Major 1 Report

Idan Albo, Amit Grosman

<u>Part 1 – Data Loading and First Look</u>

Q1.

The dataset contains 1250 rows, and 26 columns.

Q2.

The output of value_counts for conversations_per_day feature is:

```
1     232
2     213
3     191
4     158
5     127
0     117
6      69
7      46
8      34
9      21
10     20
12      7
11      5
13      4
15      3
14      2
16      1
Name: conversations_per_day, dtype: int64
```

This feature represents the number of conversations that a certain subject performed during a day.
This feature is an integer (and not continuous), therefore its type isn't continuous, but it is also not categorical, because we can define a natural order to its values.

Q3.

Feature name	Description	Type
patient_id	The subject's id	Other
age	The subject's age	Ordinal
sex	The subject's sex	Categorical
weight	The subject's weight	Continuous
blood_type	The subject's blood type	Categorical
current_location	The subject's location (in coordinates)	Other
num_of_siblings	The subject's number of siblings	Ordinal
happiness_score	Measure for satisfaction of the subject	Ordinal
household_income	Total income of the subject household (in thousands)	Ordinal

conversations_per_day	The number of conversations per day of the subject	Ordinal
sugar_levels	The subject's blood sugar levels	Continuous
sport_activity	The subject's sport activity level	Ordinal
symptoms	The subject's symptoms	Other
pcr_date	The subject's PCR test date	Ordinal
PCR_01	The subject's PCR test feature #1	Continuous
PCR_02	The subject's PCR test feature #2	Continuous
PCR_03	The subject's PCR test feature #3	Continuous
PCR_04	The subject's PCR test feature #4	Continuous
PCR_05	The subject's PCR test feature #5	Continuous
PCR_06	The subject's PCR test feature #6	Continuous
PCR_07	The subject's PCR test feature #7	Continuous
PCR_08	The subject's PCR test feature #8	Continuous
PCR_09	The subject's PCR test feature #9	Continuous
PCR_10	The subject's PCR test feature #10	Continuous

Q4.

It's important to use the exact same split for the dataset to training set & test set for all our analyses, because we want to ensure that we get consistent results from different data measurements and analyses on our training set, and between different runs of the script.
Also, we don't want to mix training and test data, so that we will not use the test data for the prediction.

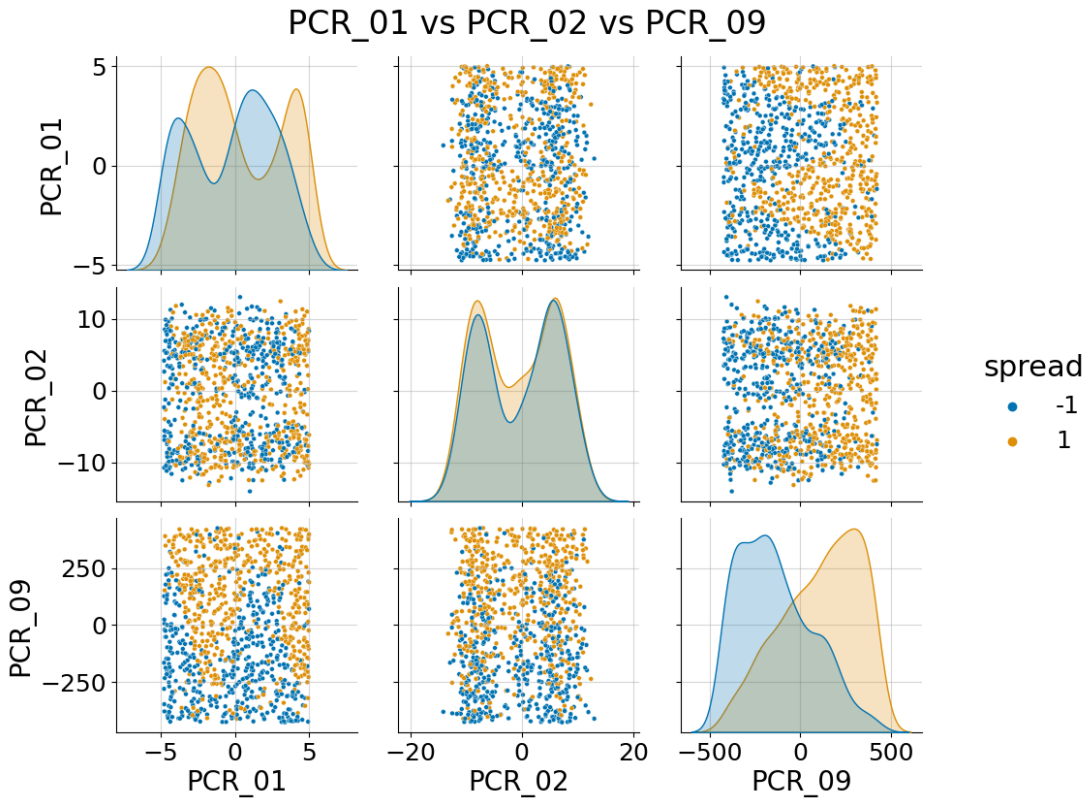
Part 2 – Warming up with k-Nearest Neighbors

Q5.

The correlation between 'spread' label and 'PCR_01','PCR_02','PCR_09' is as follows:

```
Correlation of 'spread','PCR_01' is: 0.113
Correlation of 'spread','PCR_02' is: -0.022
Correlation of 'spread','PCR_09' is: 0.513
```

Q6.



