Sperm concentration classification for male fertility analysis

Student:

Raúl Castillo Segoviano

A01351831

Teacher:

Benjamín Valdés Aguirre

Intelligent systems

May 2021

Contents

Abstract

Through the following writing it will be fully described one of the many applications intelligent systems can provide, due to the fact by the intense speed the technology is moving forward day by day, AI is coming every day closer and many facts that can involve machine learning are being applied to solve daily basis situations.

Found next, it will be described the implementation of diverse machine learning topics of like decision tree, label encoder among others, in order of finding the best classification of sperm concentration along men in between 18 and 36 years old.

Introduction

For starters, on 2018, Dr. Rachel Tavel medically reviewed an article regarding sperm count and the importance of knowing your status if you're a male trying to conceive a child along that an abnormal sperm count may also indicate an underlying health condition.

Retaking the mentioned above, it is considered as normal sperm count ranges from 15 million to more than 200 million sperm per milliliter of semen. Anything less than 15 million sperm per milliliter, or 39 million sperm per ejaculate, is considered low. A low sperm count is often referred to as oligospermia. A high, or above average, sperm count is over 200 million sperm per millimeter. [1]

To give a better description of the importance of a normal sperm count, if you're trying to conceive naturally, a healthy sperm count is often necessary. Even though it only takes one sperm and one egg to get pregnant, more healthy sperm will increase your chances of pregnancy each month.

Male infertility factor, often due to a low sperm count, is a common reason many couples have trouble conceiving. But couples may also experience other health issues that can affect fertility. In some cases, infertility may be due to female factors, like low ovarian reserve, a blocked fallopian tube, endometriosis, among others.

On the other hand, even if you aren't trying to conceive, your sperm count may be an important measure of overall health. One study found men with a low sperm count were more likely to have a higher percentage of body and higher blood pressure than men with higher sperm counts. They also experienced a higher frequency of metabolic syndrome, or higher chance of developing diabetes, heart disease, and stroke. [3]

Therefore, on 2013 David Gil from the University of Alicante, decided to analyze 100 volunteers instances that provide a semen sample according to the WHO 2010 criteria. Many features were taken in consideration such as age, childish diseases, accidents, serious trauma, etc. Concluding in a data set with 10 assets and 100 lines that is going to me taken for a machine learning system in which the data will be trained and tested in order of resulting in a program that can provide a classification according on the inputs given.

To end, there are many factors that can affect your sperm count, including lifestyle choices or underlying medical conditions. If you have a low sperm count, your doctor may recommend options for you to raise your sperm count, or they may refer you to a urologist or fertility specialist, if needed, there are many fertility options available today, including a range of treatments like:

Intrauterine insemination (IUI)

In vitro fertilization (IVF)

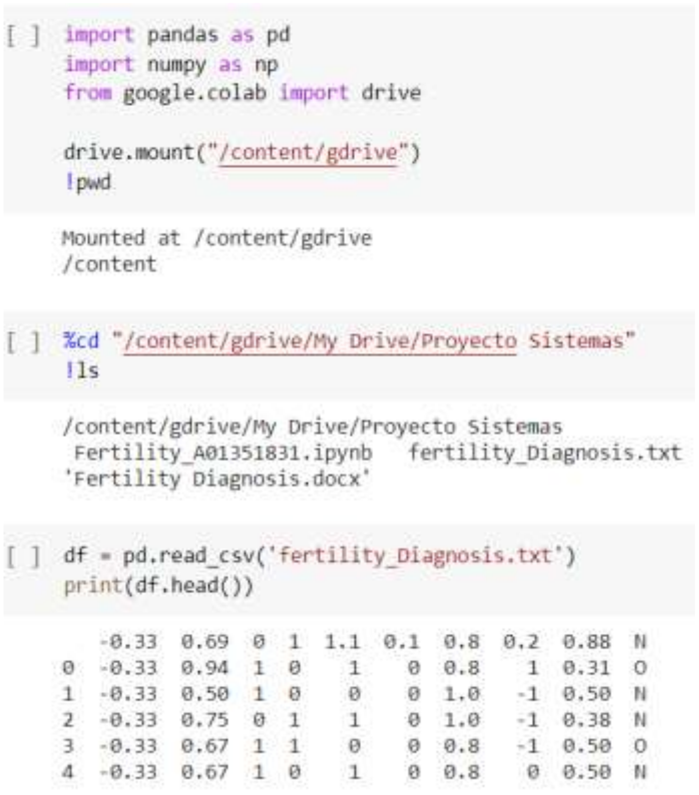
IVF with intracytoplasmic sperm injection (ICSI)

It is recommended you talk to your doctor about your concerns and options. [2]

Data set and background

Link to data set: https://archive.ics.uci.edu/ml/datasets/Fertility

Once made the previously mentioned data set into a .txt file, the first thing done was to import the pandas library and allowing the program to access my google drive, as well as reading the content found in the data set to import it and make it readable to colab by turning it into a data frame.

