Combined Cycle Power Plant
Dataset: Regression
Source: https://archive.ics.uci.edu/dataset/294/combined+cycle+power+plant

Maternal Health Risk
Dataset: Classification
Source: https://archive.ics.uci.edu/dataset/863/maternal+health+risk

1. Regression

1.1. Objectives

The primary goal is to predict the net hourly electrical energy output (EP) of a Combined Cycle Power Plant (CCPP) based on variables like temperature, ambient pressure, relative humidity, and exhaust vacuum. By predicting the EP correctly will enable us to optimise the plant's operations, improve efficiency, and potentially reduce costs.

1.2. Data Exploration

Features: 04

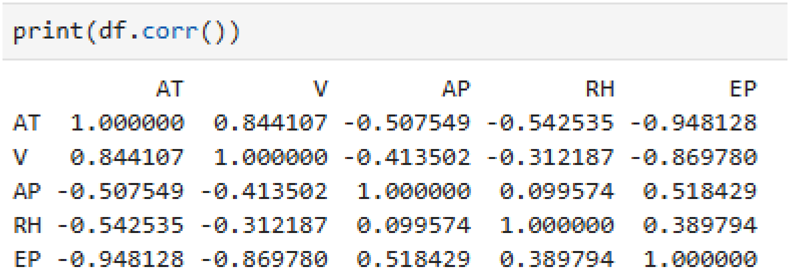
i. **Temperature (T):** Ambient temperature, which affects the performance of gas turbines.
ii. **Ambient Pressure (AP):** Atmospheric pressure, which can impact the efficiency of the plant.
iii. **Relative Humidity (RH):** Moisture content in the air, which can affect the performance of gas turbines.
iv. **Exhaust Vacuum (V):** Vacuum level in the steam turbine, which affects its efficiency.

Target Variable: **01**

Net Hourly Electrical **Energy Output (EP):** The amount of electricity generated by the plant per hour.

Note: In dataset file, column name is 'PE' instead of 'EP'. There are 5 sheets with exact same data, I chose 'Sheet5' and converted it to CSV file.

- Total rows = 9568
- Dataset is clean with no missing values
- Correlation results are showing high correlation between 'AT' and 'V' while others are showing moderate correlation but on the negative side.



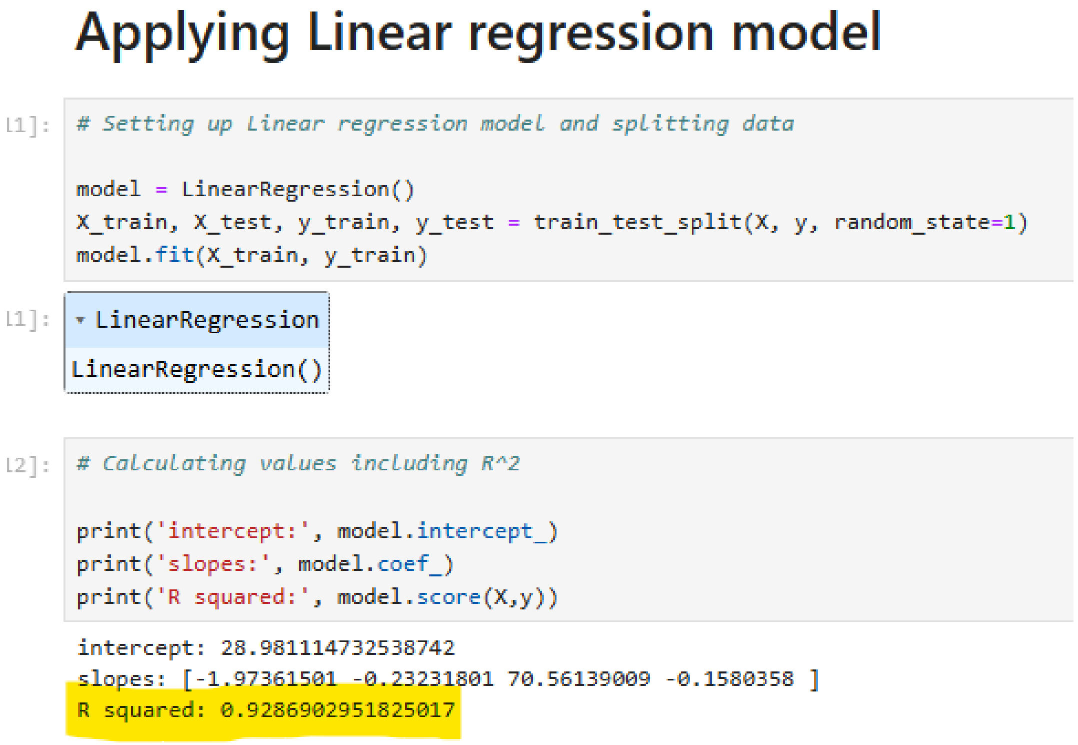
```
print(df.corr())
```

```
          AT         V        AP        RH        EP
AT  1.000000  0.844107 -0.507549 -0.542535 -0.948128
V   0.844107  1.000000 -0.413502 -0.312187 -0.869780
AP -0.507549 -0.413502  1.000000  0.099574  0.518429
RH -0.542535 -0.312187  0.099574  1.000000  0.389794
EP -0.948128 -0.869780  0.518429  0.389794  1.000000
```

