**RPPA\_model**

**Creating a proper dataset**

**01.**

We downloaded 3 datasets for protein type from TCGA.

1. KIRC\_RPPA.data 🡪 Clear cell
2. KIRP\_RPPA.data 🡪 Papilary cell
3. KICH\_RPPA.data 🡪 Chromophobe

**02.**

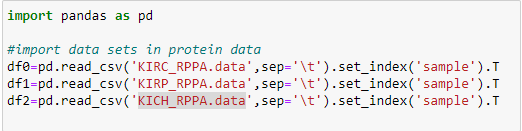
In the datasets, columns contained patients and rows contained the protein types (features). Since in TCGA, they divided the datasets for each subtype, in each dataset there were not a target row. So, we added a target row as ‘Subtype’ for each dataset with some integer values. Values can define as.

* ‘0’: KIRC
* ‘1’: KIRP
* ‘2’: KICH

**Dataset initialization**

**01.**

Then we started to create our models in python 3 environment. Here we imported all the datasets.

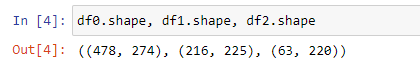


Reason for using set\_index is, if we did not define the index, by default Pandas will create an index for the DataFrame. It makes things a little more confusing, because by default the “index” is just the range of numbers starting at 0.

Here we swap the rows and columns using *transpose()* method to get the features and label into columns and patients into rows.

**02.**

Here are the shapes of each dataset.



We can clearly see that the number of features is not equal.

**Data Preprocessing**

**1. Data Integration**

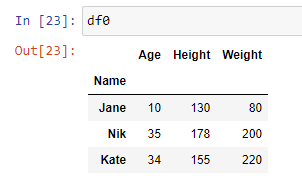
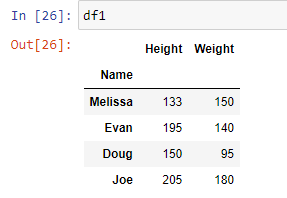
**01.**

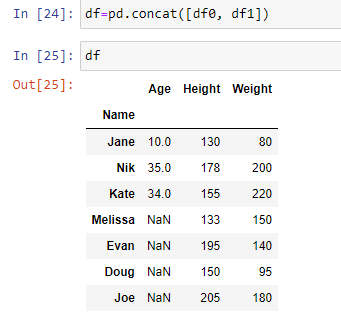
Now we need to combine these 3 datasets. Since the features are not exactly same in 3 datasets, we could not use merge method. So we used *pandas.concat* method for combine the datasets.



In concat, it simply merges the datasets if the features (column names) are same and if not, it just adds the column. By adding columns which are not in other datasets, makes some ‘NULL’ values (missing values) to the dataset.

**Ex**: How *concat* works (note that these data not related to the protein data)



* Here you can see that in concat order of the columns are not mattered. And if a column is not in both data frame, combined data frame has some missing values.

(Note: Even here used ‘df0’,’df1’ and ‘df’ for data frame naming it is differed from our main project. This is just a tested code used for show what happed in the concat method)

**02.**

Here is the shape of new data frame (Protein data).



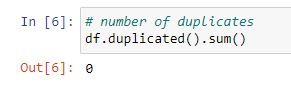
Here showing 757 records. It just the combination of df0, df1 and df2 (478+216+63=757).

Here showing 279 columns. Since in 3 datasets there were more similar columns and some different columns for each other, number of columns are differed from df0, df1, df2 and df. We actually need to remove non similar columns for further process. Since those non similar columns have missing values in combined datasets, we can easily check those values in the prepossessing steps.

**2. Data Cleaning**

**01.**

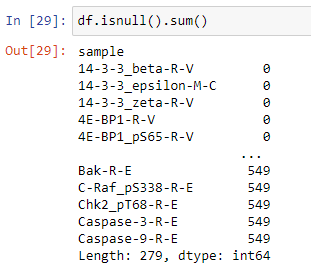
First, we try to find the number of duplicates in dataset.