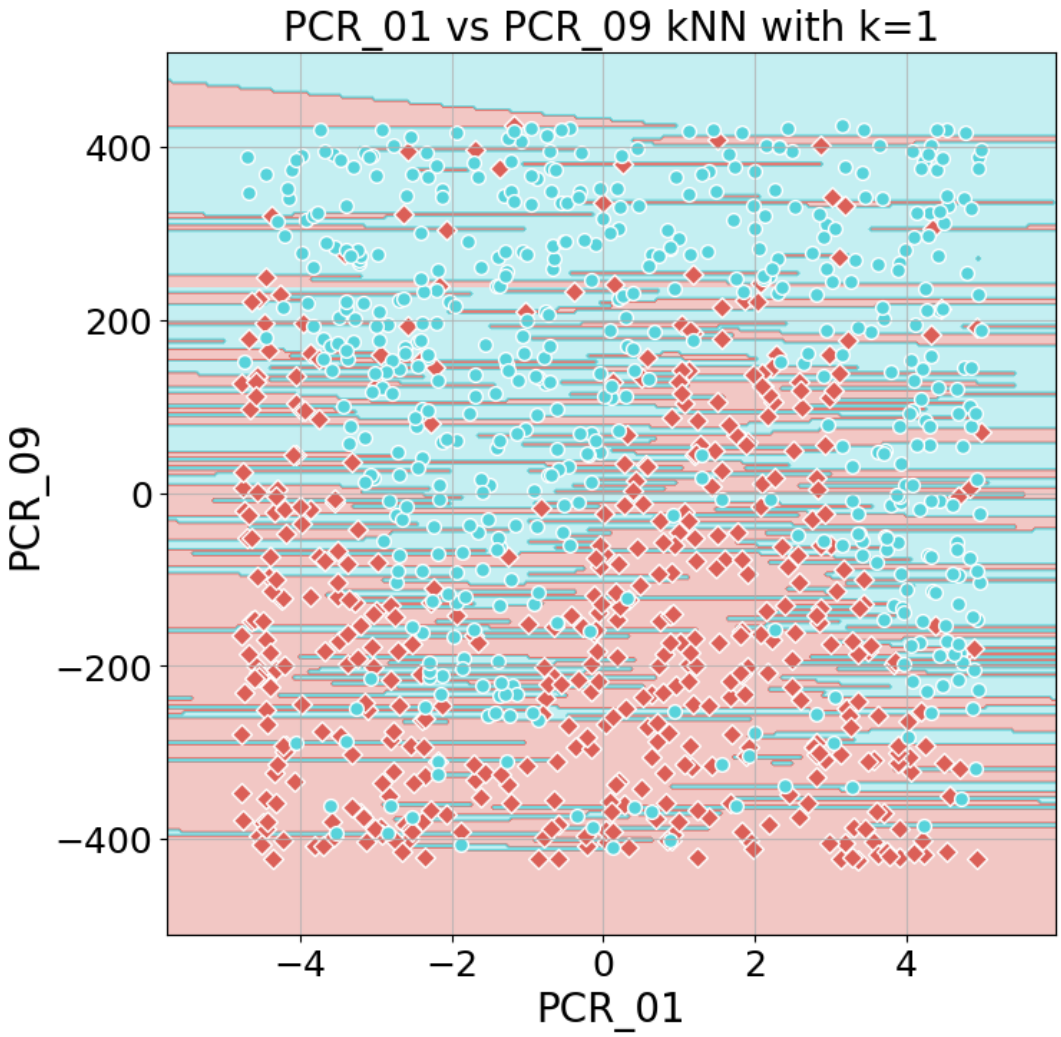
According to this figure, we can see that out of the 3 different features, 'PCR_01' & 'PCR_09' are the most useful to predict 'spread', because of the "partition" to areas according to their label (that will be useful when applying k-Nearest Neighbors methos to minimize empirical error).

Q7.

The time complexity of our prediction function applied on a single test point is:
- Computing the distance between the m datapoints with dimension d to the test point: $O(d)$ for each point, $O(md)$ total
- Finding the indexes of the k nearest neighbors (k lowest distances) using np.argpartition in $O(m)$
- Retreiving the labels of the k indexes in $O(k)$
- Summing the labels in $O(k)$ and determining its sign in $O(1)$