1.3. Modelling

As the dataset is multivariate, the best model for predicting correct EP would be Multi Linear Regression or LinearRegression.

- Performed split for Training and Test data
- Injected training data using ‘fit’ method



Applying Linear regression model

```
[1]: # Setting up Linear regression model and splitting data

     model = LinearRegression()
     X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
     model.fit(X_train, y_train)
```

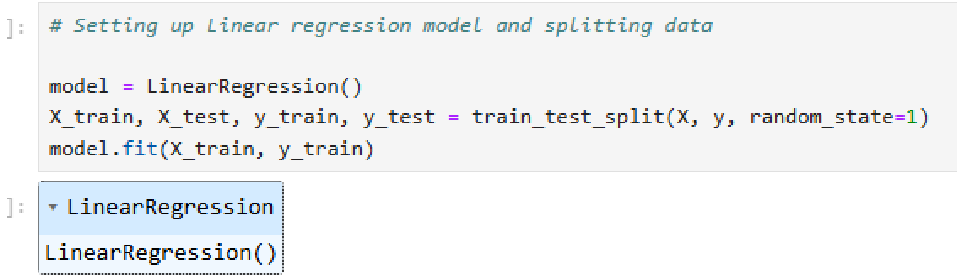
```
[1]: ▾ LinearRegression
     LinearRegression()
```

```
[2]: # Calculating values including R^2

     print('intercept:', model.intercept_)
     print('slopes:', model.coef_)
     print('R squared:', model.score(X,y))
```

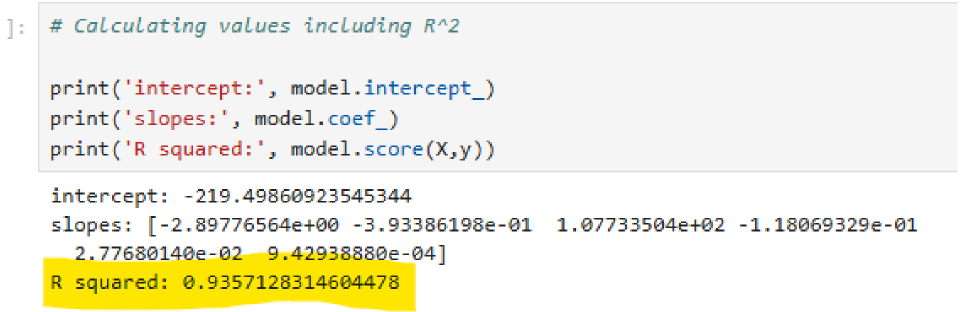
```
intercept: 28.981114732538742
slopes: [-1.97361501 -0.23231801 70.56139009 -0.1580358 ]
R squared: 0.9286902951825017
```

Second-degree Polynomial:

