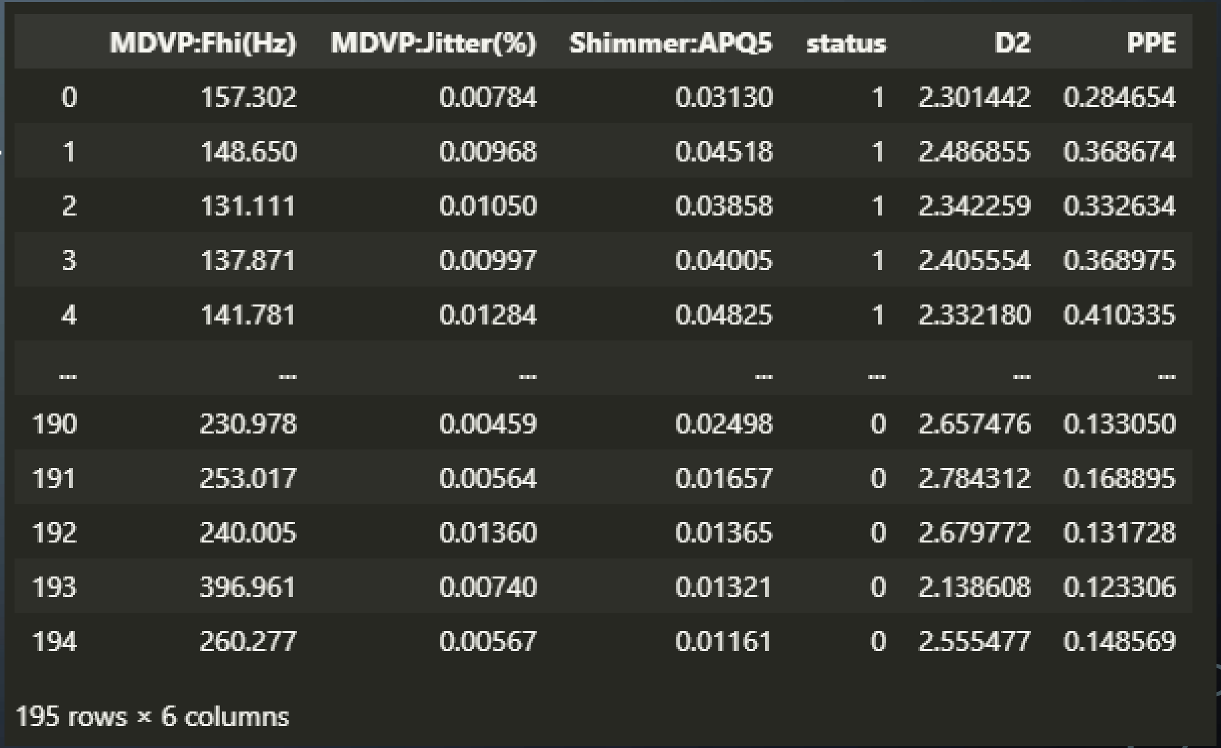
PARKINSON'S DATASET MACHINE LEARNING MODEL

ABOUT THE DATA

- After Importing and reading the data, we see that the data consists of 195 rows and 5 columns. The status column is the target, where a 1 specifies that the patient has Parkinson's and a 0 specifies the patient is healthy. The other columns here are the various features such as MDVP:Fhi(Hz) - Maximum vocal fundamental frequency, MDVP:Jitter(%) - Measure of variation in fundamental frequency, D2 – a nonlinear dynamical complexity measure and PPE -nonlinear measure of fundamental frequency variation.