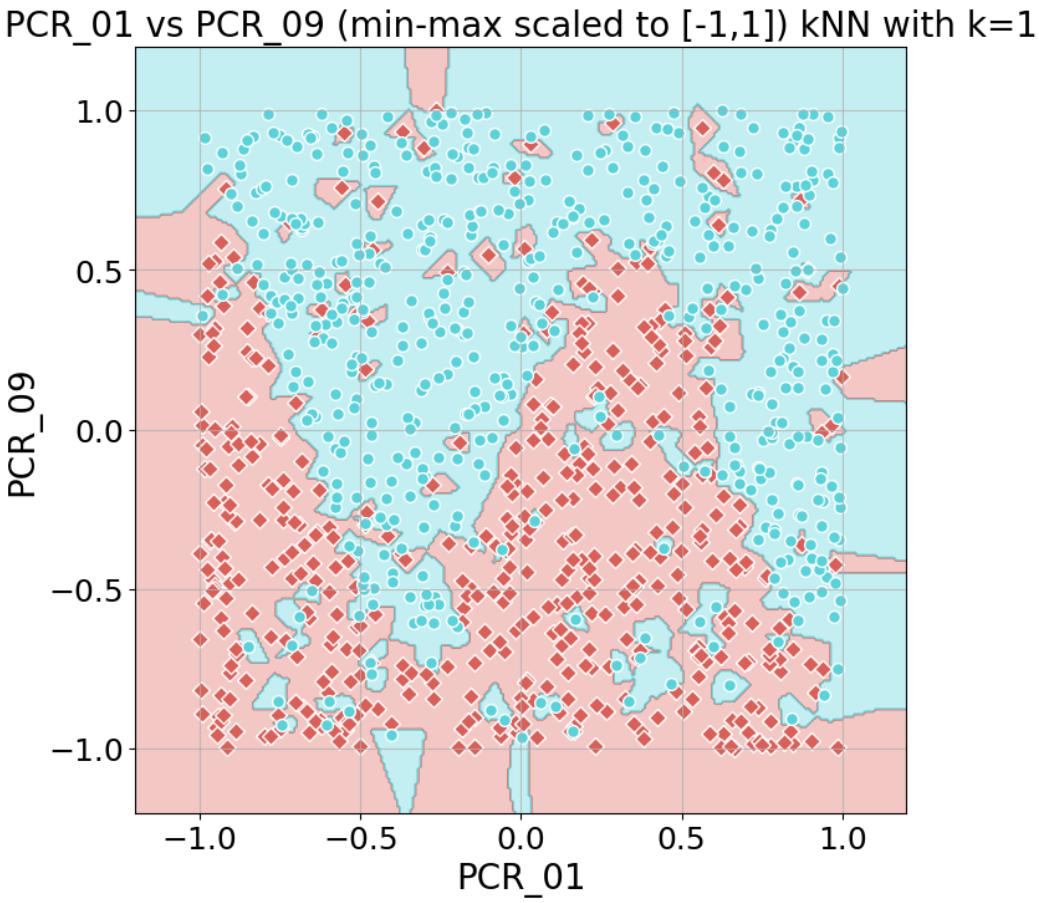
$\Rightarrow O(md + k) =_{k=O(m)} O(md)$ total

Q8.



The model predicts the training set with accuracy: 100% and the test set with accuracy: 74.8%

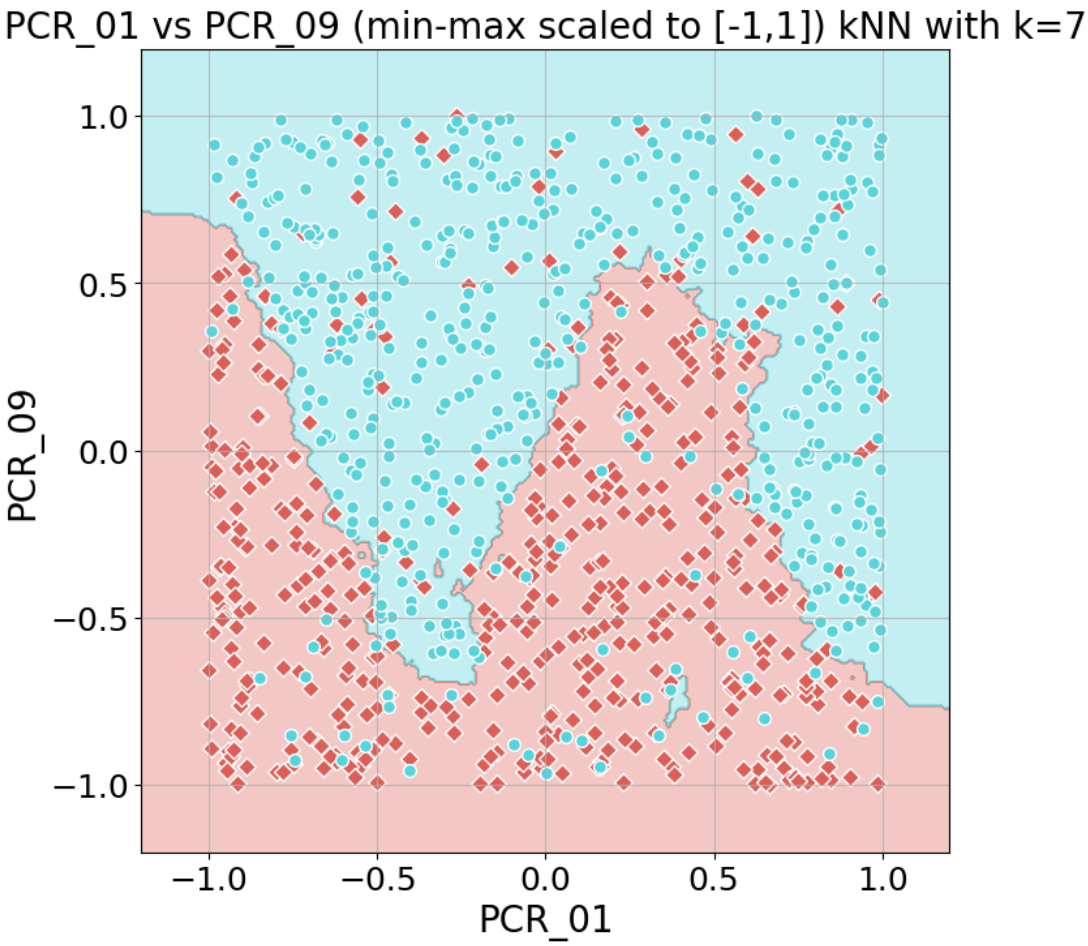
Q9.