We tested by adding 2-features (AT, V) with polynomial terms to see if performance of the model improves further.



```
]: # Setting up Linear regression model and splitting data

   model = LinearRegression()
   X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=1)
   model.fit(X_train, y_train)

]: ▾ LinearRegression
   LinearRegression()
```

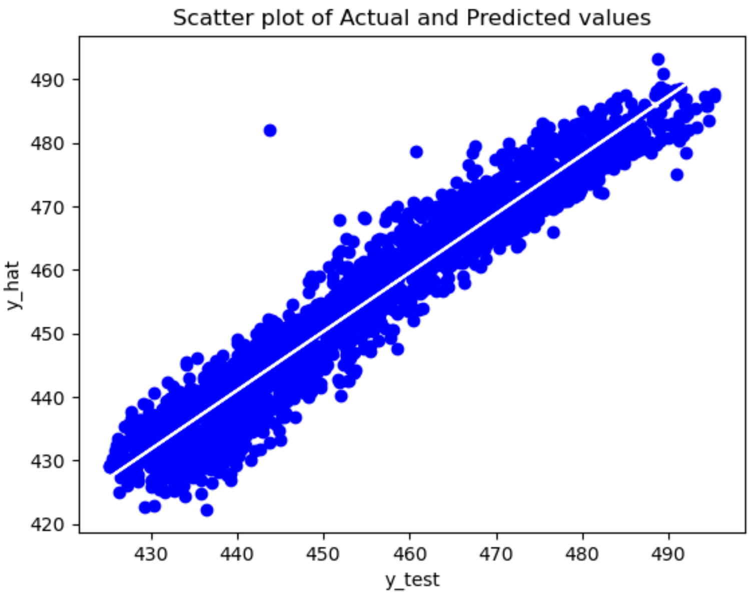


```
]: # Calculating values including R^2

   print('intercept:', model.intercept_)
   print('slopes:', model.coef_)
   print('R squared:', model.score(X,y))

   intercept: -219.49860923545344
   slopes: [-2.89776564e+00 -3.93386198e-01  1.07733504e+02 -1.18069329e-01
     2.77680140e-02  9.42938880e-04]
   R squared: 0.9357128314604478
```

1.4. Evaluation

Getting R2 ~93% which is reasonably good. We can see the negative coefficient for AT, V and RH whereas it's positive for AP.

- MSE is 4.48
- RMSE is 2.11
- Scatter plot also confirms strong prediction results on test data.
- This model can accurately predict EP value based on features (AT, V, AP, RH)



Second-degree Polynomial:

- R^2 93.5% has gone up from 92.8% without polynomial
- MSE too has dropped a bit, from 4.48 to 4.27

2. Decision Trees

2.1. Objectives

The provided dataset aims to address the challenge of maternal mortality, which is a significant concern within the UN's Sustainable Development Goals (SDGs).

Goals:

- Develop a risk assessment model to predict the potential for maternal health complications.
- Identify important factors affecting maternal health risks level using the collected attributes (age, blood pressure, blood sugar, body temperature, heart rate).