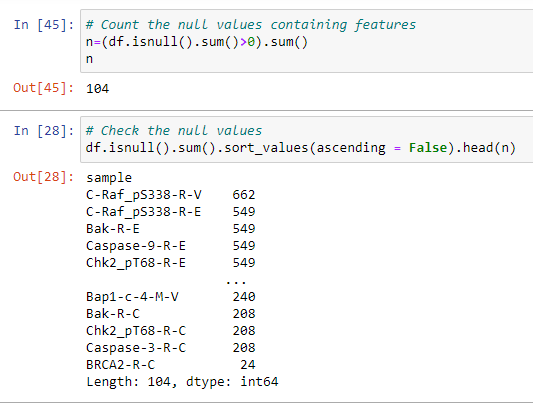
There were no any duplicates in the dataset.

**02.**

Then we check for missing values.



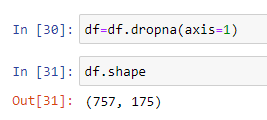
By this visualization it is difficult to take a decision about null values. So, we count the features which containing null values and try to visualize them.



As you can see, we found that there 104 features contain at least one null value. And we try to see those 104 features and their null value count. As we can see it is in the range of 24 – 662.

**03.**

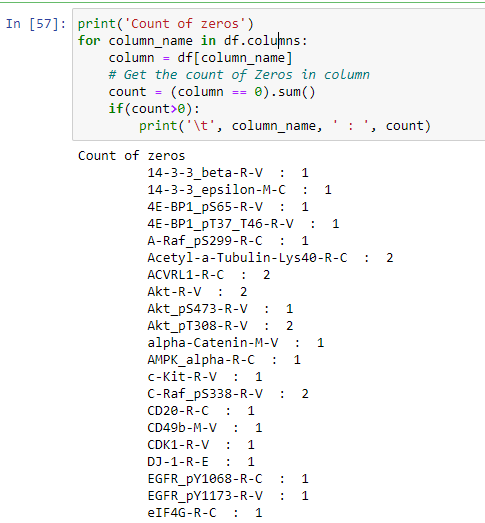
Since we had altogether 279 features, we decided to remove those 104 null values containing features.



Then we got 175 usable features (+ labels) up to now. (279-104=175)

**04.**

Then we count the zero values in the features.



