Dataset Analyses Report

Prakanshu Sahu • 20.04.2021

Overview

Recent Progress

- Analysed the dataset
- Classifier for Gender feature
- Compaed the chloride feature of different age group.

- Interesting fact about the dataset

- Practical Use of dataset

Progress - Dataset Features

Parameters

- WBC - The **normal** number of **WBCs** in the blood is 4,500 to 11,000 **WBCs** per microliter (4.5 to 11.0 × 10^9/L). **Normal value ranges** may vary slightly among different labs.
- T4- Thyroid produces a hormone called thyroxine, which is known as T4. A **normal** Total **T4 level** in adults **ranges** from 5.0 to 12.0μg/dL
- Sodium - A **normal** blood **sodium level** is between 135 and 145 milliequivalents per liter (mEq/L).

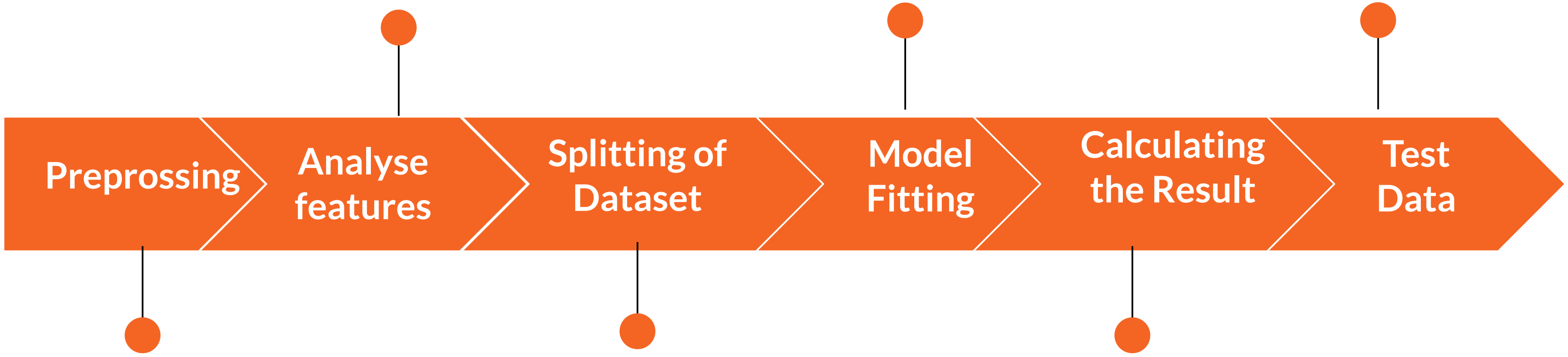
Parameters

- Chloride - A typical **normal range** is 96 to 106 milliequivalents per liter (mEq/L)
- SGPT - The **SGPT normal range** is about 7 to 56 units per liter of blood serum

Dataset Statistics

Number of variables	9
Number of observations	10000
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows (%)	0.0%
Total size in memory	3.6 MiB

Classifier for Gender Feature



Removing unnecessary features from the Dataset.

Analysing features using pandas-profiling .

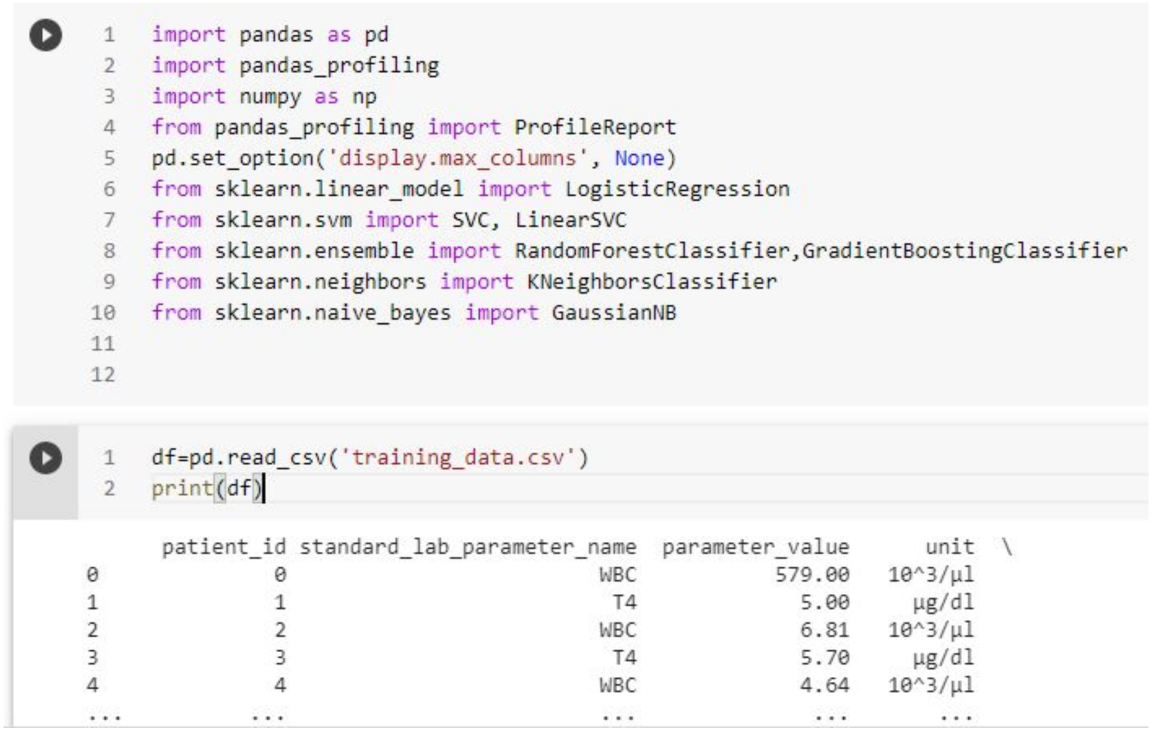
Splitting the dataset into train dataset and validation set .

Using different models like Logistic Regression, SVM , gradient boosting, Random Forest classifier.

Calculating accuracy of all the models and selecting the best model.

