test data	Class	Death rate	Total	Alive	Dead
GaussianMixture	False	0	48.83%	17250	8827	8423
GaussianMixture	False	1	72.7%	1883	514	1369
GaussianMixture	False	2	0.0%	451	451	0
GaussianMixture	True	0	1.37%	2265	2234	31
GaussianMixture	True	1	6.61%	121	113	8
GaussianMixture	True	2	0.0%	102	102	0



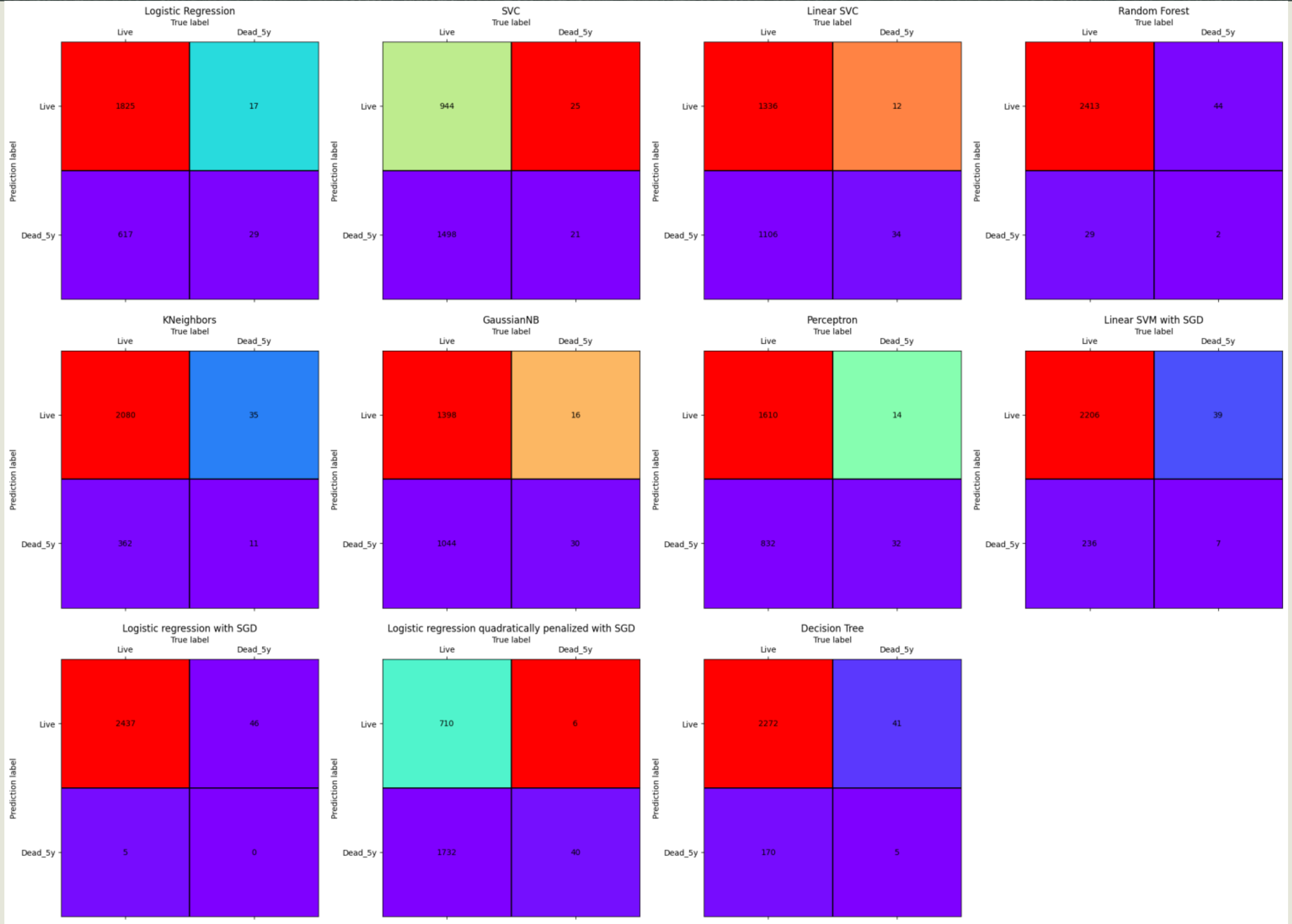
Model	Test data	Class	Death rate	Total	Alive	Dead
GaussianMixture	False	0	47.07%	16948	8971	7977
GaussianMixture	False	1	73.37%	1769	471	1298
GaussianMixture	False	2	59.48%	881	357	524
GaussianMixture	True	0	1.54%	2271	2236	35
GaussianMixture	True	1	8.26%	109	100	9
GaussianMixture	True	2	1.85%	108	106	2

- The model give us more accurate risk group dividing, we'll use the classes that GMM made and insert it into the data.
- We are trying to use weight for classes to affect the supervised model.