Count of zeros

14-3-3\_beta-R-V : 1

14-3-3\_epsilon-M-C : 1

4E-BP1\_pS65-R-V : 1

4E-BP1\_pT37\_T46-R-V : 1

A-Raf\_pS299-R-C : 1

Acetyl-a-Tubulin-Lys40-R-C : 2

ACVRL1-R-C : 2

Akt-R-V : 2

Akt\_pS473-R-V : 1

Akt\_pT308-R-V : 2

alpha-Catenin-M-V : 1

AMPK\_alpha-R-C : 1

c-Kit-R-V : 1

C-Raf\_pS338-R-V : 2

CD20-R-C : 1

CD49b-M-V : 1

CDK1-R-V : 1

DJ-1-R-E : 1

EGFR\_pY1068-R-C : 1

EGFR\_pY1173-R-V : 1

eIF4G-R-C : 1

ER-alpha-R-V : 1

ERK2-R-E : 1

ETS-1-R-V : 1

FOXO3a-R-C : 1

FOXO3a\_pS318\_S321-R-C : 1

GATA3-M-V : 2

HER2\_pY1248-R-C : 1

HER3-R-V : 1

Heregulin-R-V : 1

INPP4B-R-V : 1

MEK1\_pS217\_S221-R-V : 2

Mre11-R-C : 1

MSH2-M-V : 1

N-Ras-M-V : 1

NDRG1\_pT346-R-V : 1

Notch1-R-V : 1

p21-R-V : 1

p27-R-V : 1

p27\_pT157-R-C : 1

p62-LCK-ligand-M-C : 1

p90RSK\_pT359\_S363-R-C : 1

PAI-1-M-E : 1

PCNA-M-C : 1

PDK1-R-V : 1

PEA15-R-V : 1

PI3K-p85-R-V : 1

PKC-alpha\_pS657-R-C : 1

PKC-delta\_pS664-R-V : 1

PKC-pan\_BetaII\_pS660-R-V : 1

PR-R-V : 1

Rab25-R-V : 1

Rad50-M-V : 2

Raptor-R-V : 1

Rictor\_pT1135-R-V : 2

SETD2-R-E : 1

SF2-M-V : 1

Smad4-M-V : 1

Snail-M-E : 1

Src-M-V : 1

Stathmin-R-V : 2

EPPK1-M-E : 1

XBP1-G-C : 1

c-Abl-R-V : 1

Caspase-3-R-C : 1

CD26-R-V : 1

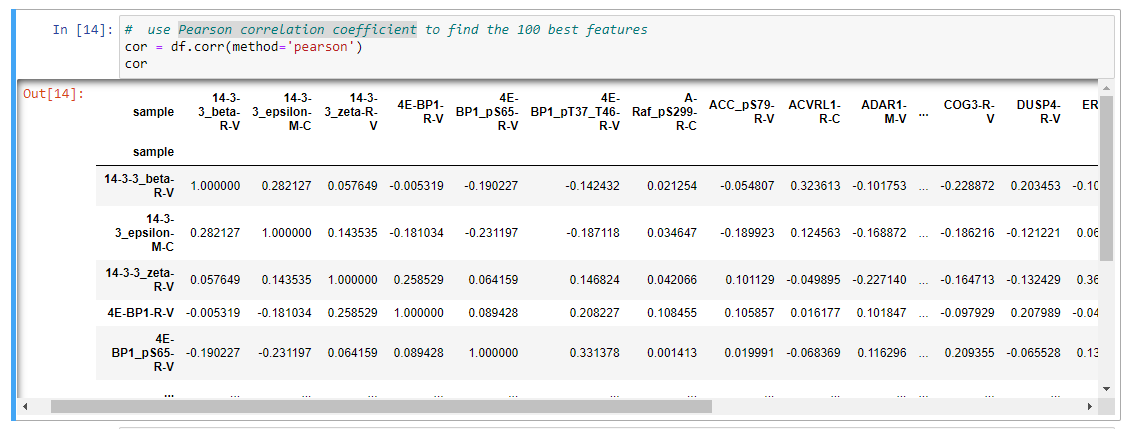
Subtype : 478

It seems that all the features containing very neglectable zero values. So, we decided to keep them as it is. Here *Subtype* is our label and it’s zero value has a meaning (explained earlier). So, no need to consider about it also.

**3. Feature Selection - Pearson correlation coefficient**

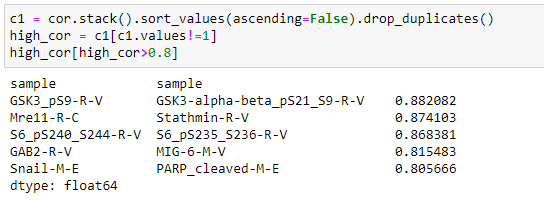
**1.**

Get the correlation coefficients in dataset



**2.**

Then first we get the high correlated (more than 0.8) features with each other.



Since these values are high, it means these are somewhat linearly dependent with other features. hence those effect on the dependent variable is almost same. So we can drop one of them.

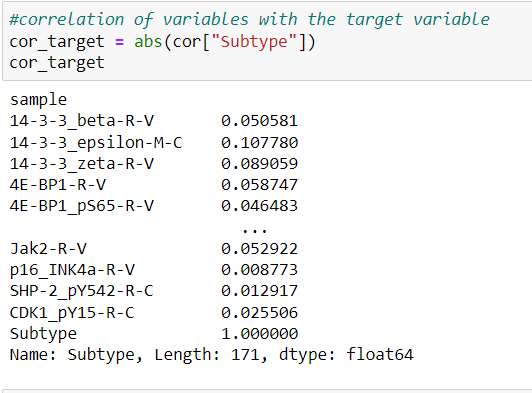


**3.**

Then again introduced Pearson correlation, since the data frame was changed.



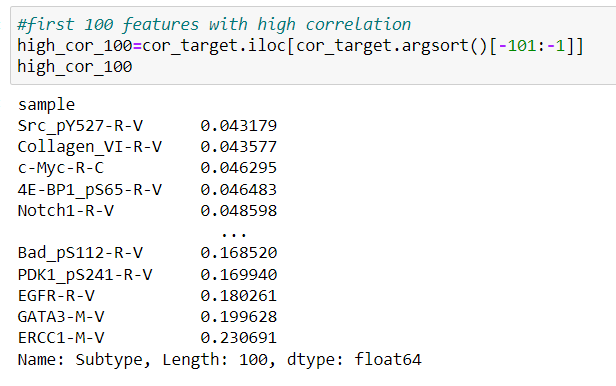
Then took the absolute correlation of features with the target variable.