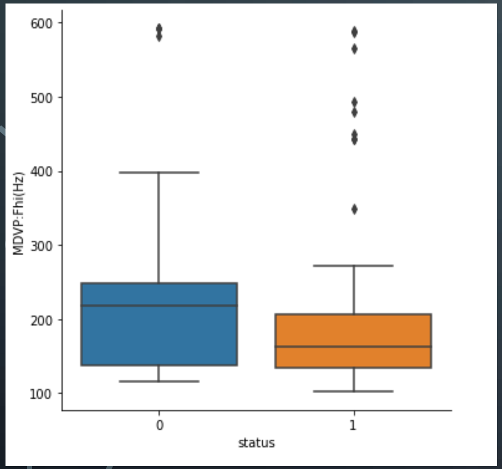


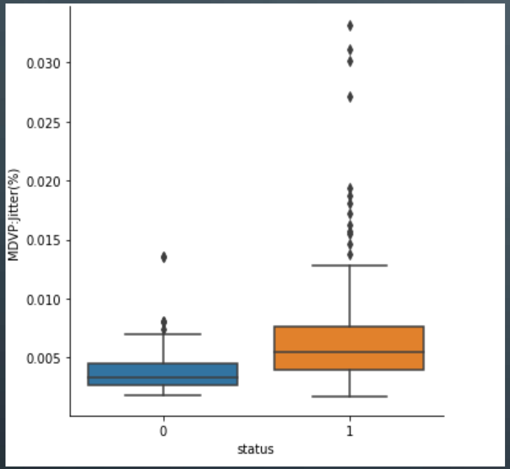
	MDVP:Fhi(Hz)	MDVP:Jitter(%)	Shimmer:APQ5	status	D2	PPE
0	157.302	0.00784	0.03130	1	2.301442	0.284654
1	148.650	0.00968	0.04518	1	2.486855	0.368674
2	131.111	0.01050	0.03858	1	2.342259	0.332634
3	137.871	0.00997	0.04005	1	2.405554	0.368975
4	141.781	0.01284	0.04825	1	2.332180	0.410335
...	...	...	...	...	...	...
190	230.978	0.00459	0.02498	0	2.657476	0.133050
191	253.017	0.00564	0.01657	0	2.784312	0.168895
192	240.005	0.01360	0.01365	0	2.679772	0.131728
193	396.961	0.00740	0.01321	0	2.138608	0.123306
194	260.277	0.00567	0.01161	0	2.555477	0.148569

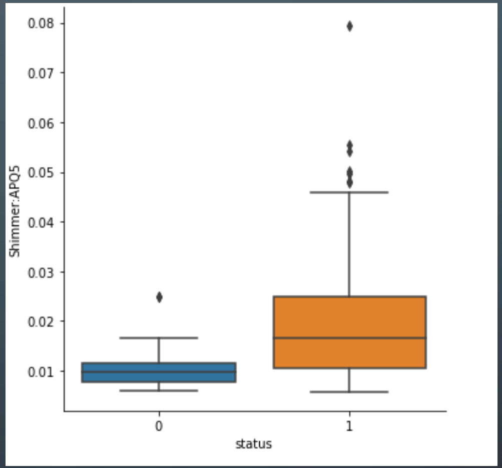
195 rows × 6 columns

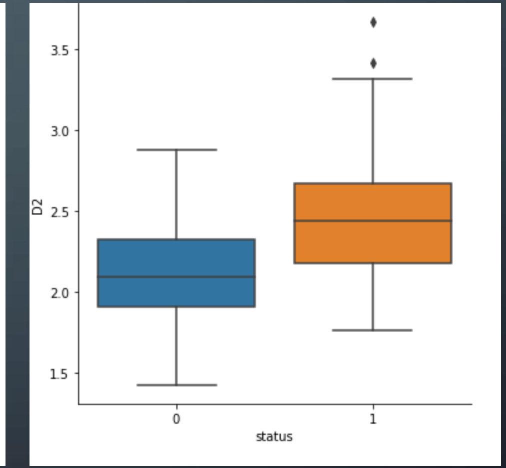
DATA VISUALIZATION AND PREPROCESSING

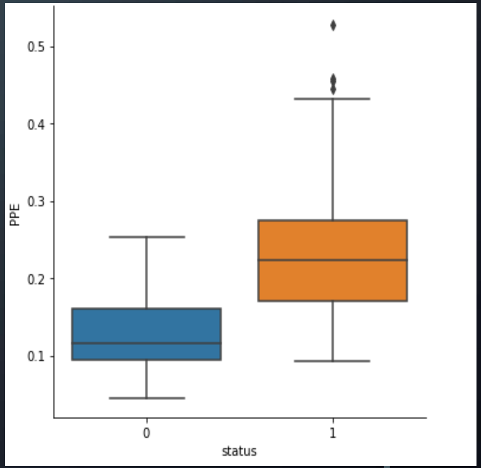
- The data has no missing values upon inspection. So we proceed on visualizing the data. First we plot Boxplots with the x-axis being the target column and y axis being the other features