2.2. Data Exploration

Features: 06

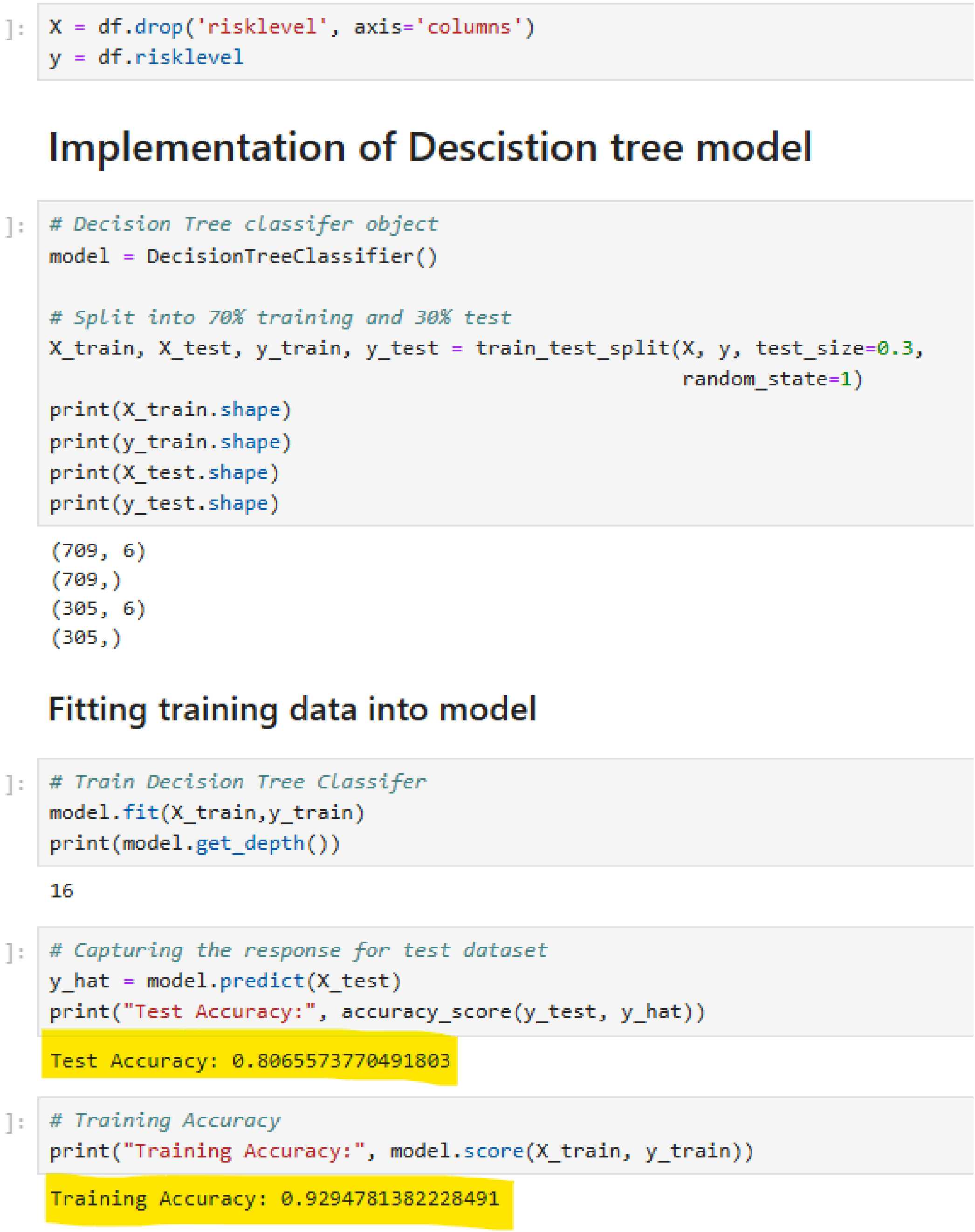
i. Age
ii. SystolicBP
iii. DiastolicBP
iv. Blood Sugar(BS)
v. BodyTemp
vi. HeartRate

Target classes: 03

RiskLevel (high risk, mid risk, low risk)

- Total rows = 1014
- Dataset is clean with no missing values
- Converted column names into lowercase for simplification

2.3. Modelling



```
X = df.drop('risklevel', axis='columns')
y = df.risklevel
```

Implementation of Descistion tree model

```
# Decision Tree classifer object
model = DecisionTreeClassifier()

# Split into 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
                                                    random_state=1)
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(709, 6)
(709,)
(305, 6)
(305,)
```

Fitting training data into model

```
# Train Decision Tree Classifer
model.fit(X_train,y_train)
print(model.get_depth())
```

```
16
```

```
# Capturing the response for test dataset
y_hat = model.predict(X_test)
print("Test Accuracy:", accuracy_score(y_test, y_hat))
```

```
Test Accuracy: 0.8065573770491803
```