Supervised Learning

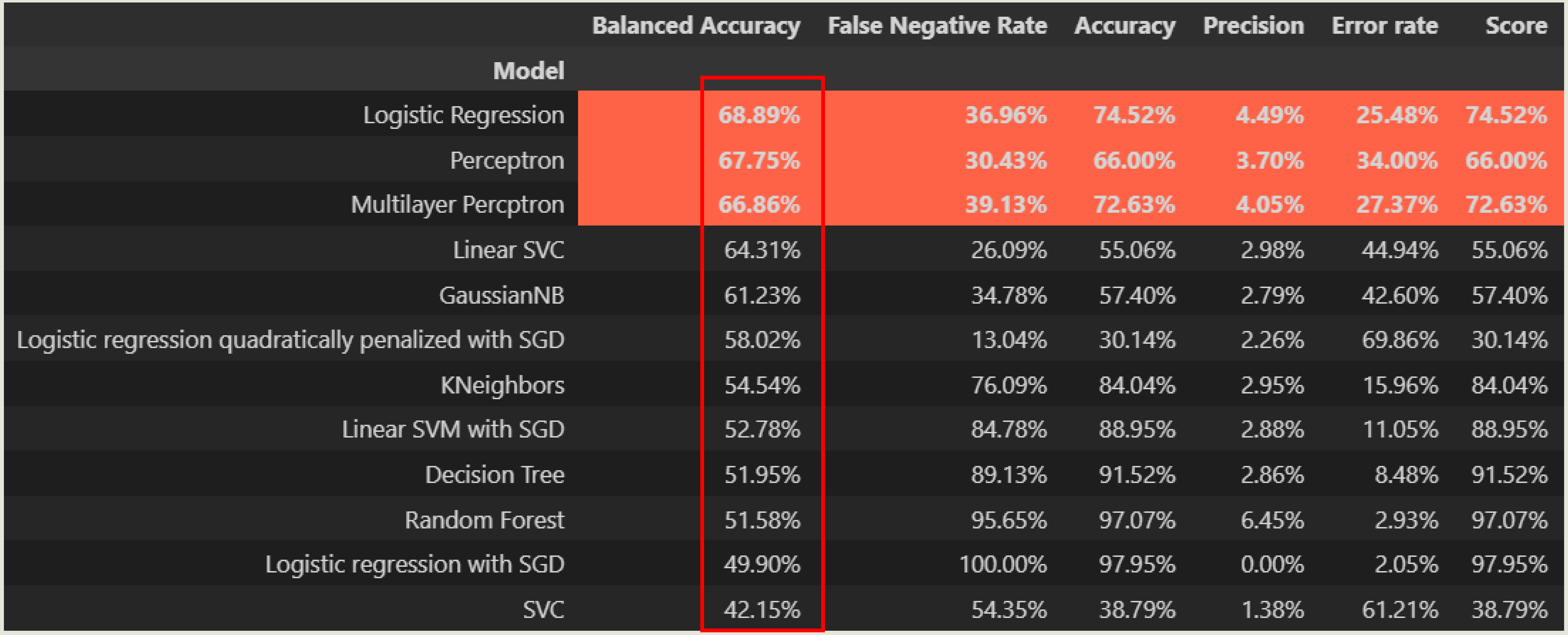
How should we choose our model?

Confusion Matrices & Related Rates



In an imbalanced dataset, one class typically has a much larger number of instances than the other class, which can lead to a bias towards the majority class.