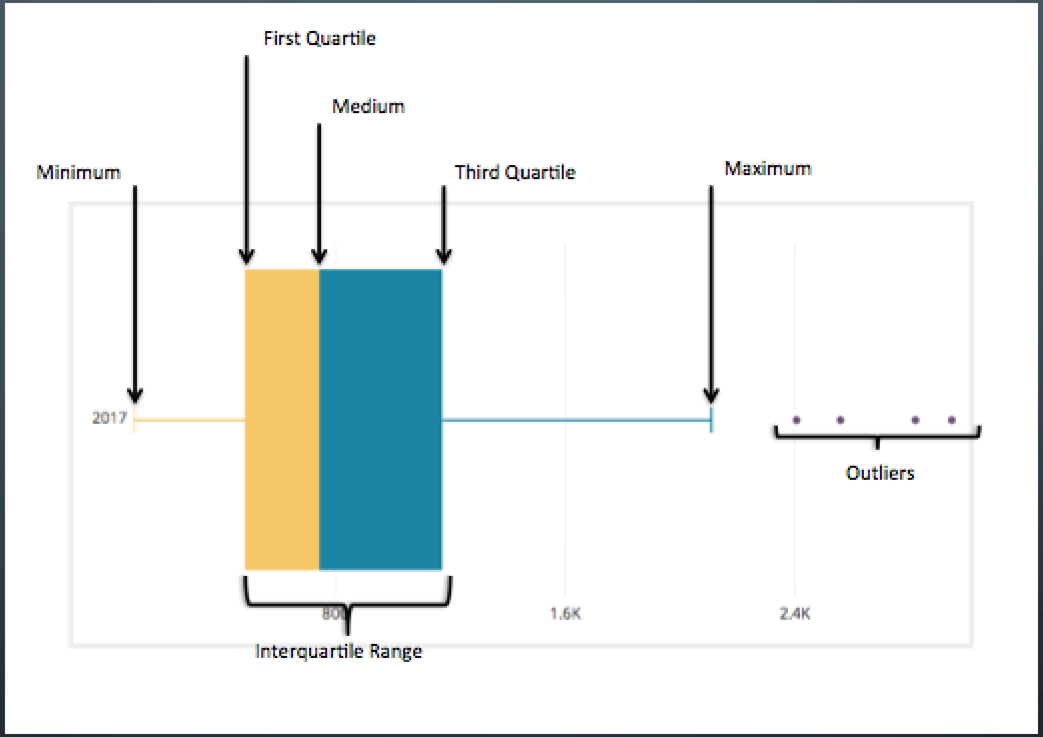






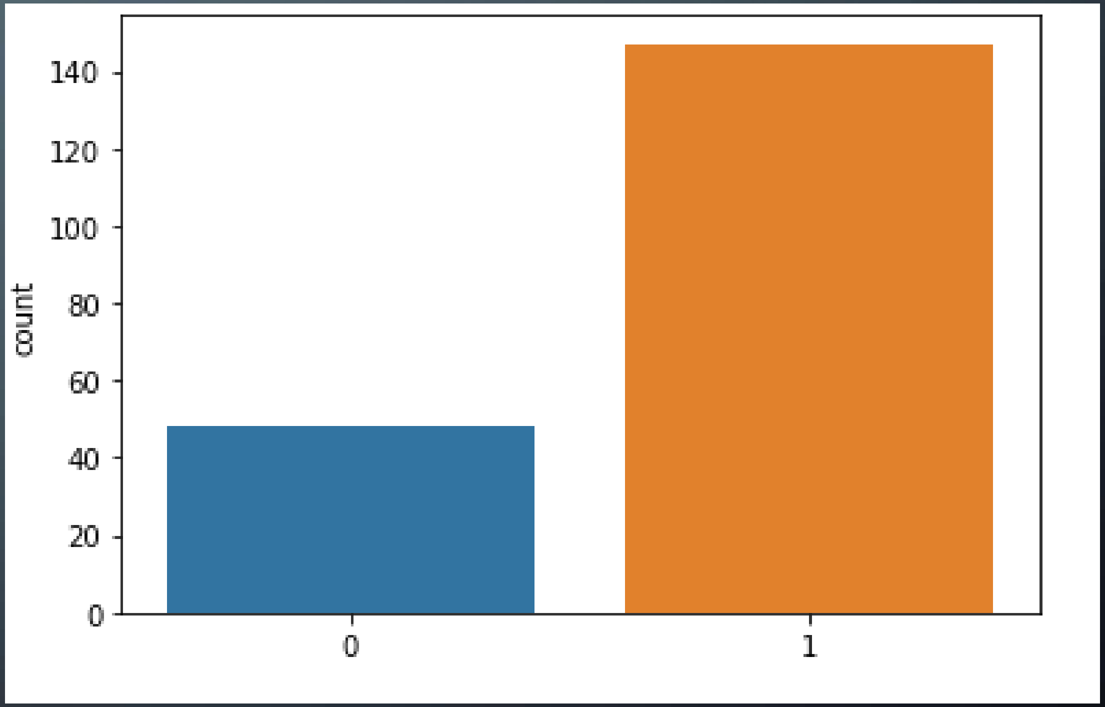


- The boxplots help us visualize the range of the data, outliers present in the data and other properties such as the median the first quartile and the second quartile as well.