Applying the test dataset to predict the gender features

Importing Libraries & Loading Train Dataset

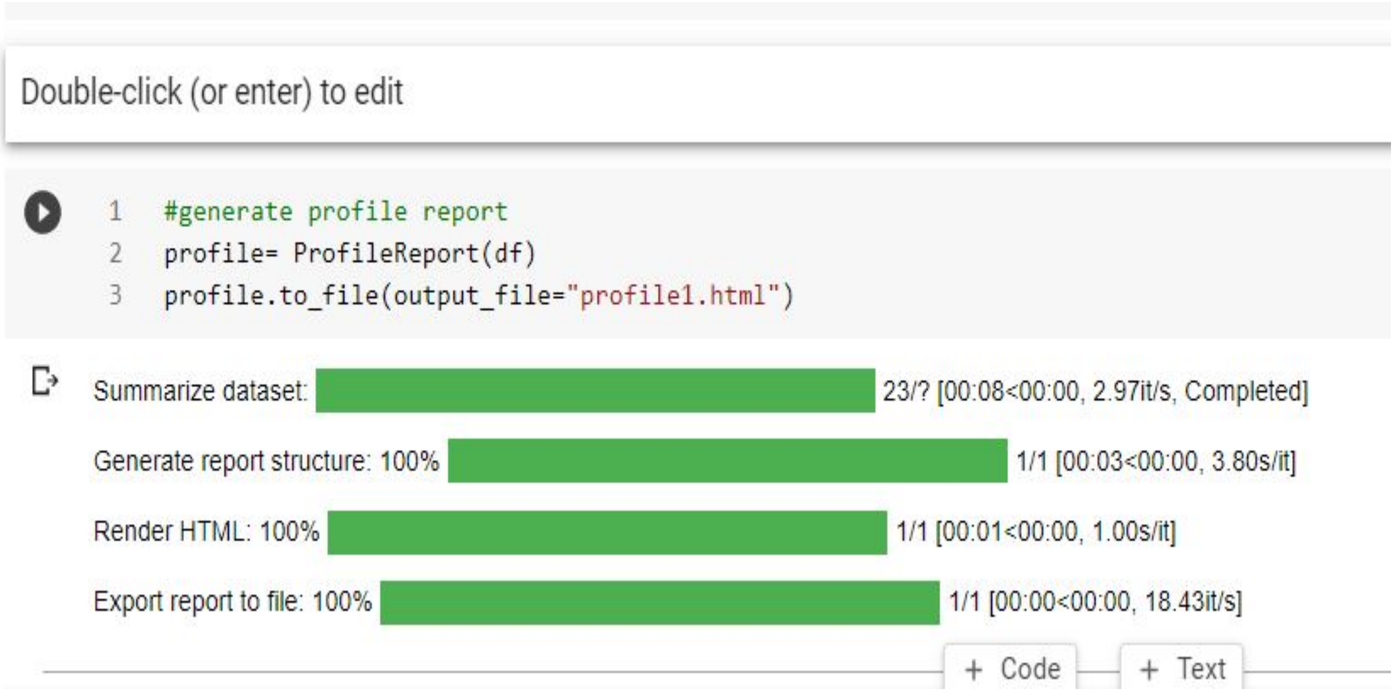


```
import pandas as pd
import pandas_profiling
import numpy as np
from pandas_profiling import ProfileReport
pd.set_option('display.max_columns', None)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier,GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
```

```
df=pd.read_csv('training_data.csv')
print(df)
```

```
      patient_id standard_lab_parameter_name  parameter_value     unit  \
0              0                         WBC           579.00  10^3/µl
1              1                          T4             5.00    µg/dl
2              2                         WBC             6.81  10^3/µl
3              3                          T4             5.70    µg/dl
4              4                         WBC             4.64  10^3/µl
...          ...                         ...              ...      ...
```

Creating a detailed report about the features in the dataset

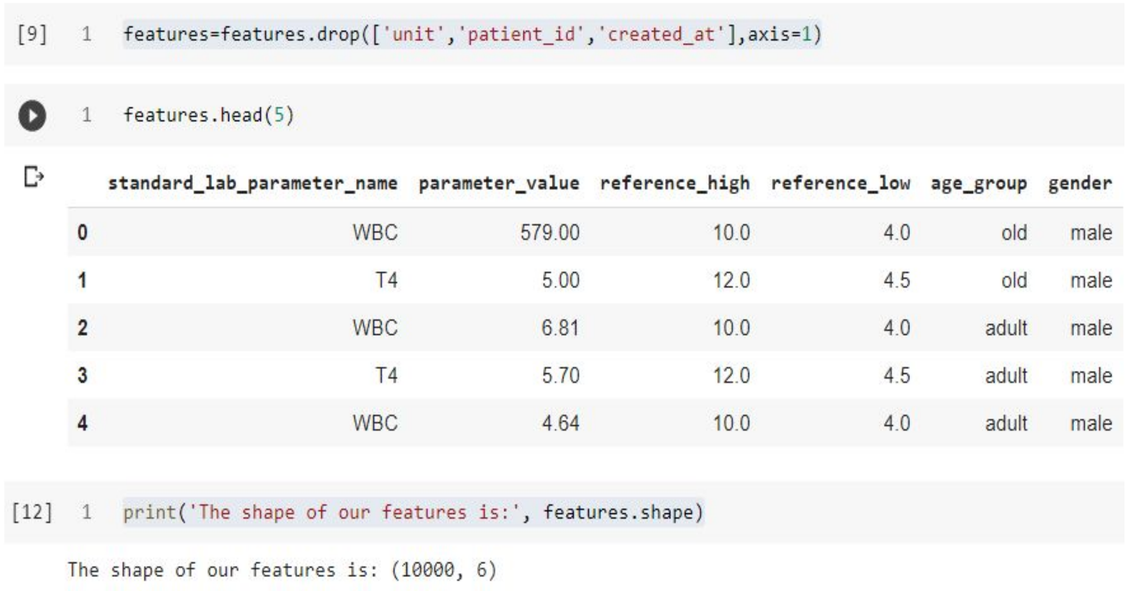


Double-click (or enter) to edit

```
#generate profile report
profile= ProfileReport(df)
profile.to_file(output_file="profile1.html")
```

```
Summarize dataset:                 23/? [00:08<00:00, 2.97it/s, Completed]
Generate report structure: 100%    1/1 [00:03<00:00, 3.80s/it]
Render HTML: 100%                  1/1 [00:01<00:00, 1.00s/it]
Export report to file: 100%        1/1 [00:00<00:00, 18.43it/s]
```

Preprocessing of Dataset



```
[9]  features=features.drop(['unit','patient_id','created_at'],axis=1)
```