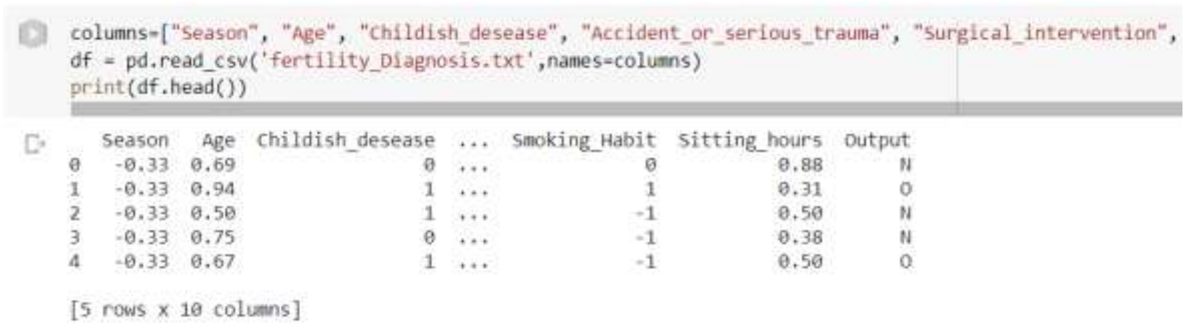


```
[ ] import pandas as pd
    import numpy as np
    from google.colab import drive

    drive.mount("/content/gdrive")
    !pwd

    Mounted at /content/gdrive
    /content
```

```
[ ] %cd "/content/gdrive/My Drive/Proyecto Sistemas"
    !ls

    /content/gdrive/My Drive/Proyecto Sistemas
     Fertility_A01351831.ipynb   fertility_Diagnosis.txt
    'Fertility Diagnosis.docx'
```

```
[ ] df = pd.read_csv('fertility_Diagnosis.txt')
    print(df.head())

       -0.33  0.69  0  1  1.1  0.1  0.8  0.2  0.88  N
    0  -0.33  0.94  1  0    1    0  0.8    1  0.31  O
    1  -0.33  0.50  1  0    0    0  1.0   -1  0.50  N
    2  -0.33  0.75  0  1    1    0  1.0   -1  0.38  N
    3  -0.33  0.67  1  1    0    0  0.8   -1  0.50  O
    4  -0.33  0.67  1  0    1    0  0.8    0  0.50  N
```

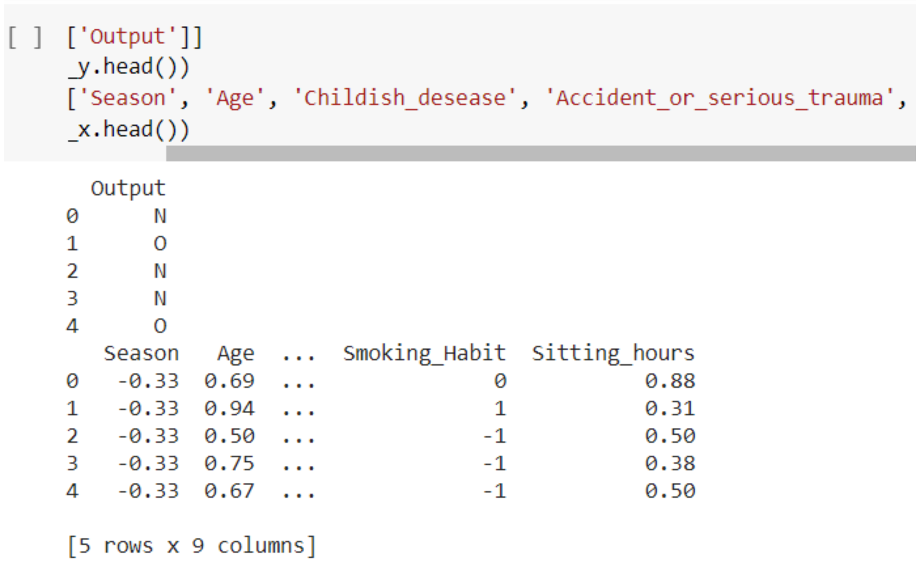
Next, the columns that included the assets were given a reasonable name. During the hole program the results were printed to ensure its correct functioning.



```
columns=["Season", "Age", "Childish_desease", "Accident_or_serious_trauma", "Surgical_intervention",
df = pd.read_csv('fertility_Diagnosis.txt',names=columns)
print(df.head())
```

```
   Season   Age  Childish_desease  ...  Smoking_Habit  Sitting_hours  Output
0   -0.33  0.69                 0  ...              0           0.88       N
1   -0.33  0.94                 1  ...              1           0.31       O
2   -0.33  0.50                 1  ...             -1           0.50       N
3   -0.33  0.75                 0  ...             -1           0.38       N
4   -0.33  0.67                 1  ...             -1           0.50       O

[5 rows x 10 columns]
```