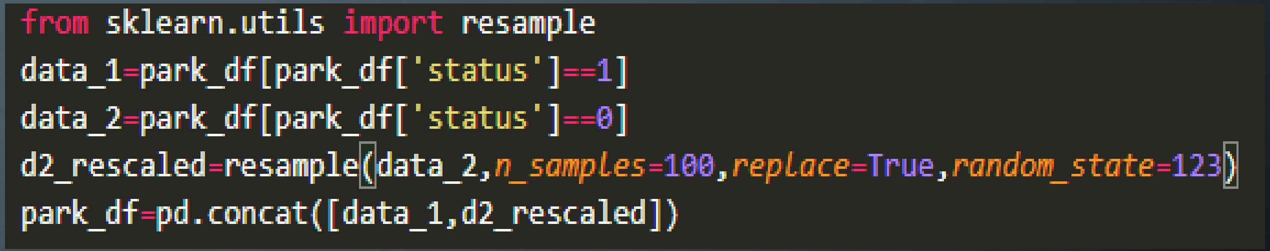
VISUALIZING THE COUNT OF THE STATUS DATA

Upon visualizing the count of the status data, we see that the amount of healthy patients in our data is very low, i.e. majority of the data is of people with Parkinson's. Hence we understand that the data is biased. We need to resample the healthy data so that our model can predict more accurately without the bias



DATA AUGMENTATION

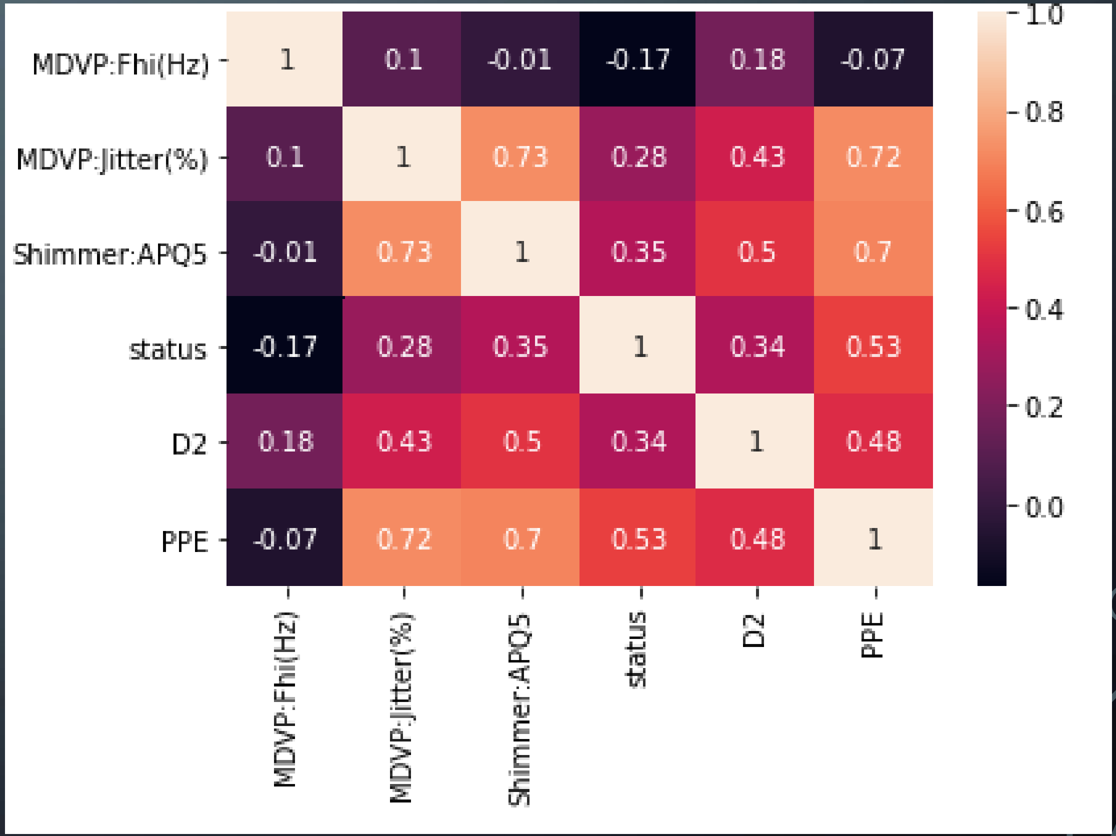
- Using the scikit-learn library, we import the resampling tools, now we divide the data so that we get rows with status 1 and 0 separately. After that we resample the status 0 rows such that a 100 samples of it can be produced to avoid the biasing. Then we concatenate both of the data into a single pandas dataframe.



```
from sklearn.utils import resample
data_1=park_df[park_df['status']==1]
data_2=park_df[park_df['status']==0]
d2_rescaled=resample(data_2,n_samples=100,replace=True,random_state=123)
park_df=pd.concat([data_1,d2_rescaled])
```