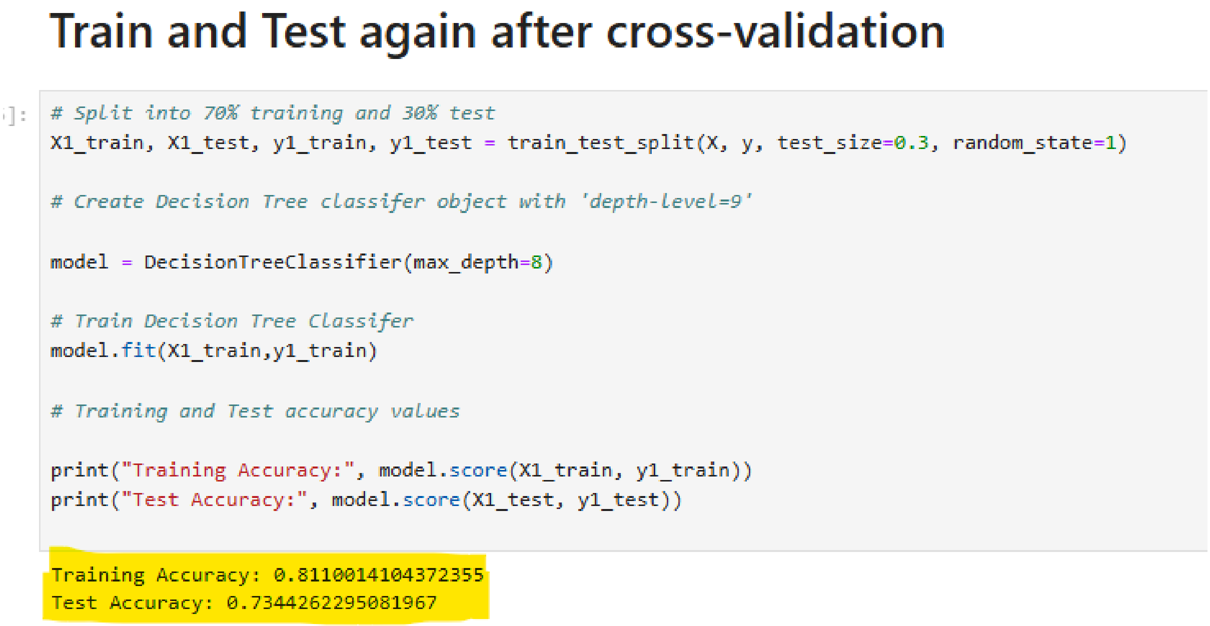
```
# Training Accuracy
print("Training Accuracy:", model.score(X_train, y_train))
```

```
Training Accuracy: 0.9294781382228491
```

2.4. Evaluation

We can clearly see the model is performing very well on training data whereas moderately good on test data.

- Results pointing towards ‘overfitting’
- Performed cross validation for ‘max_depth’ value ranging from 2-20. Cross validation results are showing typical behaviour of Decision Tree model.
- When the tree grows deeper, accuracy level goes high as well but this wouldn’t be helpful when predicting unseen data → Test dataset
- When trained and tested again using ‘max_depth=8’ the difference between training and test accuracy was reduced reasonably.