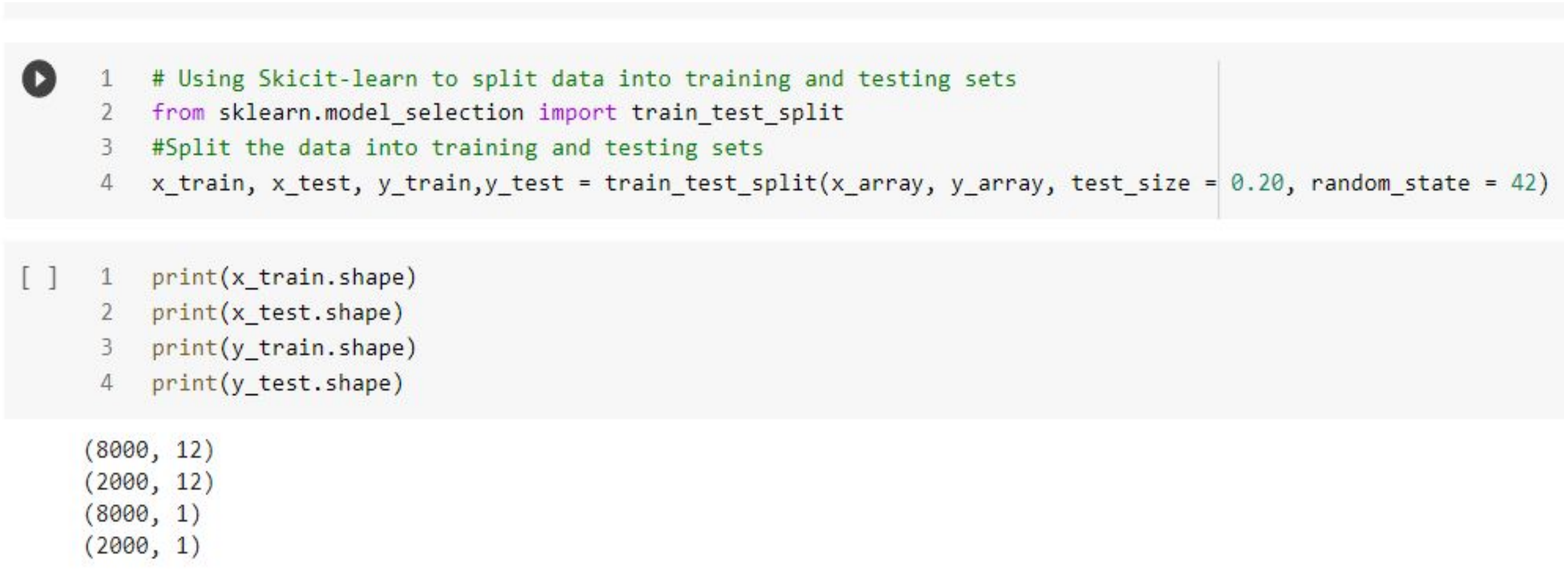
```
features.head(5)
```

	standard_lab_parameter_name	parameter_value	reference_high	reference_low	age_group	gender
0	WBC	579.00	10.0	4.0	old	male
1	T4	5.00	12.0	4.5	old	male
2	WBC	6.81	10.0	4.0	adult	male
3	T4	5.70	12.0	4.5	adult	male
4	WBC	4.64	10.0	4.0	adult	male

```
[12]  print('The shape of our features is:', features.shape)
```

```
The shape of our features is: (10000, 6)
```

Splitting the Dataset

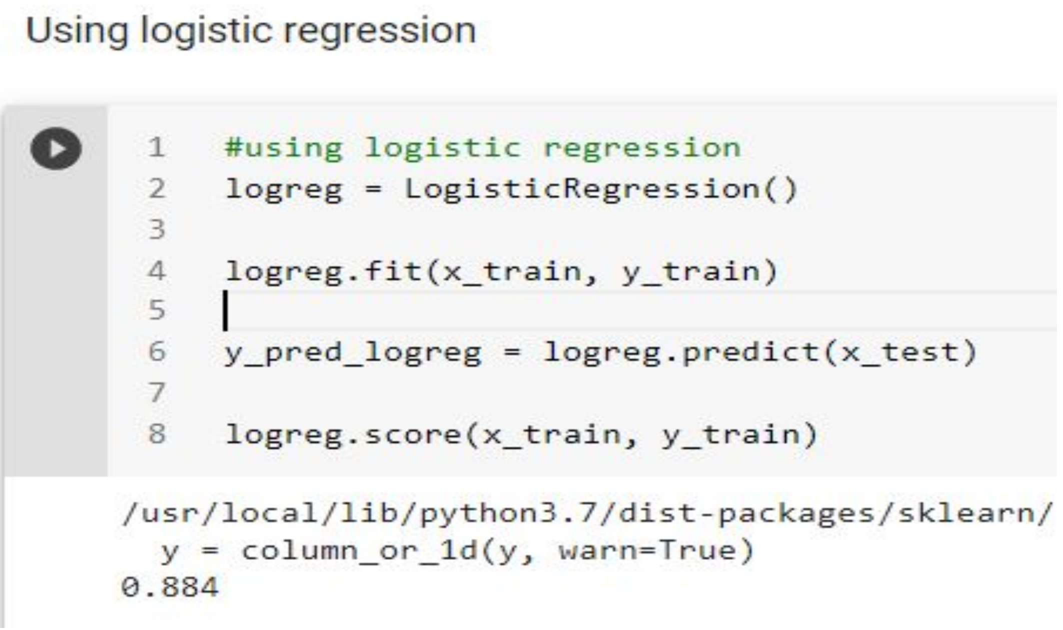


```
# Using Skicit-learn to split data into training and testing sets
from sklearn.model_selection import train_test_split
#Split the data into training and testing sets
x_train, x_test, y_train,y_test = train_test_split(x_array, y_array, test_size = 0.20, random_state = 42)
```

```
print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(8000, 12)
(2000, 12)
(8000, 1)
(2000, 1)
```

Applying Logistic Regression

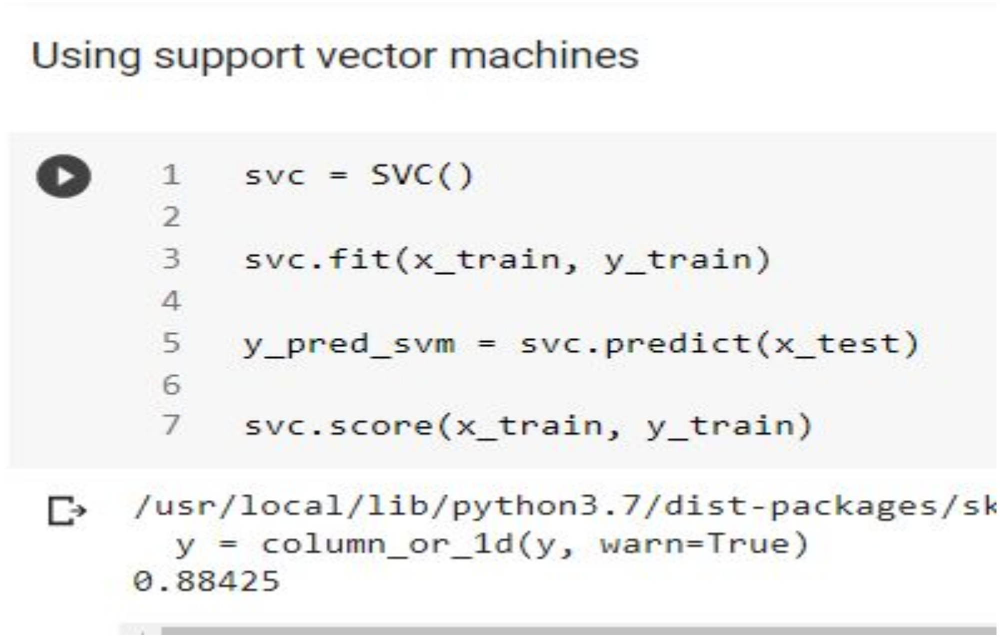

Using logistic regression

```
#using logistic regression
logreg = LogisticRegression()

logreg.fit(x_train, y_train)

y_pred_logreg = logreg.predict(x_test)

logreg.score(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/
  y = column_or_1d(y, warn=True)
0.884
```