CORRELATION MATRIX

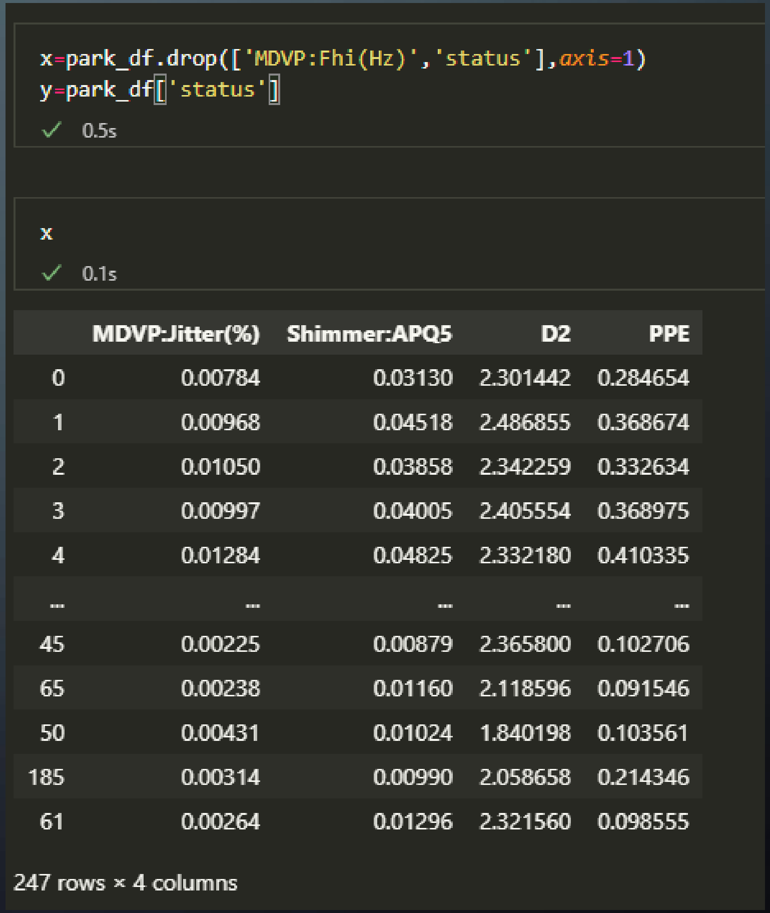
- Using the seaborn library, we plot a heatmap of the correlation matrix with highly correlated items being almost white and low correlated items being dark purple. We don’t see cases of multicollinearity in this data hence we just check the row of correlations for the status column and we find that MDVP:Fhi(Hz) shows very low correlation which signifies it has a very low impact on our target.



	MDVP:Fhi(Hz)	MDVP:Jitter(%)	Shimmer:APQ5	status	D2	PPE
MDVP:Fhi(Hz)	1	0.1	-0.01	-0.17	0.18	-0.07
MDVP:Jitter(%)	0.1	1	0.73	0.28	0.43	0.72
Shimmer:APQ5	-0.01	0.73	1	0.35	0.5	0.7
status	-0.17	0.28	0.35	1	0.34	0.53
D2	0.18	0.43	0.5	0.34	1	0.48
PPE	-0.07	0.72	0.7	0.53	0.48	1

PREPARING INPUT AND OUTPUT DATA

- As we found out in the previous slide, MDVP:Fhi(Hz) shows very low correlation with the status, hence we remove that and the status column for the x dataframe and make the status column the value of y.
- Now, we have to split our data into training and testing data.



```
x=park_df.drop(['MDVP:Fhi(Hz)','status'],axis=1)
y=park_df['status']
✓ 0.5s
```