The model predicts the training set with accuracy: 100% and the test set with accuracy: 75.6%.
We didn't change the accuracy of the training set and improved the prediction of the test set (by 0.8%).

It's important to normalize the features before using the kNN model, because different features can obtain values from different ranges, causing preferences to smaller range features when finding the nearest neighbors.
In our dataset, 'PCR_01' range is smaller than 'PCR_09' range by a factor of 100, causing adjacent points regarding 'PCR_09' appear relatively distant from points regarding 'PCR_01'.

Q10.



The model predicts the training set with accuracy: 88.2% and the test set with accuracy: 88.4%.
Compared to Q9, our predictions for the training set decreased from 100% to 88.2% (!), and our predictions to the test set increased from 75.6% to 88.4%.

The effect of k in kNN model:
For $k = 1$ we'll receive accuracy of 100% for every training set (each point is closest to itself), and for the training set we'll receive low accuracy due to overfitting (caused mostly by noise).
For $k > 1$ it's possible to receive lower accuracy than 100% on the training model, as we saw above (for example, points that we can consider as noise in the real data not necessarily share its label with its closest k neighbors). However, for the test set we're less prone to overfitting because we consider more points for each unlabeled point (but as k increase, we can suffer from underfitting).

We can clearly see in this graph that increasing k "smooths" our decision regions, and that the model will predict according to the label's general area.

Q11.

Assume a dataset with two features, $X \sim U[2,5], Y \sim \chi^2(1)$ iid.
The chi-squared distribution for some k is represented by:

$$Q \sim \chi^2(k), \;\; Q = \sum_{i=1}^{k} Z_i^2, \;\; Z_i \sim N(0,1) \; iid$$

for $k = 1$ we can denote $Y = Z^2$ for $Z \sim N(0,1)$.
Using the min-max scaling on the unbounded RV Y, we are prone to high outliers of feature Y, that will cause crowding in the lower bound for the rest of the dataset.
More formally, denote x' the normalized value of x (some point in Y dataset).

$$x' = a + \frac{(x - \min(Y))(b - a)}{\max(Y) - \min(Y)}$$

where $[a, b]$ is our normalized range.
For a single outlier $\tilde{x} \in Y$ s.t. $\tilde{x} \to \infty$ we get $x' \to 0$ for $x \in Y \setminus \{\tilde{x}\}$,
causing inaccurate distance measurements between the features (and for the same feature) as described in Q9, and causing predictions inaccuracies for small values of Y feature, due to "noises".

<u>Part 3 – Data Exploration</u>

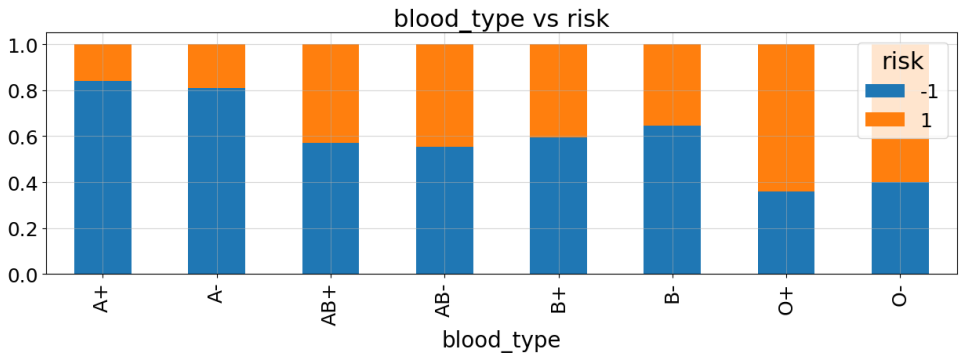
Q12.

The difference categories of blood_type feature are

$$A+, A-, B+, B-, AB+, AB-, O+, O-$$

therefore we need 8 Boolean features for one-hot-encoding.

Q13.



We can partition the blood_type into three different bloud groups:

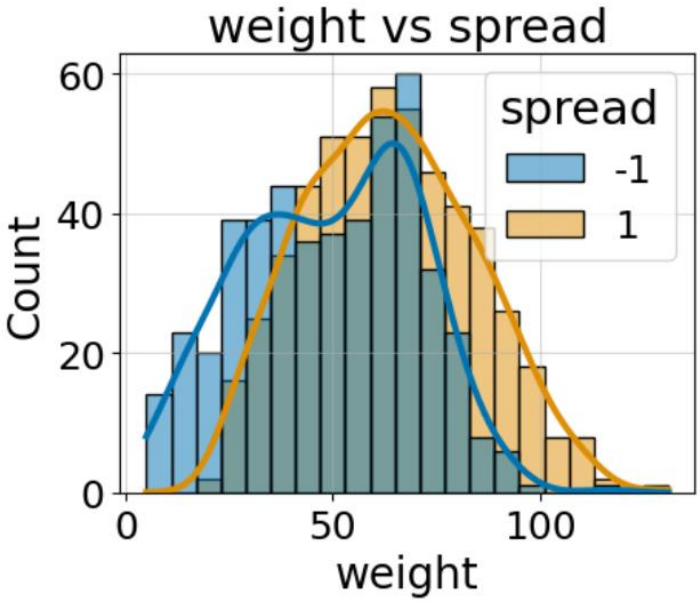
$$\{A+, A-\}, \{AB+, AB-, B+, B-\}, \{O+, O-\}$$

because for each group we obtain (approximately) the same ratio for the 'risk' label.