Accuracy=88.4%

Support Vector Machine



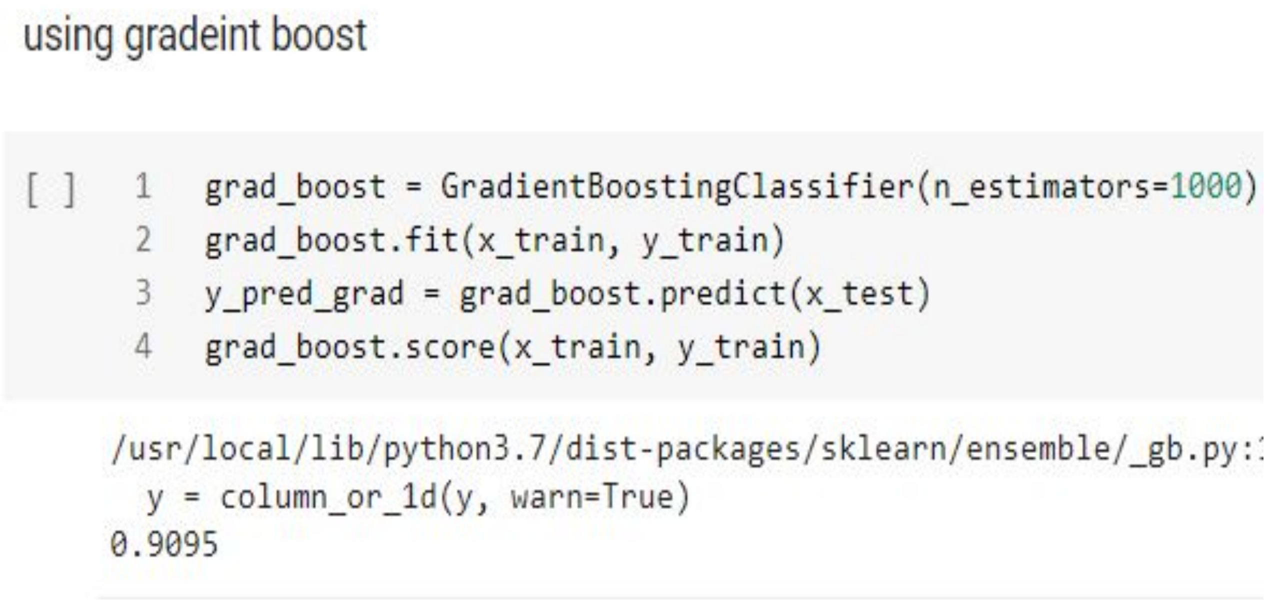
Using support vector machines

```
svc = SVC()

svc.fit(x_train, y_train)

y_pred_svm = svc.predict(x_test)

svc.score(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sk
  y = column_or_1d(y, warn=True)
0.88425
```

Accuracy=88.42%

Gradient Boost Model



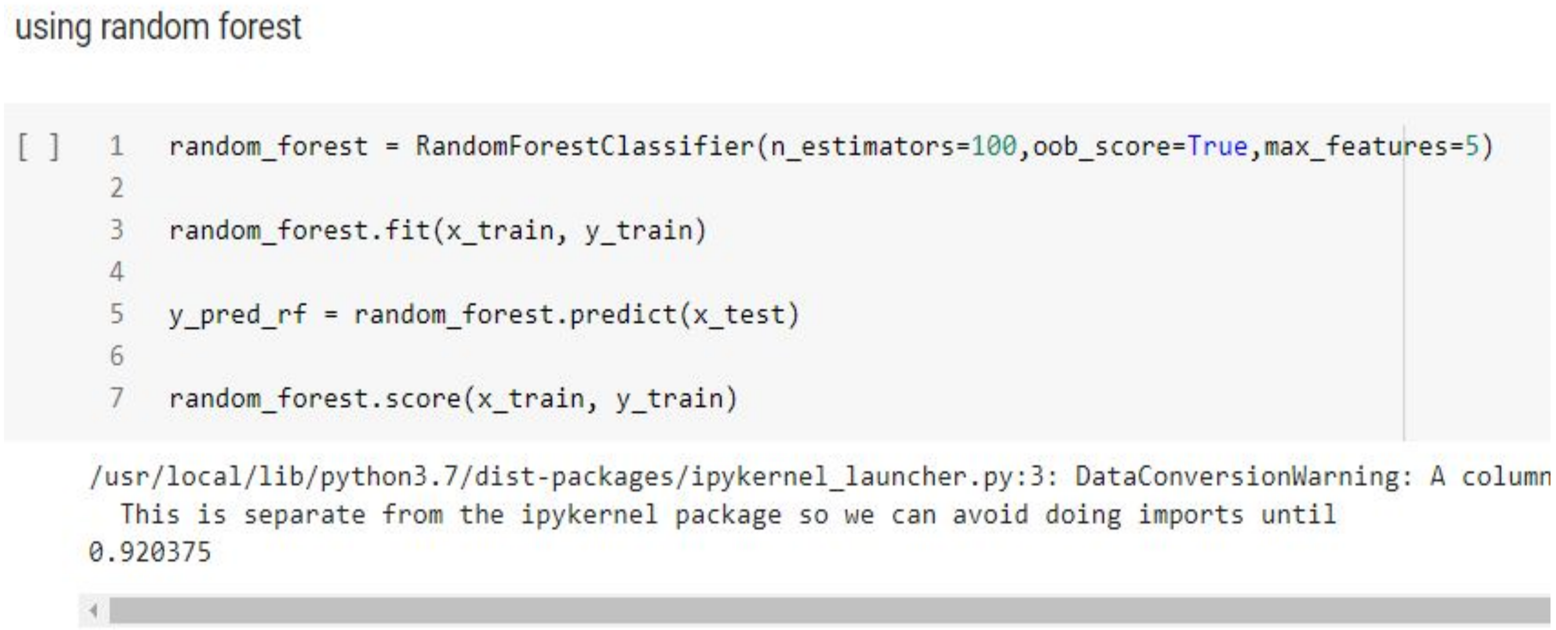
using gradeint boost

```
[ ]   grad_boost = GradientBoostingClassifier(n_estimators=1000)
      grad_boost.fit(x_train, y_train)
      y_pred_grad = grad_boost.predict(x_test)
      grad_boost.score(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/ensemble/_gb.py:
  y = column_or_1d(y, warn=True)
0.9095
```

Accuracy=90.95%

Using Random Forest Classifier



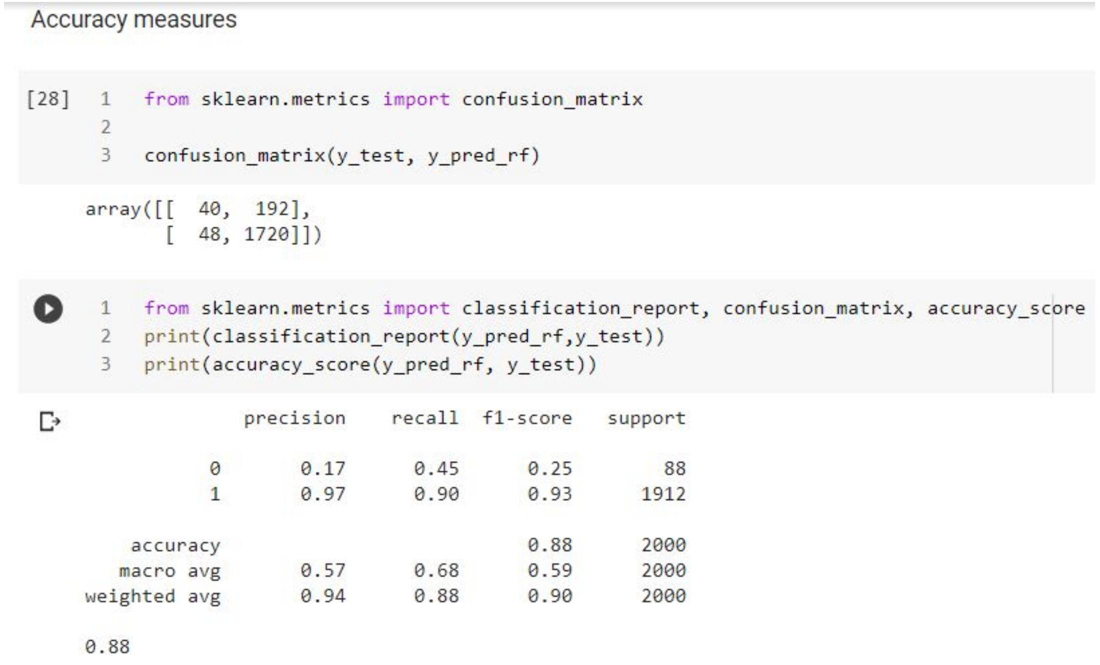
using random forest

```
[ ]
random_forest = RandomForestClassifier(n_estimators=100,oob_score=True,max_features=5)

random_forest.fit(x_train, y_train)

y_pred_rf = random_forest.predict(x_test)

random_forest.score(x_train, y_train)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3: DataConversionWarning: A column
  This is separate from the ipykernel package so we can avoid doing imports until
0.920375
```