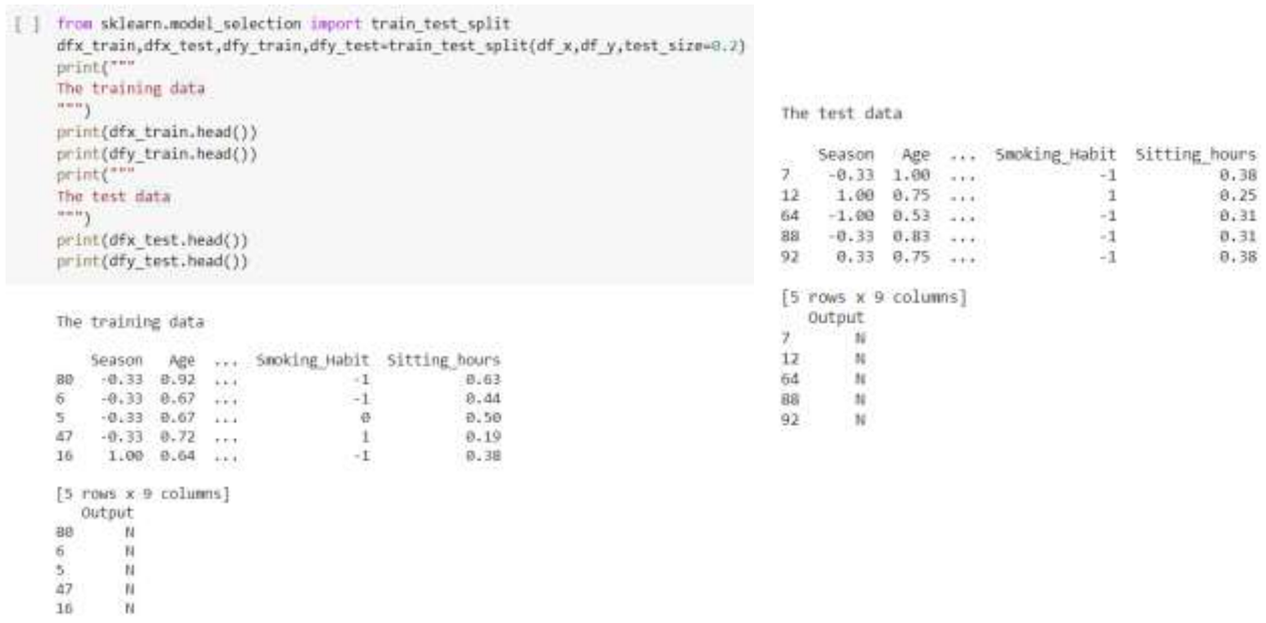
Then, the features are separated from the output in order for the program to difference between the assets and the one which is going to be my class.



```
[ ]  ['Output']]
     _y.head())
     ['Season', 'Age', 'Childish_desease', 'Accident_or_serious_trauma',
     _x.head())
```

```
  Output
0      N
1      O
2      N
3      N
4      O
   Season   Age  ...  Smoking_Habit  Sitting_hours
0   -0.33  0.69  ...              0           0.88
1   -0.33  0.94  ...              1           0.31
2   -0.33  0.50  ...             -1           0.50
3   -0.33  0.75  ...             -1           0.38
4   -0.33  0.67  ...             -1           0.50

[5 rows x 9 columns]
```

The features are once again separated, but this time the separation is to have data that is going to be trained and the rest is going to be for testing (20%).