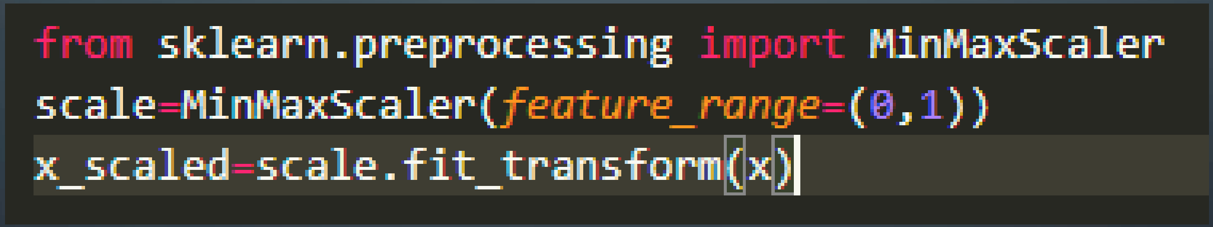
```
x
✓ 0.1s
```

	MDVP:Jitter(%)	Shimmer:APQ5	D2	PPE
0	0.00784	0.03130	2.301442	0.284654
1	0.00968	0.04518	2.486855	0.368674
2	0.01050	0.03858	2.342259	0.332634
3	0.00997	0.04005	2.405554	0.368975
4	0.01284	0.04825	2.332180	0.410335
...	...	...	...	...
45	0.00225	0.00879	2.365800	0.102706
65	0.00238	0.01160	2.118596	0.091546
50	0.00431	0.01024	1.840198	0.103561
185	0.00314	0.00990	2.058658	0.214346
61	0.00264	0.01296	2.321560	0.098555

247 rows × 4 columns

SCALING THE DATA

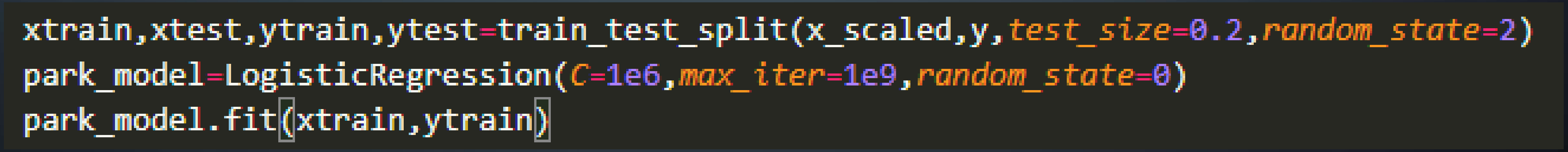
- Before preparing the training and testing models, we have scaled the data so that features with more higher magnitude do not weigh in more than the other features.



```
from sklearn.preprocessing import MinMaxScaler
scale=MinMaxScaler(feature_range=(0,1))
x_scaled=scale.fit_transform(x)
```

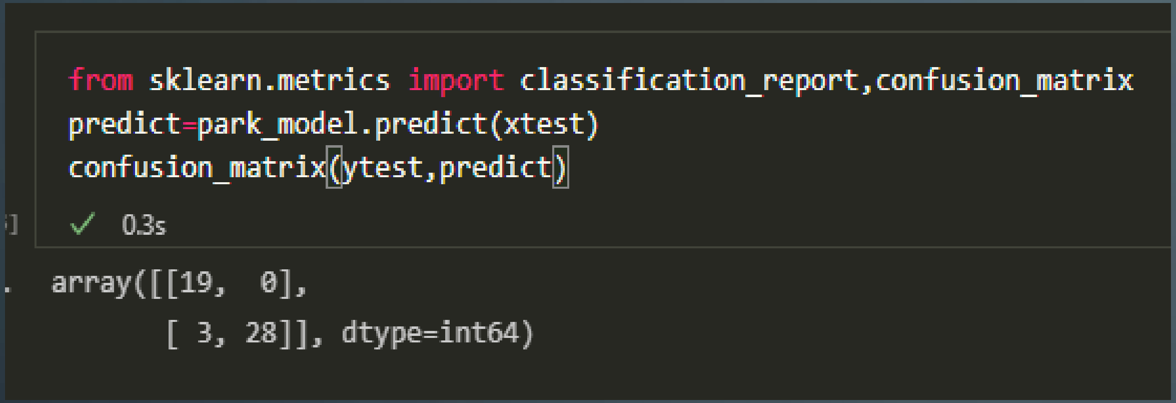
TRAINING AND TESTING DATA

- Now we split the data such that we get two sets of data, one for training the model and the other to test the model. First we split, then we fit the training data into the model. We first prepare the Logistic Regression model.