Accuracy = 92.03 %

Acurracy Measure



Accuracy measures

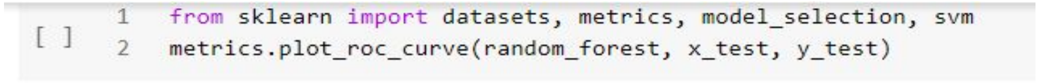
```
[28] from sklearn.metrics import confusion_matrix

     confusion_matrix(y_test, y_pred_rf)
```

```
array([[  40,  192],
       [  48, 1720]])
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
print(classification_report(y_pred_rf,y_test))
print(accuracy_score(y_pred_rf, y_test))
```

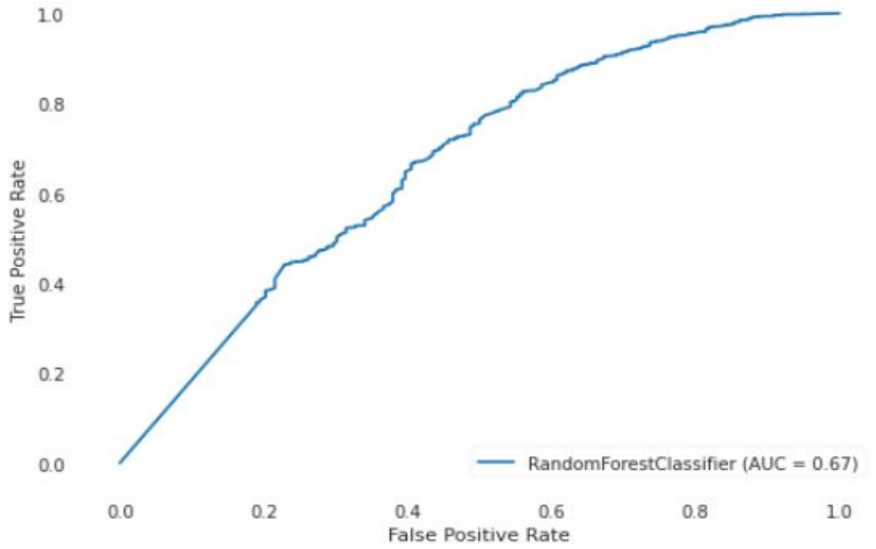
```
              precision    recall  f1-score   support

           0       0.17      0.45      0.25        88
           1       0.97      0.90      0.93      1912

    accuracy                           0.88      2000
   macro avg       0.57      0.68      0.59      2000
weighted avg       0.94      0.88      0.90      2000

0.88
```

ROC & AOC curve



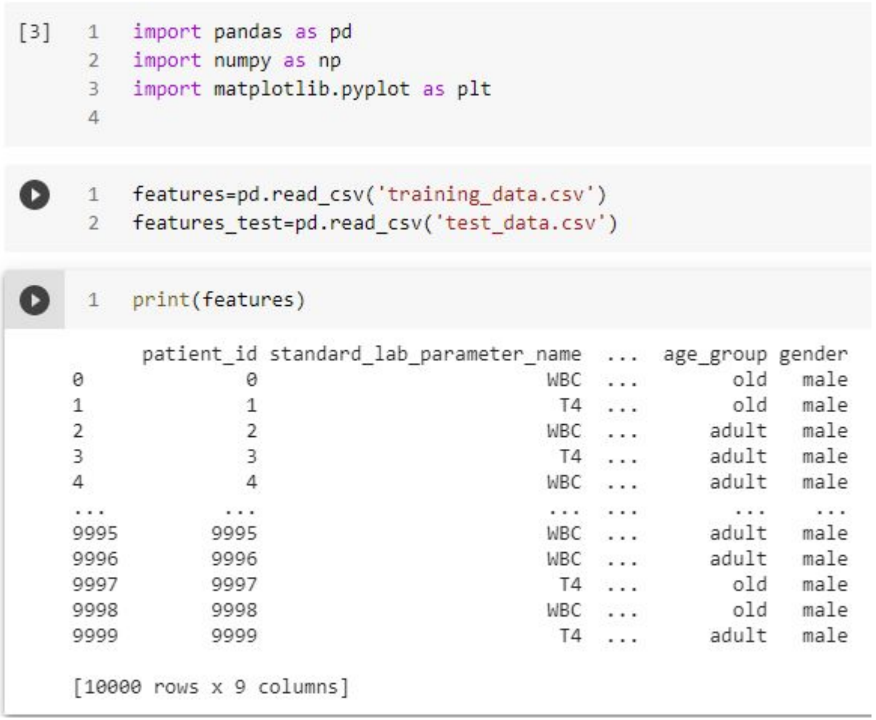
```
[ ]
from sklearn import datasets, metrics, model_selection, svm
metrics.plot_roc_curve(random_forest, x_test, y_test)
```