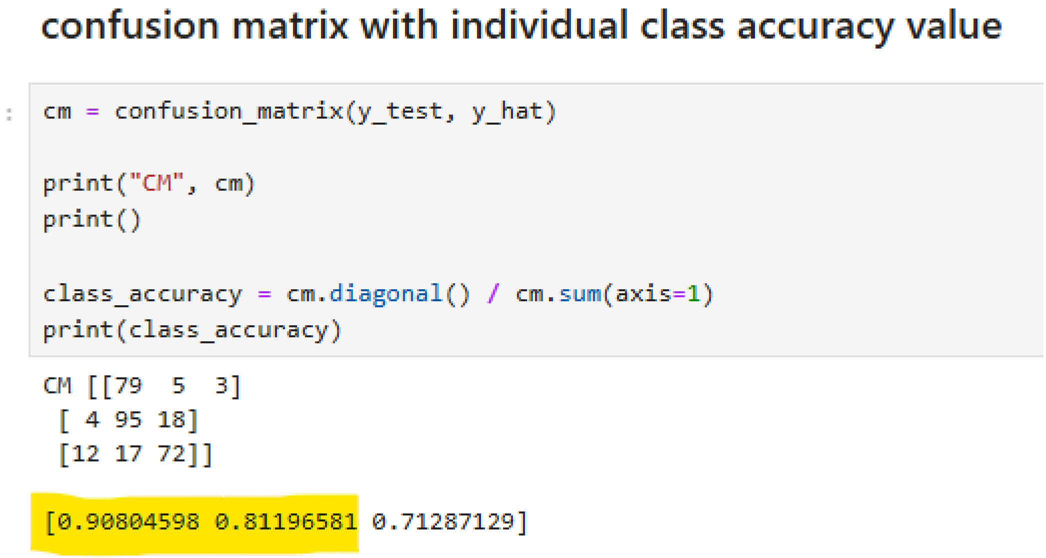


Train and Test again after cross-validation

```
# Split into 70% training and 30% test
X1_train, X1_test, y1_train, y1_test = train_test_split(X, y, test_size=0.3, random_state=1)

# Create Decision Tree classifer object with 'depth-level=9'

model = DecisionTreeClassifier(max_depth=8)

# Train Decision Tree Classifer
model.fit(X1_train,y1_train)

# Training and Test accuracy values

print("Training Accuracy:", model.score(X1_train, y1_train))
print("Test Accuracy:", model.score(X1_test, y1_test))
```

```
Training Accuracy: 0.8110014104372355
Test Accuracy: 0.7344262295081967
```

- This Decision Tree model is performing well for multi-class predictions. This further can be improved by implementing 'post-pruning'.



confusion matrix with individual class accuracy value

```
cm = confusion_matrix(y_test, y_hat)

print("CM", cm)
print()

class_accuracy = cm.diagonal() / cm.sum(axis=1)
print(class_accuracy)
```

```
CM [[79  5  3]
 [ 4 95 18]
 [12 17 72]]

[0.90804598 0.81196581 0.71287129]
```

3. kNN

3.1. Objectives

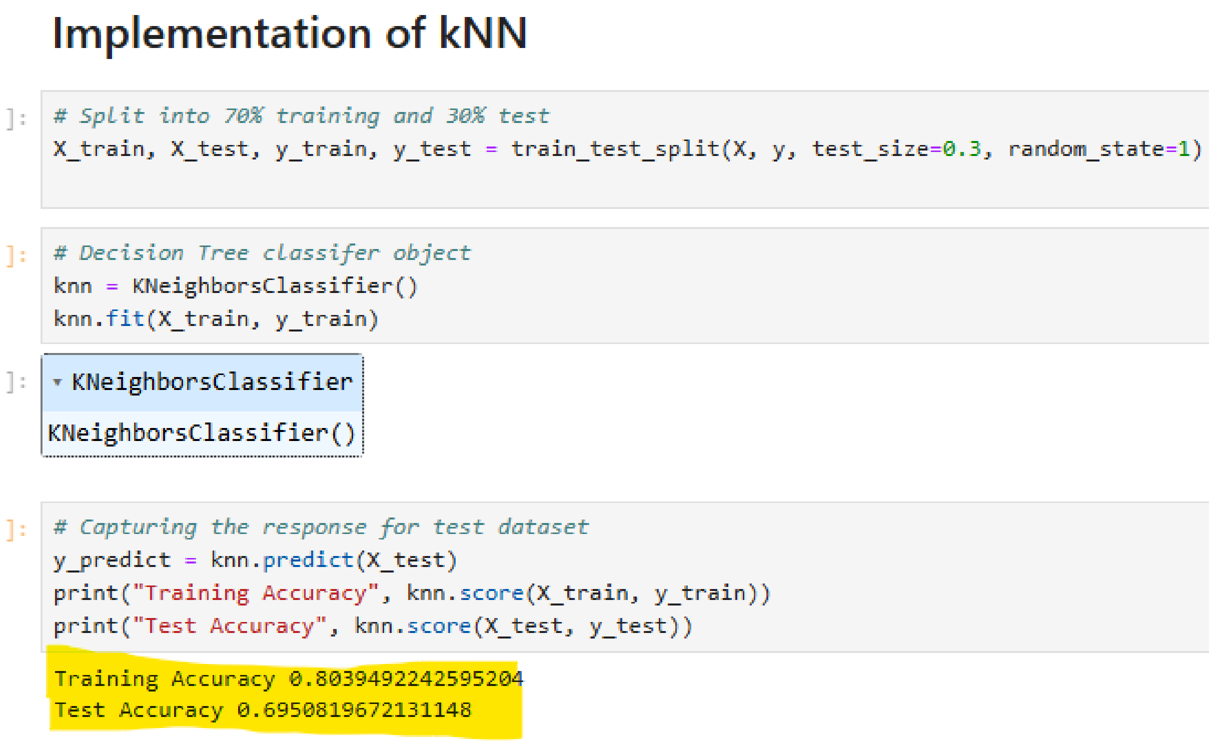
The primary goal here is to assess the usability of kNN to predict and classify risk levels associated with maternal health based on certain features ((age, blood pressure, blood sugar, body temperature, heart rate).