Here we planned to test for best 100,50,20 features first. Considering changing of the accuracy we hope to change the selecting number of features.

**4.**

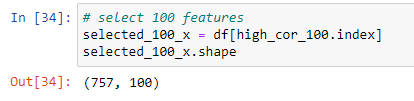
So first we got the best 100 features for further process.



Here features are taken in ascending order of coefficient. So, we can see that it is in the range of 0.043179 to 0.230691. Actually, this is a low correlation range.

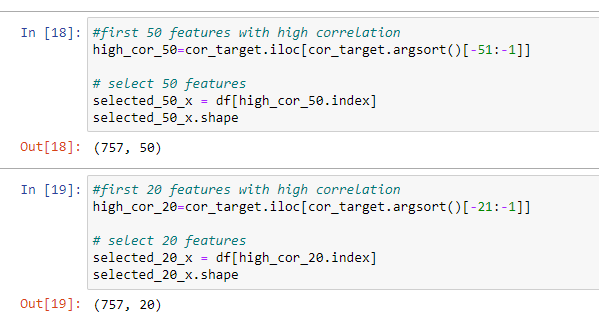
**5.**

Then we created the feature data frame with 100 features.



**6.**

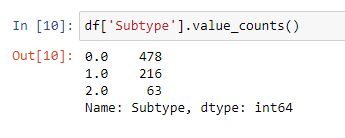
Then we created data frames for best 50 and 20 features also.



**4. Dataset balancing**

**01.**

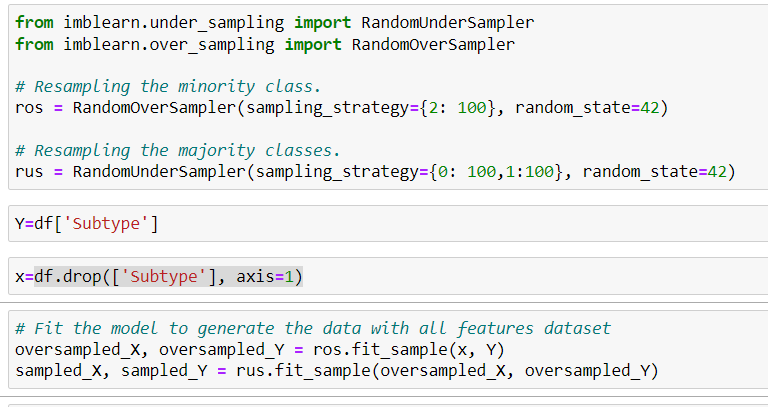
Then we viewed the value count of each subtype.

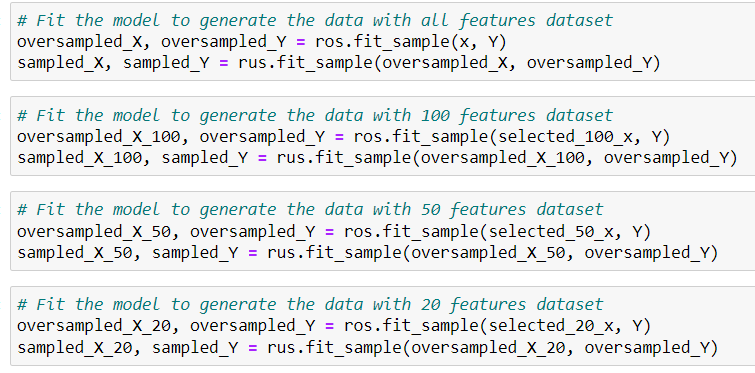


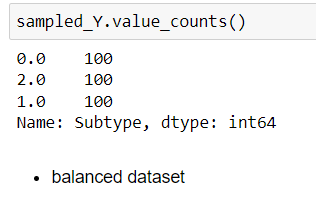
As we can see the dataset is unbalance over each subtype. So, we need to handle these unbalancing.