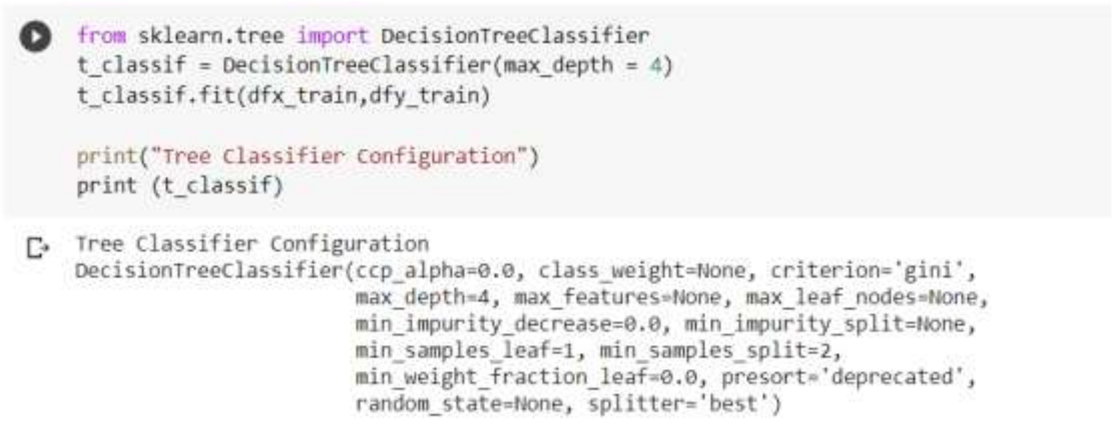
```
[ ]  from sklearn.model_selection import train_test_split
     dfx_train,dfx_test,dfy_train,dfy_test=train_test_split(df_x,df_y,test_size=0.2)
     print("""
     The training data
     """)
     print(dfx_train.head())
     print(dfy_train.head())
     print("""
     The test data
     """)
     print(dfx_test.head())
     print(dfy_test.head())
```

```
The training data

    Season   Age  ...  Smoking_Habit  Sitting_hours
80   -0.33  0.92  ...             -1           0.63
6    -0.33  0.67  ...             -1           0.44
5    -0.33  0.67  ...              0           0.50
47   -0.33  0.72  ...              1           0.19
16    1.00  0.64  ...             -1           0.38

[5 rows x 9 columns]
   Output
80      N
6       N
5       N
47      N
16      N
```

```
The test data

    Season   Age  ...  Smoking_Habit  Sitting_hours
7    -0.33  1.00  ...             -1           0.38
12    1.00  0.75  ...              1           0.25
64   -1.00  0.53  ...             -1           0.31
88   -0.33  0.83  ...             -1           0.31
92    0.33  0.75  ...             -1           0.38

[5 rows x 9 columns]
   Output
7       N
12      N
64      N
88      N
92      N
```

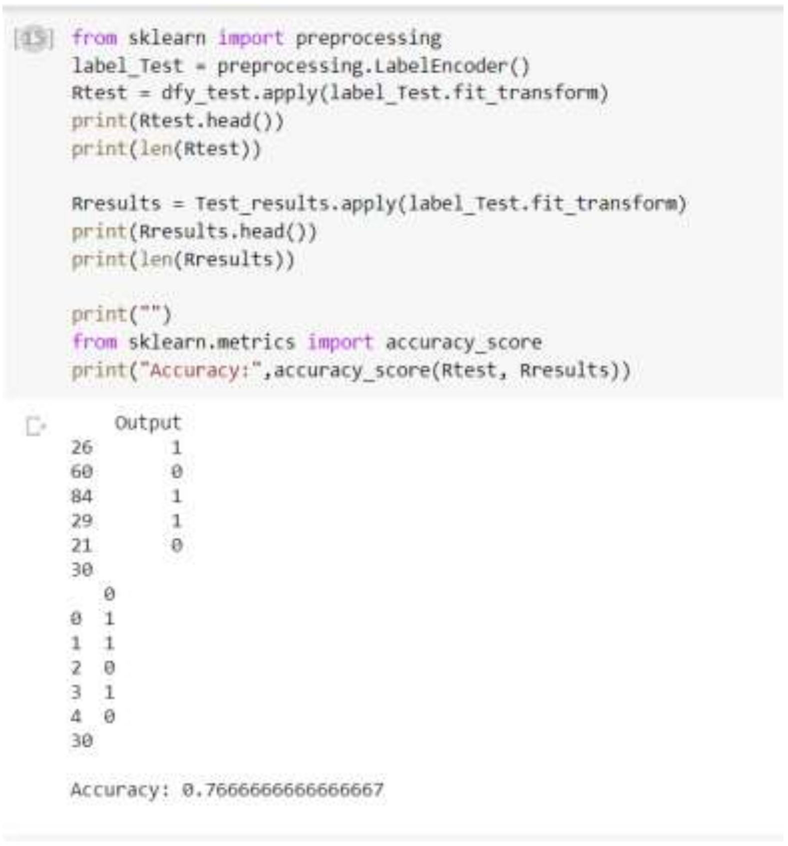
Followed, I decided to use the hyperparameter decision tree. This because this type of models can order decisions and their possible consequence including chance event outcomes.