3.2. Data Exploration

We used the same dataset 'maternal health risk' which we used previously for Decision Tree classifiers.

3.3. Modelling

We implemented kNN classifier by splitting data into 70/30 as 30% for testing without any fixed 'k' value.



Implementation of kNN

```
# Split into 70% training and 30% test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)
```

```
# Decision Tree classifer object
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
```

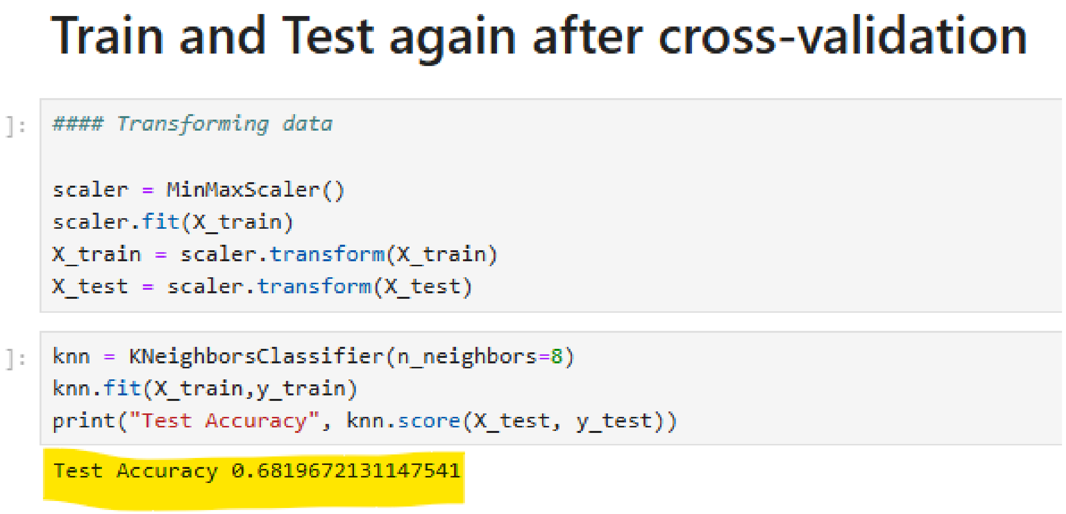
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
# Capturing the response for test dataset
y_predict = knn.predict(X_test)
print("Training Accuracy", knn.score(X_train, y_train))
print("Test Accuracy", knn.score(X_test, y_test))
```

```
Training Accuracy 0.8039492242595204
Test Accuracy 0.6950819672131148
```

3.4. Evaluation

- The results of training and test accuracy were 80% and 69.5% respectively.
- This means kNN is performing worse as compared to Decision Tree model, at least with the current dataset
- After running cross-validation and performing training and test again with 'k' value 8
- Test accuracy hasn't been improved
- After comparing kNN results with Decision Tree model, it would be recommended to use Decision Tree model for current dataset as Decision Tree is showing far better results than kNN.



Train and Test again after cross-validation

```
#### Transforming data

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
knn = KNeighborsClassifier(n_neighbors=8)
knn.fit(X_train,y_train)
print("Test Accuracy", knn.score(X_test, y_test))
```

```
Test Accuracy 0.6819672131147541
```