**02.**

We handle the unbalancing of recodes using up sampling and under sampling to get a fair and balanced dataset.



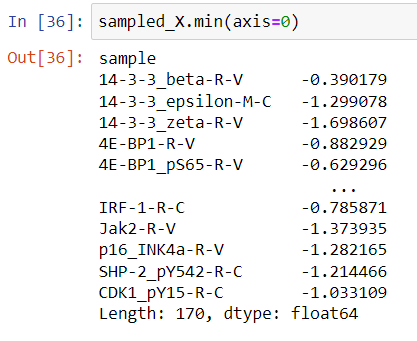
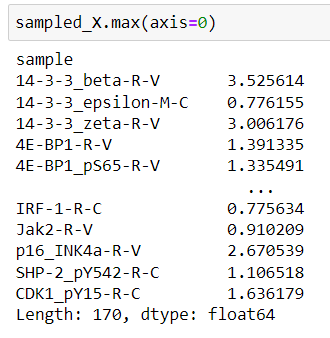




So, we get a balanced dataset with 300 records.

**5. Feature Scaling**

Here I checked whether the dataset need a scaling. For that I just print the max and min values of each feature.

As we can see the data set has different but small range of values.

**Test SVM**

**01.**

First, we try to find the best kernel to use for further process. Here we used cross validation for this test.

* **Accuracy SVM with rbf kernel: 0.88 (+/- 0.06)**
* Accuracy SVM with poly kernel: 0.47 (+/- 0.08)
* Accuracy SVM with linear kernel: 0.82 (+/- 0.13)
* Accuracy SVM with sigmoid kernel: 0.64 (+/- 0.05)

Rbf kernel performed well.

**02.**

Then, we find the accuracy without data transformation. Then we tried both standardization and normalization. So, we observed that standardization performed well with SVM. Here also used cross validation for this test.

* Accuracy SVM without transformation: 0.88 (+/- 0.06)
* **Accuracy SVM with Standardization: 0.93 (+/- 0.06)**
* Accuracy SVM with Normalization: 0.89 (+/- 0.06)

**03.**

Then we did the standardization for other datasets, which we created with different number of features (20,50,100) and evaluate those models using SVM by cross validations.

* Accuracy SVM using best 20 features: 0.78 (+/- 0.12)
* Accuracy SVM using best 50 features: 0.88 (+/- 0.07)
* Accuracy SVM using best 100 features: 0.90 (+/- 0.04)

While increasing the number of features we got higher accuracy. So, we tried more than 100 number of features also. Then when we using 150 features, we got same accuracy as when we tried all features.

* Accuracy SVM using best 120 features: 0.91 (+/- 0.06)
* Accuracy SVM using best 140 features: 0.94 (+/- 0.04)
* **Accuracy SVM using best 150 features: 0.93 (+/- 0.03)**

**Test RF**

**01.**

First, we try different combinations of parameters with the help of GridSearchCV. Here we tested best 3 parameters in RF and took the best combination.

* Best parameters are:
  + 'criterion': 'gini',
  + 'max\_depth': 8,
  + 'n\_estimators': 1000

Then we did the cross validation.

* Accuracy RF with best\_params: 0.97 (+/- 0.03)

**02.**

Then, we find the accuracy without data transformation. After that we tried both standardization and normalization. But we observed all the performance were same. Reason is, **RF does not need a data transformation since it is a Tree-based model and they are not based on the distance where features have an effect on one another**.

* Accuracy RF without transformation: 0.97 (+/- 0.03)
* Accuracy RF with Standardization: 0.97 (+/- 0.03)
* Accuracy SVM with Normalization: 0.97 (+/- 0.03)

**03.**

Then we test the model for other datasets, which we created with different number of features (20, 50, 100, 120, 140, 150) and evaluate those models’ using RF by cross validations.