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f1f9b371f50>
```



Comparison of Chloride With age_group

Importing Libraries & Loading Train Dataset



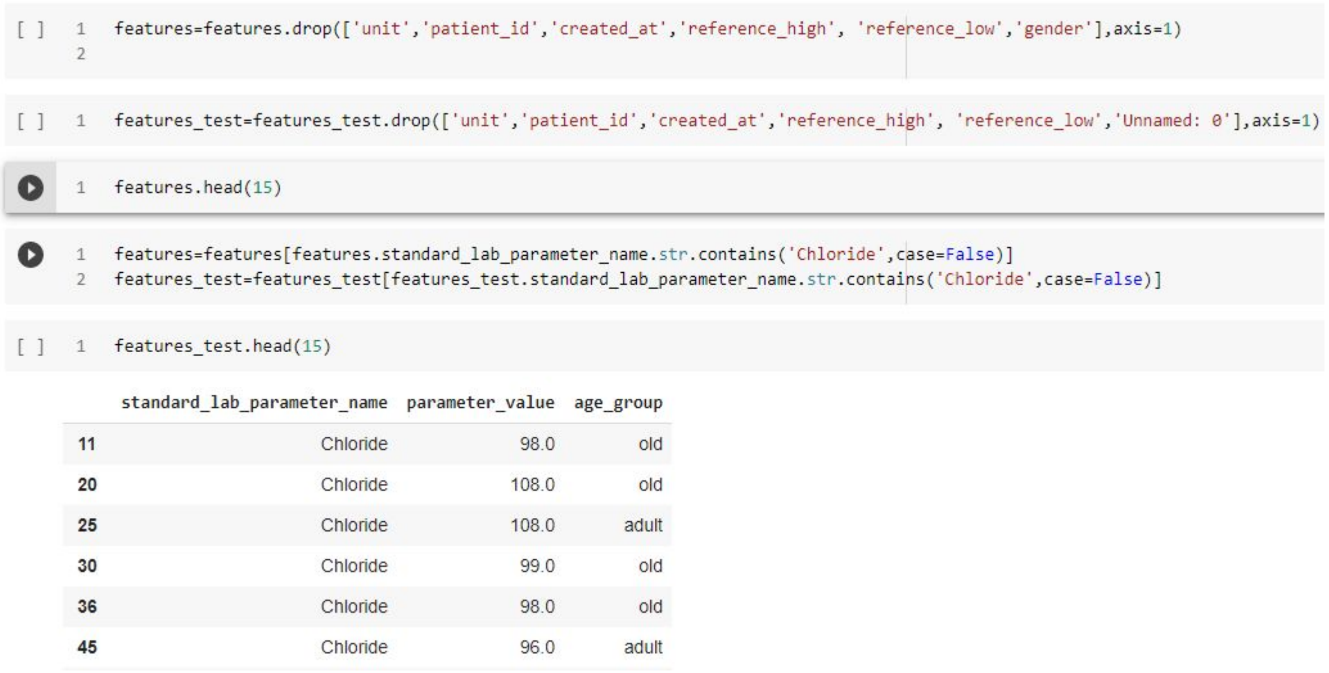
```
[3]
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

```
features=pd.read_csv('training_data.csv')
features_test=pd.read_csv('test_data.csv')
```

```
print(features)
```

```
      patient_id standard_lab_parameter_name  ...  age_group gender
0              0                         WBC  ...        old   male
1              1                          T4  ...        old   male
2              2                         WBC  ...      adult   male
3              3                          T4  ...      adult   male
4              4                         WBC  ...      adult   male
...          ...                         ...  ...        ...    ...
9995        9995                         WBC  ...      adult   male
9996        9996                         WBC  ...      adult   male
9997        9997                          T4  ...        old   male
9998        9998                         WBC  ...        old   male
9999        9999                          T4  ...      adult   male

[10000 rows x 9 columns]
```

Preprocessing of Data



```
[ ]  features=features.drop(['unit','patient_id','created_at','reference_high', 'reference_low','gender'],axis=1)
```

```
[ ]  features_test=features_test.drop(['unit','patient_id','created_at','reference_high', 'reference_low','Unnamed: 0'],axis=1)
```

```
features.head(15)
```

```
features=features[features.standard_lab_parameter_name.str.contains('Chloride',case=False)]
features_test=features_test[features_test.standard_lab_parameter_name.str.contains('Chloride',case=False)]
```

```
[ ]  features_test.head(15)
```

	standard_lab_parameter_name	parameter_value	age_group
11	Chloride	98.0	old
20	Chloride	108.0	old
25	Chloride	108.0	adult
30	Chloride	99.0	old
36	Chloride	98.0	old
45	Chloride	96.0	adult

Plotted Eight separate Graphs

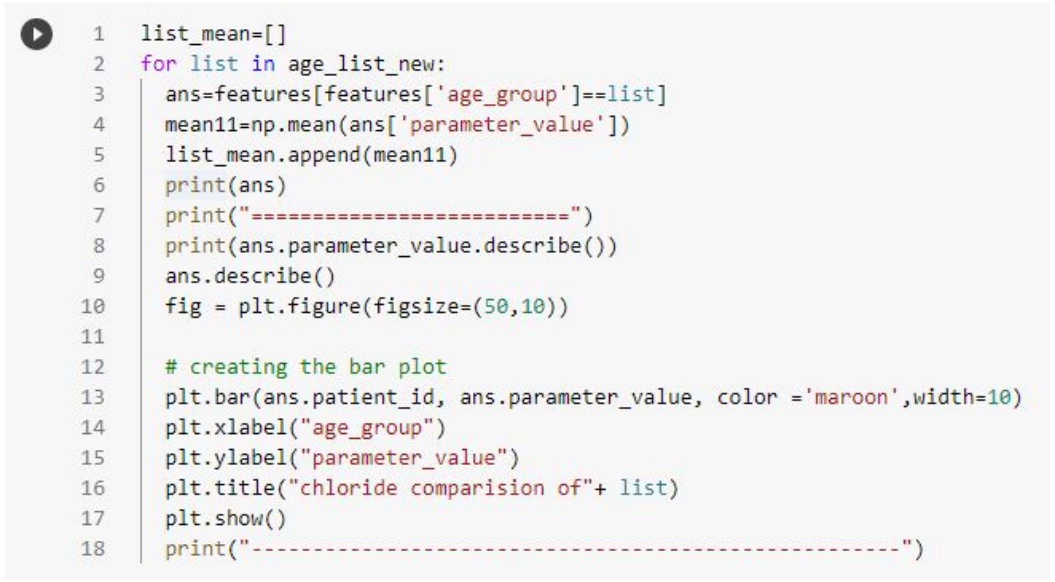
For Train and Test dataset

Patient_id Vs old

Patient_id Vs adult

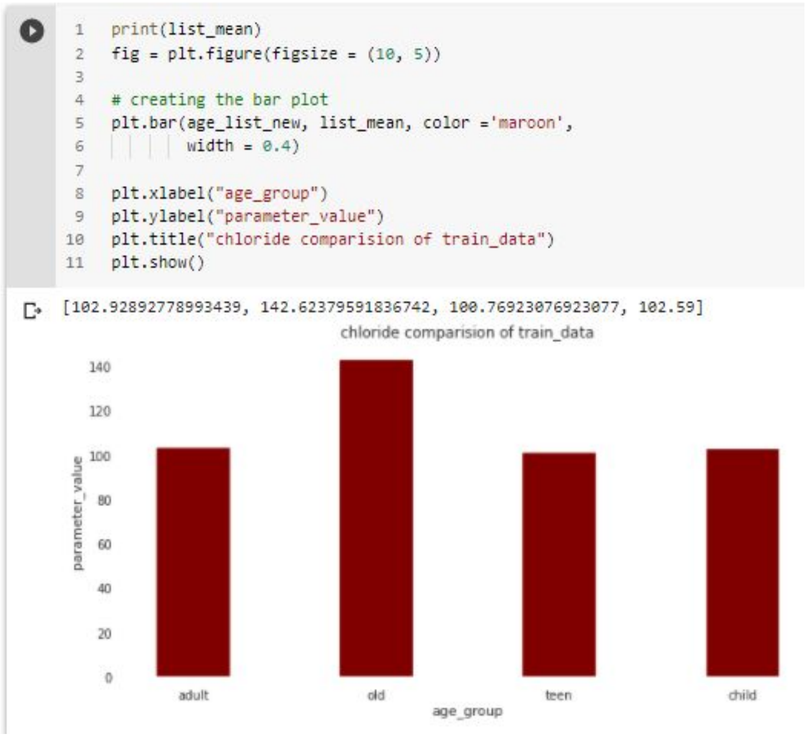
Patient_id Vs teen

Patient_id Vs child



```
list_mean=[]
for list in age_list_new:
  ans=features[features['age_group']==list]
  mean11=np.mean(ans['parameter_value'])
  list_mean.append(mean11)
  print(ans)
  print("==========================")
  print(ans.parameter_value.describe())
  ans.describe()
  fig = plt.figure(figsize=(50,10))

  # creating the bar plot
  plt.bar(ans.patient_id, ans.parameter_value, color ='maroon',width=10)
  plt.xlabel("age_group")
  plt.ylabel("parameter_value")
  plt.title("chloride comparision of"+ list)
  plt.show()
  print("-------------------------------------------------------")
```