Q14.

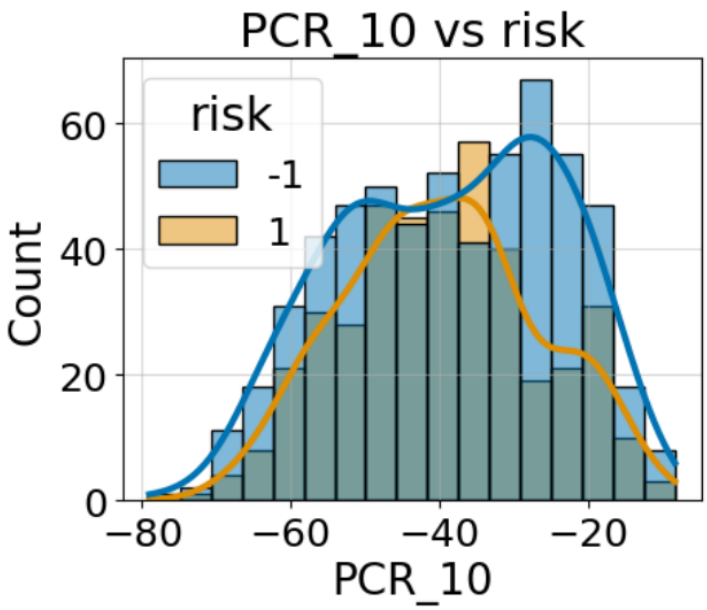
We can extract information from the symptoms feature by creating 5 new Boolean features for each symptom listed in symptoms feature, and for every datapoint that lists a certain symptom, we'll encode it as 1 in its corresponding columns and 0 for the rest of the symptoms' columns.

Q15.



The above univariate plot of the feature 'weight' is informative, because we can see different distributions over our samples, which are not aligned (higher weight is more likely to labeled spread=1, and lower weight is more likely to labeled spread=-1).
That can be helpful to evaluate a sample's 'spread' label by its 'weight' value.

Q16.