* Accuracy RF using best 20 features: 0.85 (+/- 0.12)
* Accuracy RF using best 50 features: 0.92 (+/- 0.05)
* Accuracy RF using best 100 features: 0.94 (+/- 0.05)
* Accuracy RF using best 120 features: 0.95 (+/- 0.03)
* Accuracy RF using best 140 features: 0.96 (+/- 0.02)
* Accuracy RF using best 150 features: 0.96 (+/- 0.02)

While increasing the number of features we got higher accuracy. But we did not reach the accuracy, that we got from all feature (170) dataset. Here, we tried more than 160 features also. Then also we could not reach that performance.

* Accuracy RF using best 160 features: 0.96 (+/- 0.01)

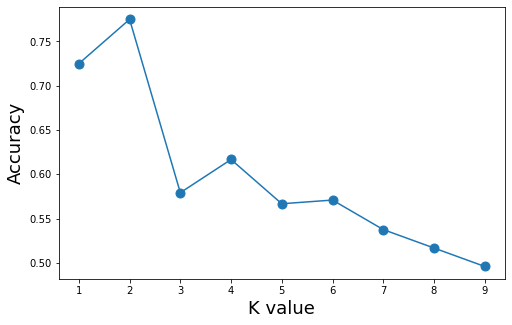
So best performance is with **all features** and cross validation score is:

**0.97 (+/- 0.03)**

**Test KNN**

**01.**

First, we try to find the best k value to use for further process. Here accuracy is got from cross validation’s (cv=5) mean value.



Here k=2 is showing the best accuracy. But since we have 3 classes, we choose **4** as the optimal value for k value.

**02.**

Then, we find the accuracy without data transformation. After that we tried both standardization and normalization. Here normalization performed well with KNN here. Here also for testing we used cross validation for this selection.

* Accuracy KNN without transformation: 0.68 (+/- 0.12)
* Accuracy KNN with Standardization: 0.76 (+/- 0.14)
* **Accuracy KNN with Normalization: 0.76 (+/- 0.12)**

**03.**

Then we did the normalization for other datasets, which we created with different number of features (20,50,100) and evaluate those models using KNN by cross validations.

* Accuracy KNN using best 20 features: 0.73 (+/- 0.11)
* Accuracy KNN using best 50 features: 0.81 (+/- 0.11)
* Accuracy KNN using best 100 features: 0.78 (+/- 0.11)

Here performance showed a monotonic variation when increasing the number of features. We got higher accuracy when choosing best 50 features. So, we tried 60 and 40 features also. Then when we using 40 features, we got better accuracy than accuracy 50 features.

* Accuracy KNN using best 60 features: 0.79 (+/- 0.05)
* **Accuracy KNN using best 40 features: 0.81 (+/- 0.09)**

Because of the monotonic variation we tried best 30 features also.

* Accuracy KNN using best 30 features: 0.79 (+/- 0.07)

Considering above results we can see that for KNN (k=4), we got the best accuracy with 40 features.

**Overall Results – Protein Data**

|  |  |  |  |
| --- | --- | --- | --- |
| **Number of features** | **Cross Validation Score** | | |
| **SVM (Standardized)** | **RF** | **KNN (Normalized with min max scaler)** |
| All (170) | 0.93 (+/- 0.06) | 0.97 (+/- 0.03) | 0.76 (+/- 0.12) |
| 160 | - | 0.96 (+/- 0.01) | - |
| 150 | 0.93 (+/- 0.03) | 0.96 (+/- 0.02) | - |
| 140 | 0.94 (+/- 0.04) | 0.96 (+/- 0.02) | - |
| 120 | 0.91 (+/- 0.06) | 0.95 (+/- 0.03) | - |
| 100 | 0.90 (+/- 0.04) | 0.94 (+/- 0.05) | 0.78 (+/- 0.11) |
| 60 | - | - | 0.79 (+/- 0.05) |
| 50 | 0.88 (+/- 0.07) | 0.92 (+/- 0.05) | 0.81 (+/- 0.11) |
| 40 | - | - | 0.81 (+/- 0.09) |
| 30 | - | - | 0.79 (+/- 0.07) |
| 20 | 0.78 (+/- 0.12) | 0.85 (+/- 0.12) | 0.73 (+/- 0.11) |

* Best accuracy showing RF with 170 features. Used parameters are:
  + 'criterion': 'Gini',
  + 'max\_depth': 8,
  + 'n\_estimators': 1000