At first, I tried selecting a depth of 4 but gave an accuracy of 76%. By testing different values, the best result was a depth of 6 since the accuracy was now 85%. Bigger values didn't improve the percentage and didn't make sense to run it more times than necessary. Once trained the features, the algorithm is now capable of predicting results.



```
from sklearn.tree import DecisionTreeClassifier
t_classif = DecisionTreeClassifier(max_depth = 4)
t_classif.fit(dfx_train,dfy_train)

print("Tree Classifier Configuration")
print (t_classif)
```

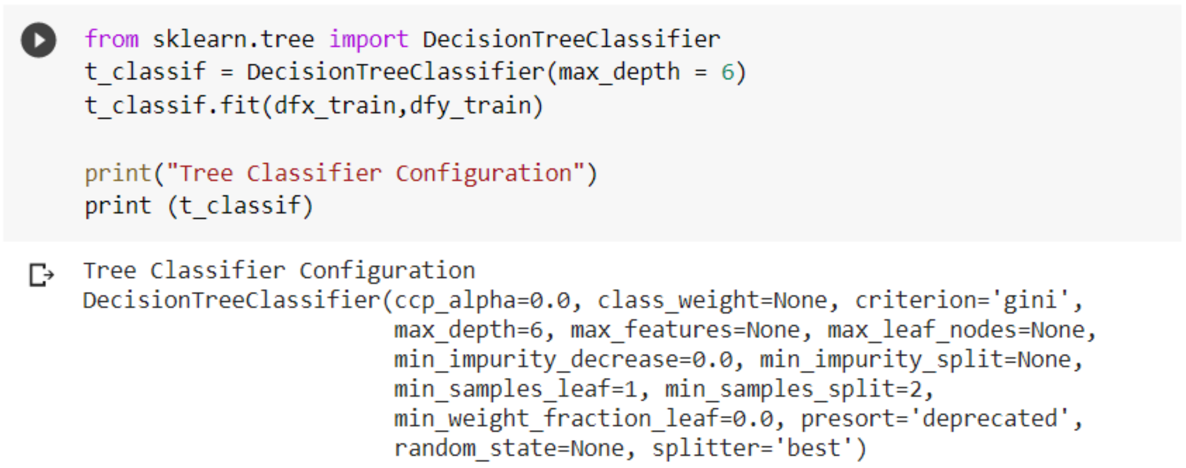
```
Tree Classifier Configuration
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=4, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```



```
[15] from sklearn import preprocessing
label_Test = preprocessing.LabelEncoder()
Rtest = dfy_test.apply(label_Test.fit_transform)
print(Rtest.head())
print(len(Rtest))

Rresults = Test_results.apply(label_Test.fit_transform)
print(Rresults.head())
print(len(Rresults))