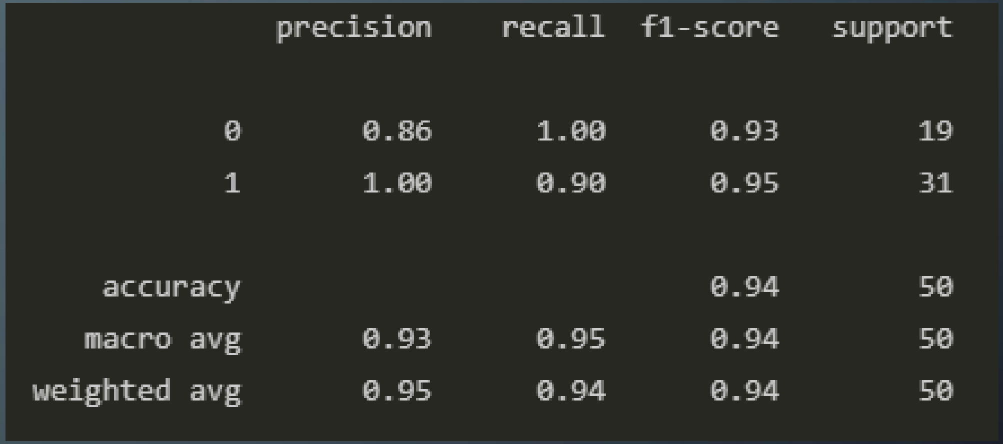


```
xtrain,xtest,ytrain,ytest=train_test_split(x_scaled,y,test_size=0.2,random_state=2)
park_model=LogisticRegression(C=1e6,max_iter=1e9,random_state=0)
park_model.fit(xtrain,ytrain)
```

CLASSIFICATION REPORT AND CONFUSION MATRIX OF THE LOGISTIC REGRESSION MODEL



```
from sklearn.metrics import classification_report,confusion_matrix
predict=park_model.predict(xtest)
confusion_matrix(ytest,predict)
✓ 0.3s
array([[19,  0],
       [ 3, 28]], dtype=int64)
```

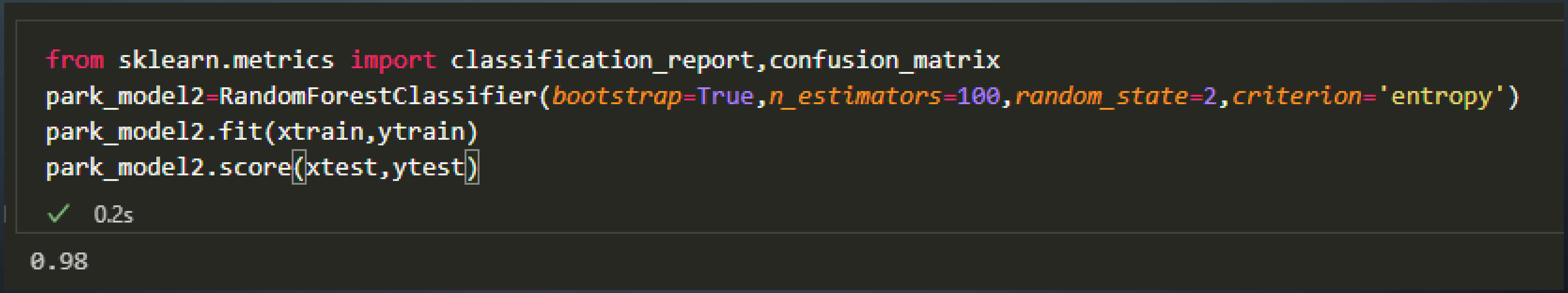


```
              precision    recall  f1-score   support

           0       0.86      1.00      0.93        19
           1       1.00      0.90      0.95        31

    accuracy                           0.94        50
   macro avg       0.93      0.95      0.94        50
weighted avg       0.95      0.94      0.94        50
```

- The Confusion matrix gives us the number of false positives, false negatives, true positives and true negatives our model has produced, whereas the classification report gives us a detailed report showing us the accuracy of our model, f1-score, recall etc.

RANDOM FOREST CLASSIFICATION MODEL

- Next we prepare the random forest classification model using the same sets of data we used for logistic regression.