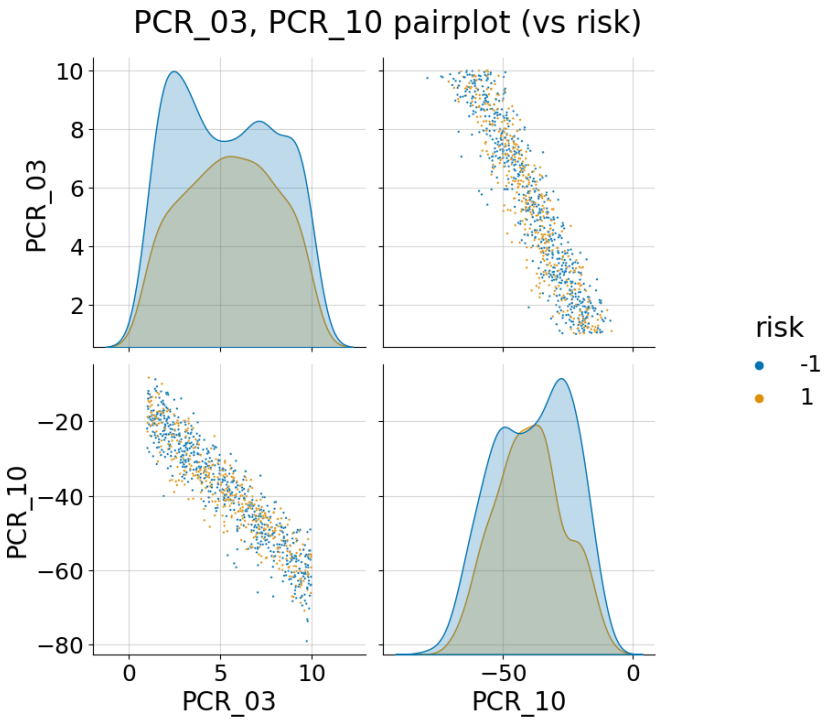


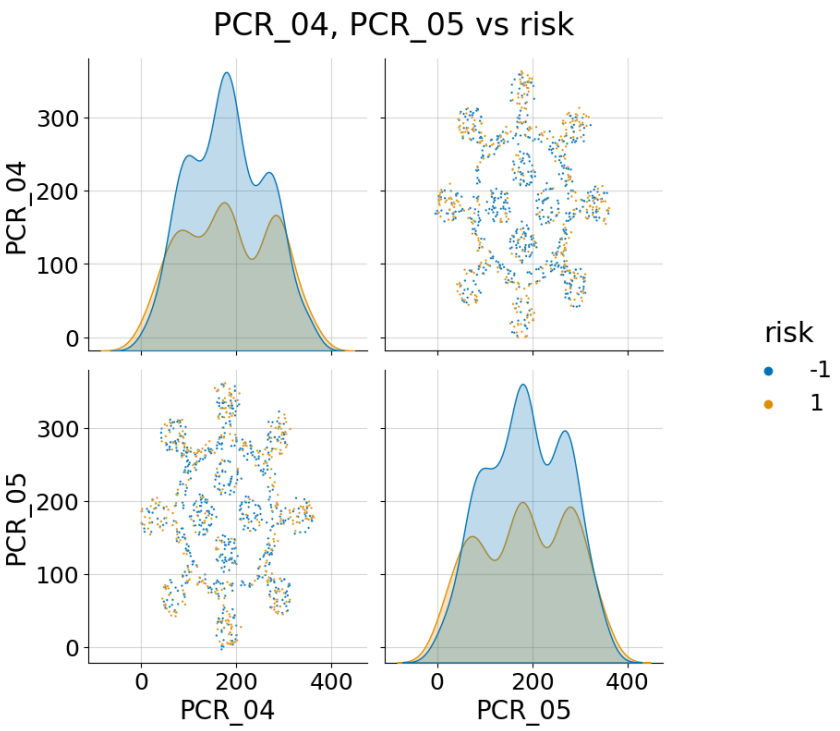
The above univariate plot of the feature 'PCR_10' is informative, because we can see different distributions over our samples, which are not aligned (most subjects in the training set are labeled spread=-1, therefore joint common values of 'PCR_10' are more likely to label risk=1).
That can be helpful to evaluate a sample's 'risk' label by its 'PCR_10' value.

Q17.

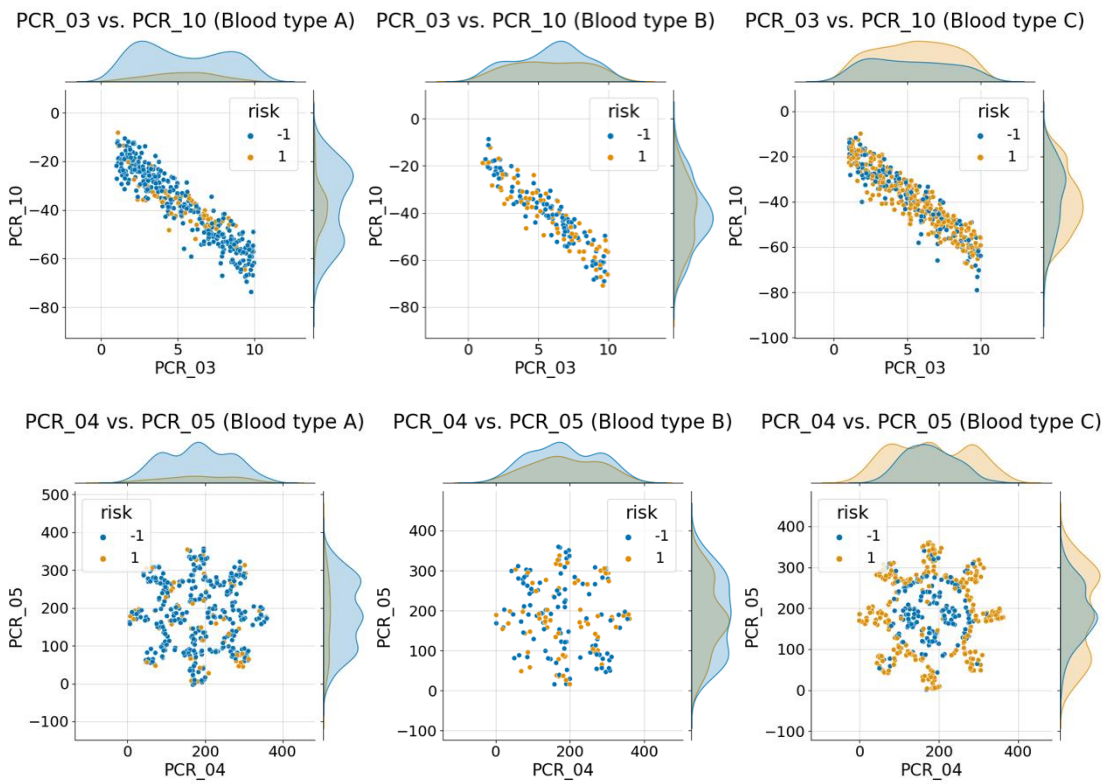
The two pairs of features we choose formed the "interesting" structures were 'PCR_04','PCR_05' and 'PCR_03','PCR_10'.
However, both couples we selected does not explain the risk label by itself (because the data isn't separable regarding those features).

'PCR_03','PCR_10' have big correlation factor (-0.929807), and 'PCR_04','PCR_05' format a suspiciously interesting scattering.





Q18.



Indeed, one of the plots seems informative for predicting risk label ('PCR_04','PCR_05' with blood type C)

Q19.