print("")
from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(Rtest, Rresults))
```

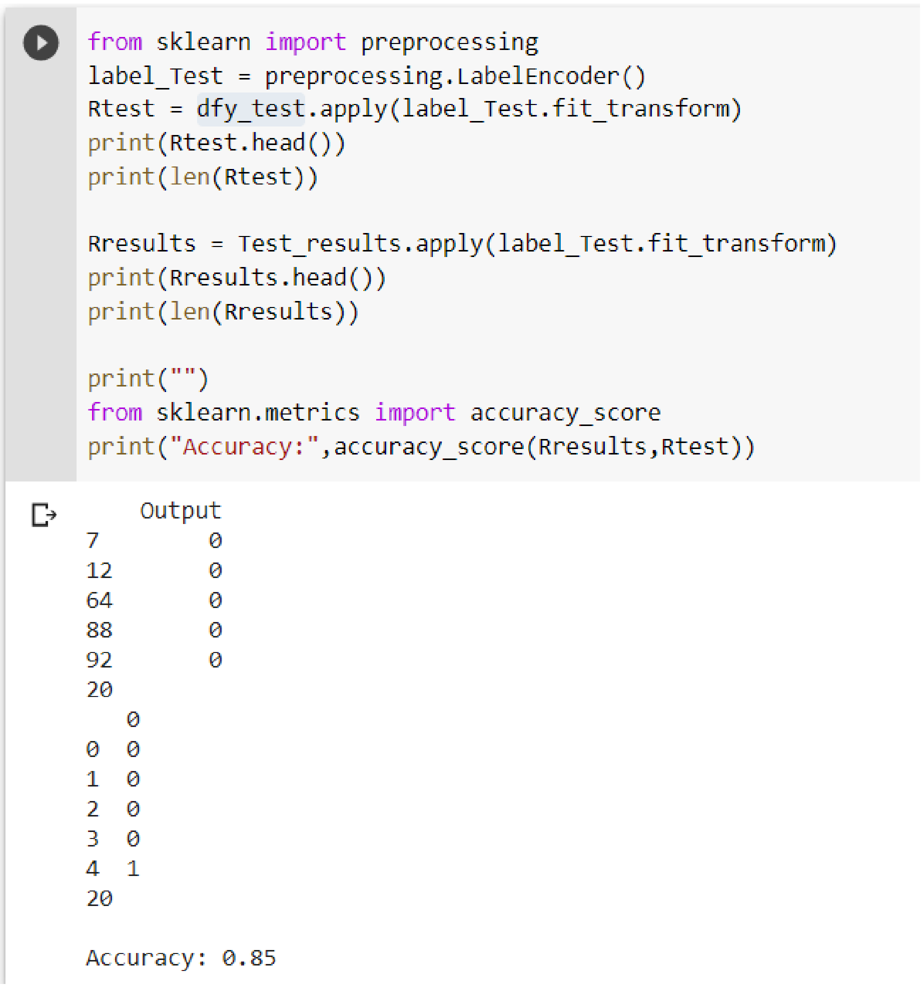
```
    Output
26       1
60       0
84       1
29       1
21       0
30
   0
0  1
1  1
2  0
3  1
4  0
30

Accuracy: 0.7666666666666667
```

First attempt



```
from sklearn.tree import DecisionTreeClassifier
t_classif = DecisionTreeClassifier(max_depth = 6)
t_classif.fit(dfx_train,dfy_train)

print("Tree Classifier Configuration")
print (t_classif)
```

```
Tree Classifier Configuration
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=6, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```



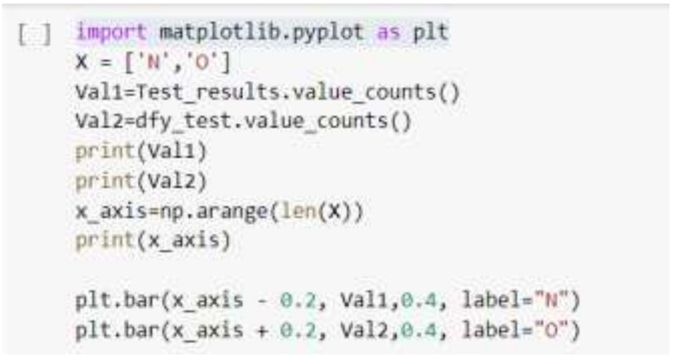
```
from sklearn import preprocessing
label_Test = preprocessing.LabelEncoder()
Rtest = dfy_test.apply(label_Test.fit_transform)
print(Rtest.head())
print(len(Rtest))

Rresults = Test_results.apply(label_Test.fit_transform)
print(Rresults.head())
print(len(Rresults))

print("")
from sklearn.metrics import accuracy_score
print("Accuracy:",accuracy_score(Rresults,Rtest))
```

```
    Output
7        0
12       0
64       0
88       0
92       0
20
   0
0  0
1  0
2  0
3  0
4  1
20

Accuracy: 0.85
```

Best result

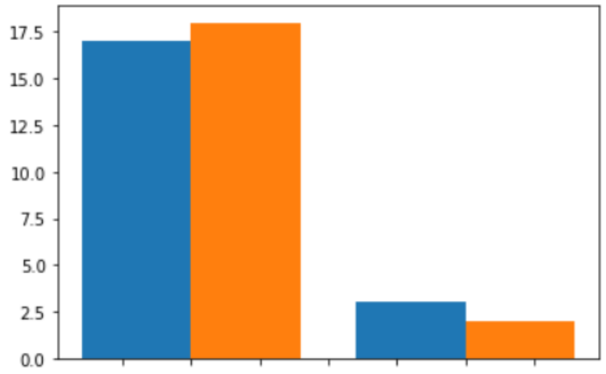
Once the decision tree was created, I decided to apply the process of label encoding, this due to the fact that the outputs were given in the form of [N , O]. Therefore, the program is able to separate the outcomes into now [0 , 1]. This was able because the variables were categorizable into a unique integer. This was done to the trained data and the tested one, this is how the accuracy percentage was obtained.