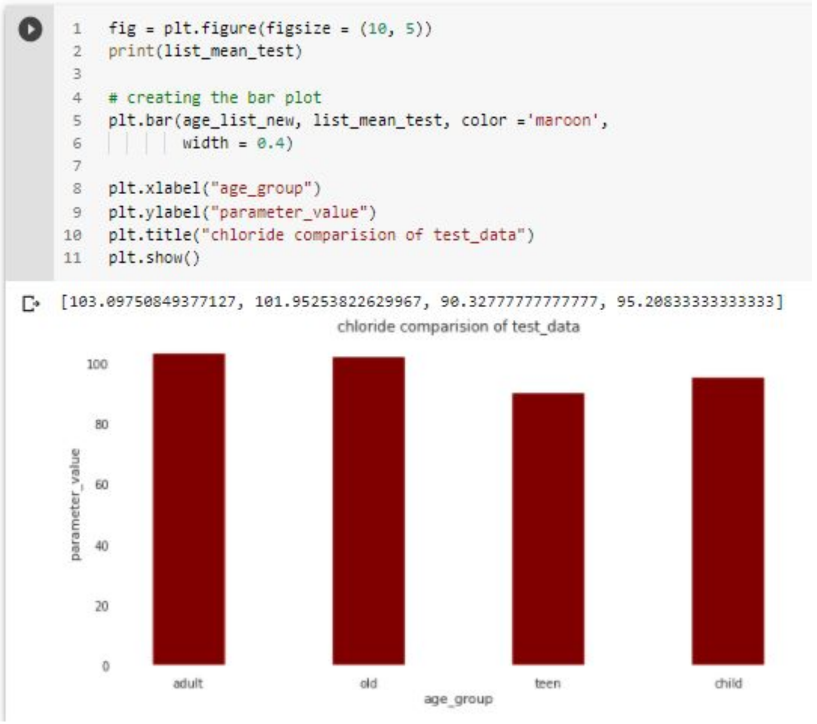
Chloride Vs age_group Train Data



```
print(list_mean)
fig = plt.figure(figsize = (10, 5))

# creating the bar plot
plt.bar(age_list_new, list_mean, color ='maroon',
        width = 0.4)

plt.xlabel("age_group")
plt.ylabel("parameter_value")
plt.title("chloride comparision of train_data")
plt.show()
```

```
[102.92892778993439, 142.62379591836742, 100.76923076923077, 102.59]
```

Chloride Vs age_group Test Data

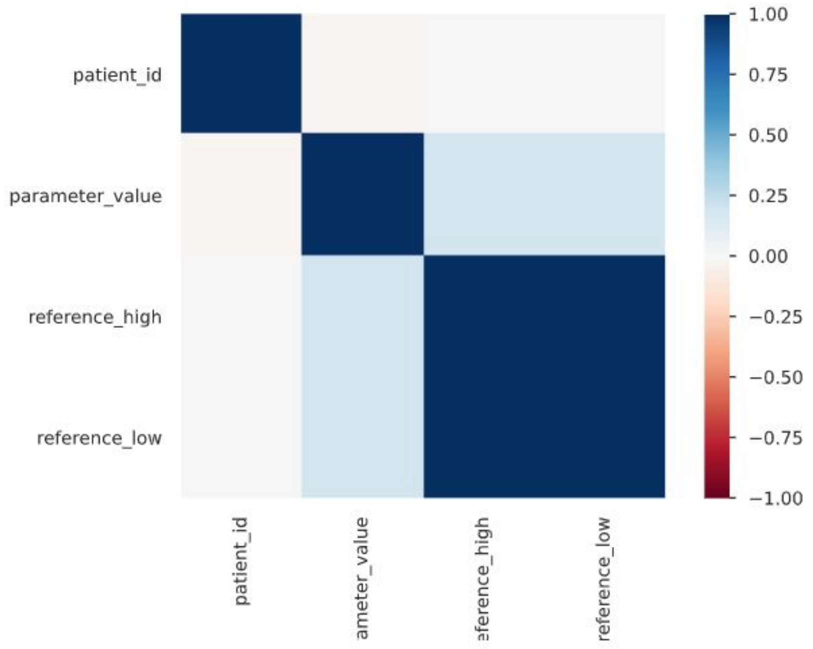


```
fig = plt.figure(figsize = (10, 5))
print(list_mean_test)

# creating the bar plot
plt.bar(age_list_new, list_mean_test, color ='maroon',
        width = 0.4)

plt.xlabel("age_group")
plt.ylabel("parameter_value")
plt.title("chloride comparision of test_data")
plt.show()
```

```
[103.09750849377127, 101.95253822629967, 90.32777777777777, 95.20833333333333]
```

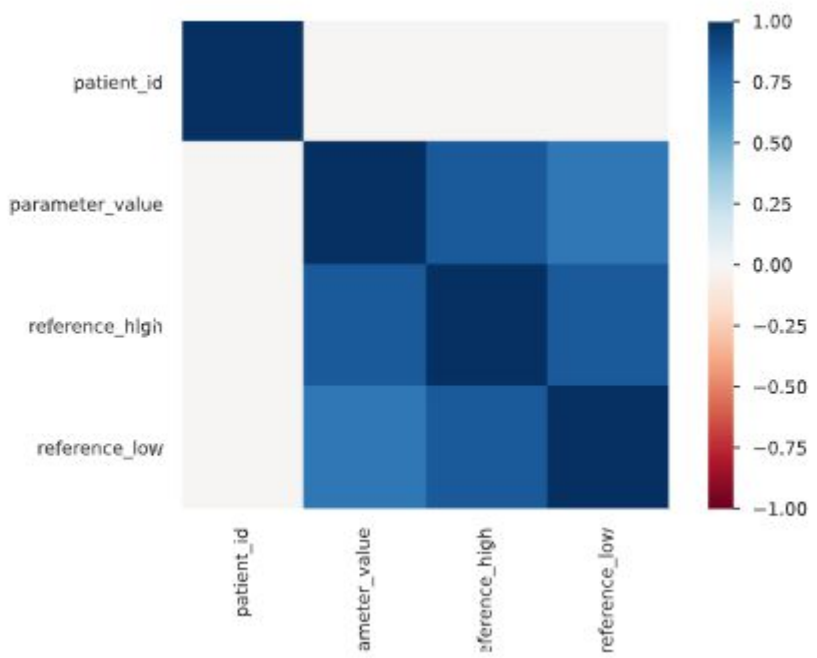
Some Interesting Facts

Pearson’s Corelation



The Pearson's correlation coefficient (*r*) is a measure of linear correlation between two variables. It's value lies between -1 and +1, -1 indicating total negative linear correlation, 0 indicating no linear correlation and 1 indicating total positive linear correlation. Furthermore, *r* is invariant under separate changes in location and scale of the two variables, implying that for a linear function the angle to the x-axis does not affect *r*.