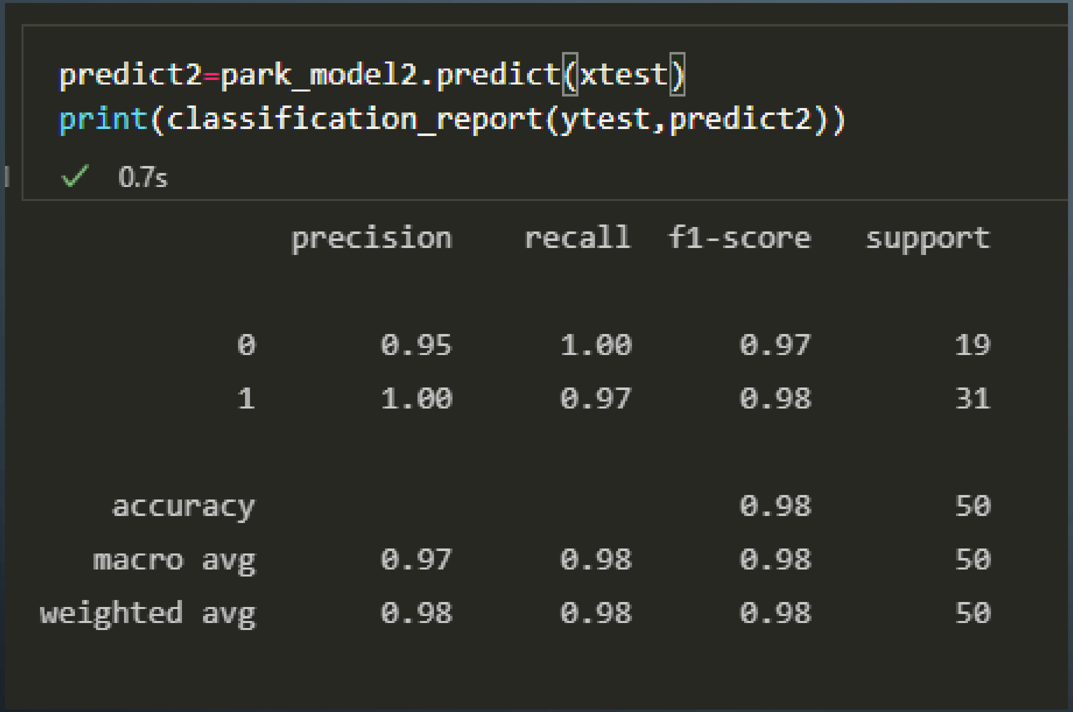


```
from sklearn.metrics import classification_report,confusion_matrix
park_model2=RandomForestClassifier(bootstrap=True,n_estimators=100,random_state=2,criterion='entropy')
park_model2.fit(xtrain,ytrain)
park_model2.score(xtest,ytest)
```

✓ 0.2s

```
0.98
```

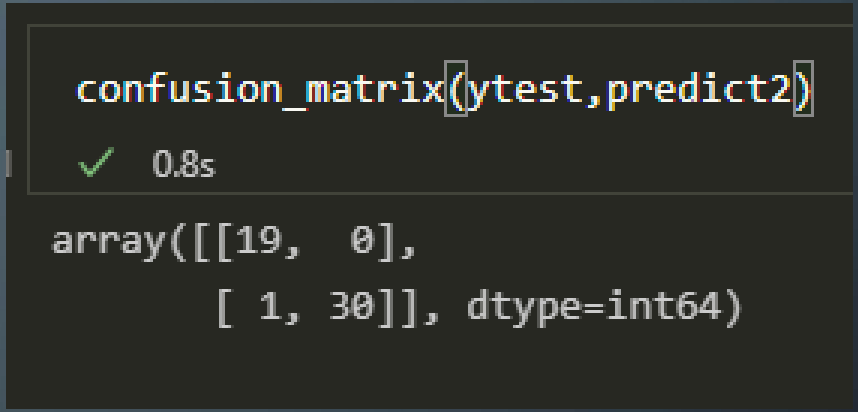
CLASSIFICATION REPORT AND CONFUSION MATRIX OF THE RANDOM FOREST CLASSIFICATION MODEL



```
predict2=park_model2.predict(xtest)
print(classification_report(ytest,predict2))
```

✓ 0.7s

```
              precision    recall  f1-score   support

           0       0.95      1.00      0.97        19
           1       1.00      0.97      0.98        31

    accuracy                           0.98        50
   macro avg       0.97      0.98      0.98        50
weighted avg       0.98      0.98      0.98        50
```



```
confusion_matrix(ytest,predict2)
```

✓ 0.8s

```
array([[19,  0],
       [ 1, 30]], dtype=int64)
```