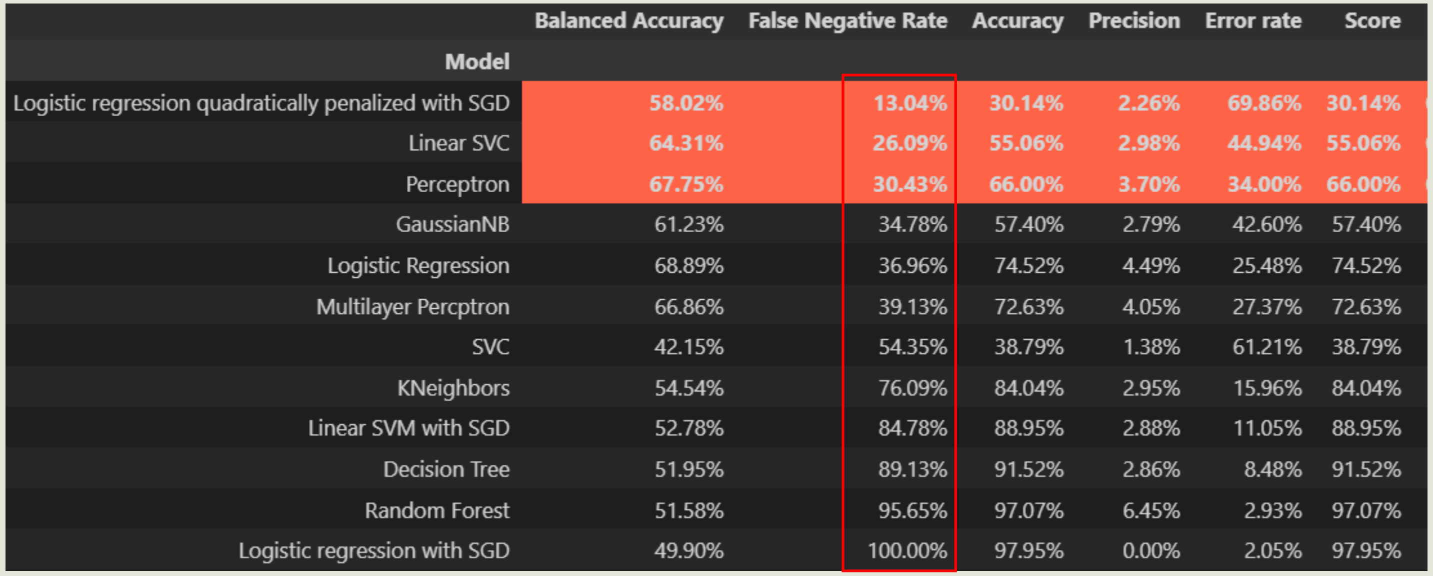
When using traditional metrics such as accuracy, a model can still have a high accuracy score even if it does not perform well on the minority class. For example, a model that always predicts the majority class will have a high accuracy, even though it does not perform well on the minority class.



	Balanced Accuracy	False Negative Rate	Accuracy	Precision	Error rate	Score
Model						
Logistic Regression	**68.89%**	**36.96%**	**74.52%**	**4.49%**	**25.48%**	**74.52%**
Perceptron	**67.75%**	**30.43%**	**66.00%**	**3.70%**	**34.00%**	**66.00%**
Multilayer Percptron	**66.86%**	**39.13%**	**72.63%**	**4.05%**	**27.37%**	**72.63%**
Linear SVC	64.31%	26.09%	55.06%	2.98%	44.94%	55.06%
GaussianNB	61.23%	34.78%	57.40%	2.79%	42.60%	57.40%
Logistic regression quadratically penalized with SGD	58.02%	13.04%	30.14%	2.26%	69.86%	30.14%
KNeighbors	54.54%	76.09%	84.04%	2.95%	15.96%	84.04%
Linear SVM with SGD	52.78%	84.78%	88.95%	2.88%	11.05%	88.95%
Decision Tree	51.95%	89.13%	91.52%	2.86%	8.48%	91.52%
Random Forest	51.58%	95.65%	97.07%	6.45%	2.93%	97.07%
Logistic regression with SGD	49.90%	100.00%	97.95%	0.00%	2.05%	97.95%
SVC	42.15%	54.35%	38.79%	1.38%	61.21%	38.79%

Model Picking

- If we look from an unbiased point of view, we should use the model with the highest balanced accuracy, therefore we'll get the best result for any data.
- On the other hand, we are testing death prediction in 5 years, taking this into consider should maybe measure the **False Negative Rate**.
  It can show how many dead in 5 years people the model "missed".



Model	Balanced Accuracy	False Negative Rate	Accuracy	Precision	Error rate	Score
Logistic regression quadratically penalized with SGD	58.02%	13.04%	30.14%	2.26%	69.86%	30.14%
Linear SVC	64.31%	26.09%	55.06%	2.98%	44.94%	55.06%
Perceptron	67.75%	30.43%	66.00%	3.70%	34.00%	66.00%
GaussianNB	61.23%	34.78%	57.40%	2.79%	42.60%	57.40%
Logistic Regression	68.89%	36.96%	74.52%	4.49%	25.48%	74.52%
Multilayer Percptron	66.86%	39.13%	72.63%	4.05%	27.37%	72.63%
SVC	42.15%	54.35%	38.79%	1.38%	61.21%	38.79%
KNeighbors	54.54%	76.09%	84.04%	2.95%	15.96%	84.04%
Linear SVM with SGD	52.78%	84.78%	88.95%	2.88%	11.05%	88.95%
Decision Tree	51.95%	89.13%	91.52%	2.86%	8.48%	91.52%
Random Forest	51.58%	95.65%	97.07%	6.45%	2.93%	97.07%
Logistic regression with SGD	49.90%	100.00%	97.95%	0.00%	2.05%	97.95%

Conclusions

- Imbalanced data provides a lot of difficulties that we must chose a road along the way, it can be a waste of time in some cases.
- In our case the we can notice the tradeoffs in the models and keep it in mind while considering what should we use.
- For future work we should create a new model that will optimize the following Min-Max problem:
    - ➢ Minimize the false negative rate
    - ➢ Maximize the balanced accuracy