Next, from the 20% tested data resulting in 20 assets, were graphed the comparation between the predicted outcomes by counting the length on each possible result. Visually it is better visible the differences.

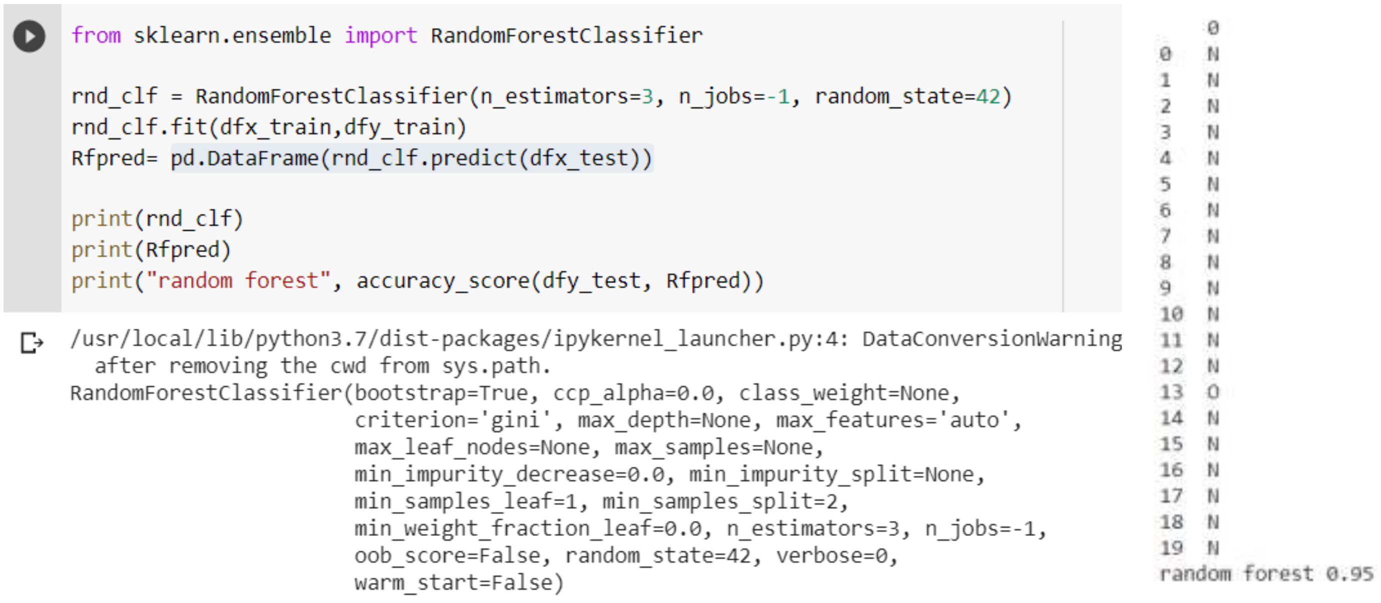


```
[ ] import matplotlib.pyplot as plt
    X = ['N','O']
    Val1=Test_results.value_counts()
    Val2=dfy_test.value_counts()
    print(Val1)
    print(Val2)
    x_axis=np.arange(len(X))
    print(x_axis)

    plt.bar(x_axis - 0.2, Val1,0.4, label="N")
    plt.bar(x_axis + 0.2, Val2,0.4, label="O")
```

```
N    17
O     3
dtype: int64
Output
N         18
O          2
dtype: int64
[0 1]
<BarContainer object of 2 artists>
```



Since a decision tree is considered a weak learner, even variating the parameters didn't improve any better than 85% the accuracy of the program. Therefore, I decided to create a random forest, since this is a set of decision trees it makes it now a strong learner, achieving now a new accuracy of 95%.