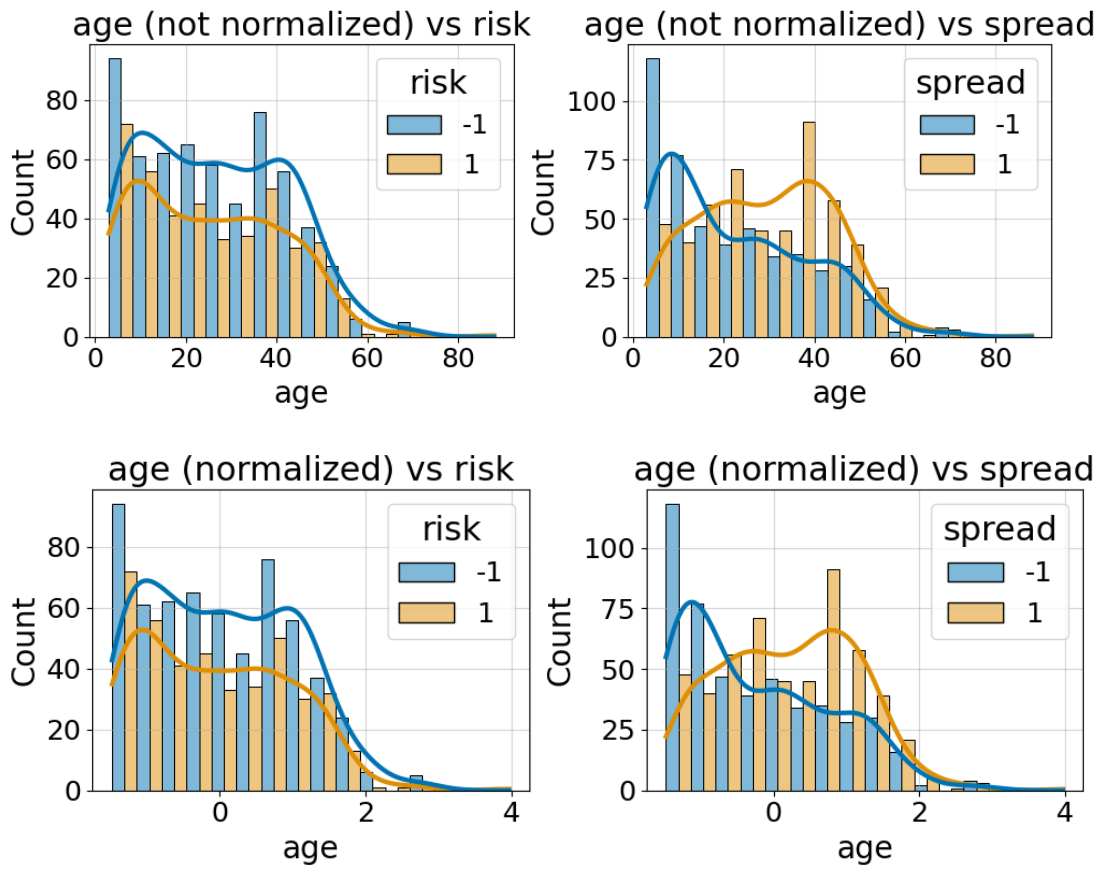
kNN: The model will help predict the risk target variable for some of the graphs but not for all of them. kNN is usually helpful when samples with the same label are clustered.
In our case: PCR_03 vs PCR_10 for blood type A or for PCR_04 vs PCR_05 for blood type C, the model's accuracy will be relatively high, but for PCR_03 vs PCR_10 for blood type C the samples are intertwined so kNN is not optimal for this case.

Decision Trees: The decision trees model will also work for some of the graphs above and won't work for others, with the same reasoning, this model works well when the samples are clustered by their risk label.

Linear models: The linear models won't help us predict the risk target variable for any of the graphs above.
For linear models (with no mapping) to be effective, there needs to be separation by hyperplane between the two groups of risk labels (that doesn't exist for the graphs above).

<u>Part 4 – More Data Normalization</u>

Q20.



Part 5 – Feature Selection

Q21.

Let $d_1 \in \mathbb{N}$ be the number of features in the dataset and assume that the time needed to train a model (regardless of its number of features) is $O(1)$.
In forward feature selection, we start with an empty set of features, and in each iteration we add a feature that has not been picked yet (that contributes the most to the accuracy of the model).
Therefore, in the first iteration we train all the models that contains 1 feature - d_1 options (in d_1 "time"). In the second iteration we train all the models with 1 additional feature from the remaining $d_1 - 1$ features and so forth. In total, the time complexity:

$$d_1 + (d_1 - 1) + \cdots + (d_1 - d_2 + 1) = \sum_{i=0}^{d_2-1} d_1 - i = d_1 d_2 - \frac{(d_2 - 1)(d_2 - 2)}{2}$$

$$\Rightarrow \quad O(d_1 d_2)$$

And in backwards feature selection, we start with a full set of all d_1 features and removing them until we are left with d_2 features.
At starting point, we have d_1 models. Next, $d_1 - 1$ and so forth, until we choose between $d_2 + 1$ features which should we remove. In total, the time complexity:

$$d_1 + (d_1 - 1) + \cdots + (d_2 + 1) = \sum_{i=0}^{d_1-d_2-1} d_1 - i = d_1(d_1 - d_2 - 1) - \frac{(d_1 - d_2 - 1)(d_1 - d_2)}{2}$$

$$\Rightarrow \quad O\big(d_1(d_1 - d_2)\big)$$

Q22.

Sample/feature	x[1]	x[2]	x[3]	y
#1	1	-1	1	1
#2	1	1	1	1
#3	-1	1	1	-1
#4	-1	1	1	-1
#5	-1	-1	-1	-1
#6	-1	-1	-1	-1

In forward selection we will select x[1] and in backward selection we will select x[2]/x[3].