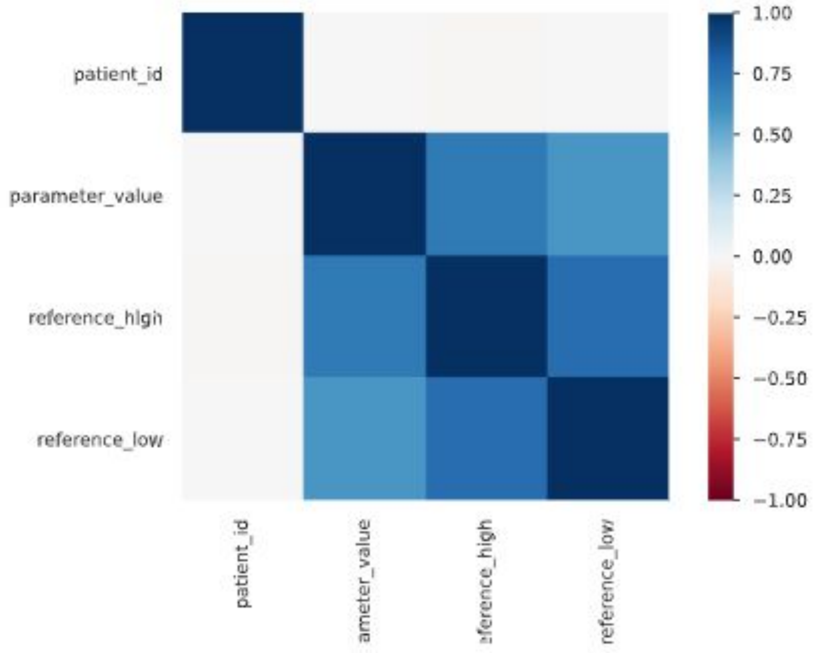
Spearman’s Rank Correlation



The Spearman's rank correlation coefficient (*ρ*) is a measure of monotonic correlation between two variables, and is therefore better in catching nonlinear monotonic correlations than Pearson's *r*. It's value lies between -1 and +1, -1 indicating total negative monotonic correlation, 0 indicating no monotonic correlation and 1 indicating total positive monotonic correlation.

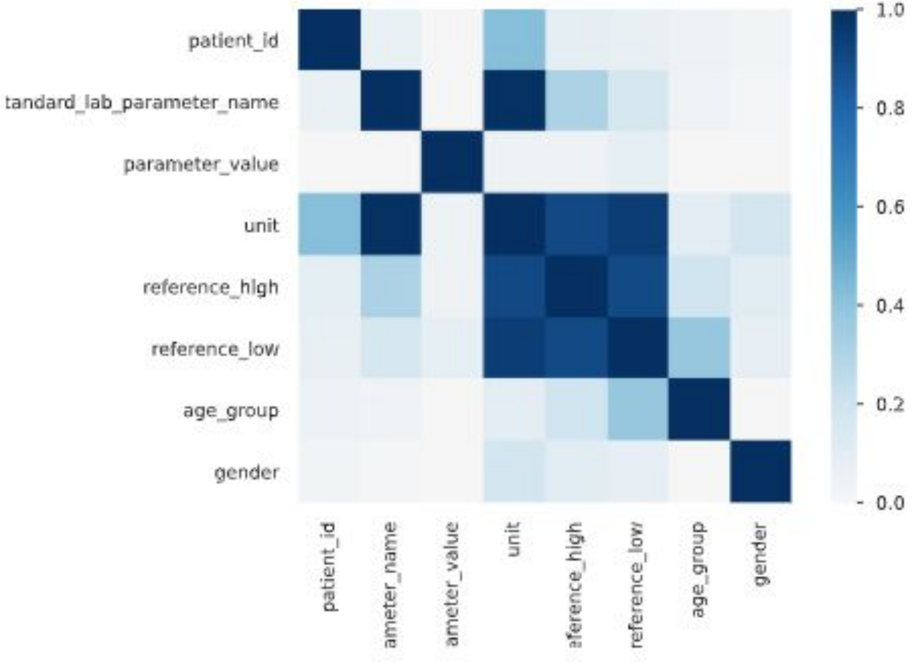
To calculate *ρ* for two variables *X* and *Y*, one divides the covariance of the rank variables of *X* and *Y* by the product of their standard deviations.

Kendall’s Correlation



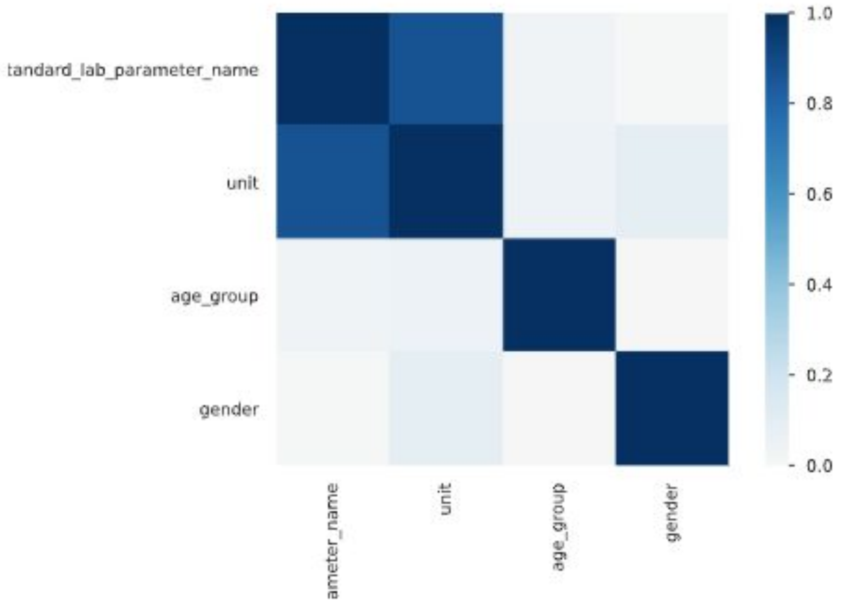
Similarly to Spearman's rank correlation coefficient, the Kendall rank correlation coefficient (*τ*) measures ordinal association between two variables. It's value lies between -1 and +1, -1 indicating total negative correlation, 0 indicating no correlation and 1 indicating total positive correlation.
To calculate *τ* for two variables *X* and *Y*, one determines the number of concordant and discordant pairs of observations. *τ* is given by the number of concordant pairs minus the discordant pairs divided by the total number of pairs.

Phik Correlation



Phik (φk) is a new and practical correlation coefficient that works consistently between categorical, ordinal and interval variables, captures non-linear dependency and reverts to the Pearson correlation coefficient in case of a bivariate normal input distribution.

Cramer’s V Correlation



Cramér's V is an association measure for nominal random variables. The coefficient ranges from 0 to 1, with 0 indicating independence and 1 indicating perfect association. The empirical estimators used for Cramér's V have been proved to be biased, even for large samples.

Practical Use Of Dataset

Practical Uses

- ➔ As dataset contain standard lab parameter we can predict the deficiency and excess of the parameter in a particular age_group.
- ➔ Can find the average values of the lab parameter according to the particular age_groups.
- ➔