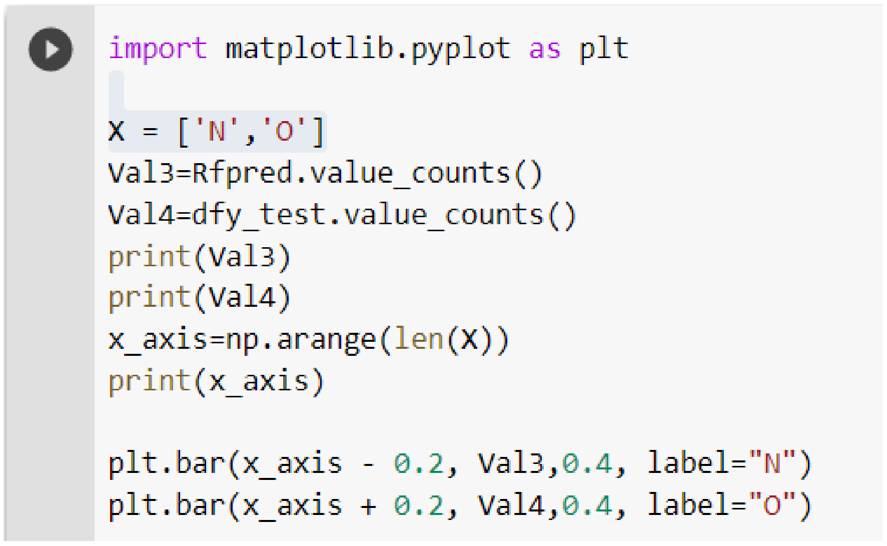


```
from sklearn.ensemble import RandomForestClassifier

rnd_clf = RandomForestClassifier(n_estimators=3, n_jobs=-1, random_state=42)
rnd_clf.fit(dfx_train,dfy_train)
Rfpred= pd.DataFrame(rnd_clf.predict(dfx_test))

print(rnd_clf)
print(Rfpred)
print("random forest", accuracy_score(dfy_test, Rfpred))
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4: DataConversionWarning
  after removing the cwd from sys.path.
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=3, n_jobs=-1,
                       oob_score=False, random_state=42, verbose=0,
                       warm_start=False)
```

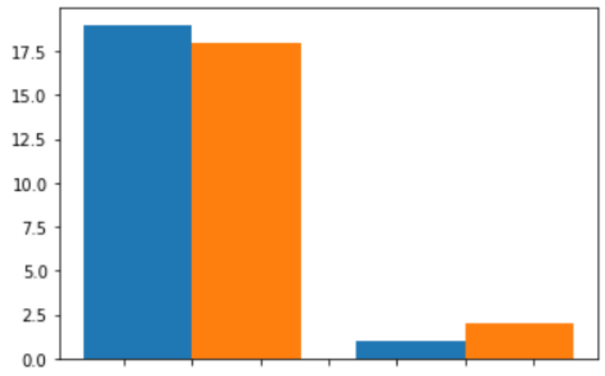
```
    0
0   N
1   N
2   N
3   N
4   N
5   N
6   N
7   N
8   N
9   N
10  N
11  N
12  N
13  O
14  N
15  N
16  N
17  N
18  N
19  N
random forest 0.95
```

Once again, the data once trained was graphed again to visualize better the results and compare the outcomes of the prediction.



```
import matplotlib.pyplot as plt

X = ['N','O']
Val3=Rfpred.value_counts()
Val4=dfy_test.value_counts()
print(Val3)
print(Val4)
x_axis=np.arange(len(X))
print(x_axis)

plt.bar(x_axis - 0.2, Val3,0.4, label="N")
plt.bar(x_axis + 0.2, Val4,0.4, label="O")
```

```
N    19
O     1
dtype: int64
Output
N         18
O          2
dtype: int64
[0 1]
<BarContainer object of 2 artists>
```



Finally, our data is trained, the algorithm is ready to create its own predictions with high accuracy and leave a place for the user to input its own values and the program will calculate the sperm concentration.

For this part, the data was adjusted in order for it being making it more personal by adding the name and more comfortable to the user, specifically on the values for the age and daily hours sitting, the user now inputs the number directly, but the algorithm reads a (0-1) float number being 1 the highest and 0 the lowest. You can see the comparation between the decision tree and random forest prediction.

References

[1] Chertoff, J. (2018) *What is a normal sperm count.* Retrieved from: https://www.healthline.com/health/mens-health/normal-sperm-count

[2] Clinic, M. (2020) *Low sperm count.* Retrieved from: https://www.mayoclinic.org/diseases-conditions/low-sperm-count/symptoms-causes/syc-20374585

[3] Preidt, R. (2018) *Low Sperm Count May Signal Serious Health Risks*. Retrieved from: https://www.webmd.com/infertility-and-reproduction/news/20180319/low-sperm-count-may-signal-serious-health-risks

Link to data set: https://archive.ics.uci.edu/ml/datasets/Fertility