Before this update, we couldn't find a solution, so we wrote a python script that iterates over all the possible options for 4 & 5 points in the dataset (and each point can be any vector over $\{\pm 1\}^4$. Here is the snippet we used for iterating over a dataset with 5 points:

```python
with warnings.catch_warnings():
    warnings.filterwarnings("ignore", category=UndefinedMetricWarning)

    count = 0
    # check all 4 points
    for a1p1 in range(-1, 2, 2):
        for a2p1 in range(-1, 2, 2):
            for a3p1 in range(-1, 2, 2):
                for labelp1 in range(-1, 2, 2):
                    point1 = [a1p1, a2p1, a3p1, labelp1]

                    for a1p2 in range(-1, 2, 2):
                        for a2p2 in range(-1, 2, 2):
                            for a3p2 in range(-1, 2, 2):
                                for labelp2 in range(-1, 2, 2):
                                    point2 = [a1p2, a2p2, a3p2, labelp2]

                                    for a1p3 in range(-1, 2, 2):
                                        for a2p3 in range(-1, 2, 2):
                                            for a3p3 in range(-1, 2, 2):
                                                for labelp3 in range(-1, 2, 2):
                                                    point3 = [a1p3, a2p3, a3p3, labelp3]

                                                    for a1p4 in range(-1, 2, 2):
                                                        for a2p4 in range(-1, 2, 2):
                                                            for a3p4 in range(-1, 2, 2):
                                                                for labelp4 in range(-1, 2, 2):
                                                                    point4 = [a1p4, a2p4, a3p4, labelp4]

                                                                    df = pd.DataFrame([point1, point2, point3, point4])
                                                                    reg = linear_model.LinearRegression()
                                                                    sfs_backward = SequentialFeatureSelector(reg,
                                                                                                             n_features_to_select=1,
                                                                                                             direction="backward",
                                                                                                             cv=4)
                                                                    sfs_forward = SequentialFeatureSelector(reg,
                                                                                                            n_features_to_select=1,
                                                                                                            direction="forward",
                                                                                                            cv=4)
                                                                    X = df.iloc[:, :-1]  # Features (first three columns)
                                                                    y = df.iloc[:, -1]  # Target (last column)
                                                                    sfs_backward.fit(X, y)
                                                                    sfs_forward.fit(X, y)
                                                                    fw = sfs_forward.get_feature_names_out()
                                                                    bw = sfs_backward.get_feature_names_out()
                                                                    if fw != bw:
                                                                        count += 1
                                                                        print(df)
                                                                        print("forward: ", fw)
                                                                        print("backward: ", bw)
print(count)
```

Q23.

The three features that were selected using the Sequential Feature Selection are: 'weight', 'PCR_01', 'PCR_09' (that should predict well the 'spread' label), which we found in Q6 & Q15.

Q24.

Generally, it's important to normalize the data before the performing sequential feature selection, because for the nearest neighbor's classifier we want to use a metric which doesn't allow feature preferences (for example, when a feature has a relatively small value interval than a different feature, as we saw on Q9).

However, this depends on this specific learning algorithm. For a different learning algorithm, it's possible that the normalized step isn't necessary. For example, for a decision tree (using the ID3 algorithm), we select each feature at a time and apply a rule, without considering different features and their scales.

<u>Part 6 – Data Preparation Pipeline</u>

Q25.

Feature Name	Keep	New	Normalization Method	Explanation
patient_id	V	X	---	The patient ID doesn't add any data so we decided not to normalize it.
age	V	X	Standard	This feature is prone to outliers (by its distribution), which standard scaler can handle well
sex	V	X	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
weight	V	X	Standard	This feature has a similar distribution to a normal RV, which standard scaler can handle well
num_of_siblings	V	X	Standard	This feature was normalized using the standard scaler to be durable to outliers (as we can see its distribution)
happiness_score	V	X	Standard	This feature was normalized using the standard scaler to be durable to outliers (as we can see its distribution)
household_income	V	X	Standard	This feature was normalized using the standard scaler to be durable to outliers (as we can see its distribution)
conversations_per_day	V	X	Standard	This feature was normalized using the standard scaler to be durable to outliers (as we can see its distribution)
sugar_levels	V	X	Standard	This feature has a similar distribution to a normal RV, which standard scaler can handle well

sport_activity	V	X	Standard	This feature was normalized using the standard scaler to be durable to outliers (as we can see its distribution)
pcr_date	X	X	---	We turned this feature into pcr_date_timestamp, in order to be left with numerical values only.
pcr_date_timestamp	V	V	Standard	A new feature that represents the timestamp of the PCR test date in seconds since epoch, scaled with the standard scaler to prevent unbounded range deformation of timestamps.
PCR_01	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_02	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_03	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_04	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_05	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_06	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)

PCR_07	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_08	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_09	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
PCR_10	V	X	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
blood_type	X	X	---	In Q13 we were asked to split this feature into three Boolean features indicating the subjects blood type.
group_1_blood_type	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
group_2_blood_type	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
group_3_blood_type	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
symptoms	X	X	---	We transformed the symptoms feature into 5 other features (one for each symptom) to be left with numerical values only. Normalized to [-1,1] to be in the same value range of the rest of the data.

sore_throat	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
cough	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
shortness_of_breath	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
fever	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
smell_loss	V	V	Min-Max	This binary feature is informative. We normalized it to fit its range to the different features
current_location	X	X	---	We transformed the feature into coordinate_X and coordinate_Y to be left with numerical values only.
coordinate_X	V	V	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)
coordinate_Y	V	V	Min-Max	Scaled using min-max scaler on bounded range values (we saw no outliers in its distribution)