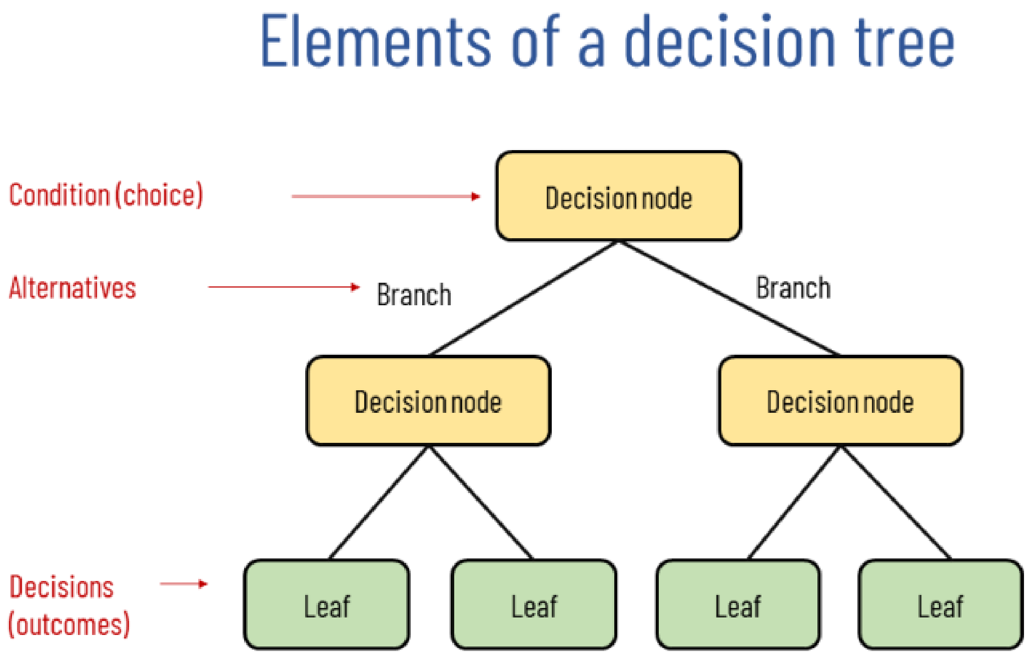
Random Forest

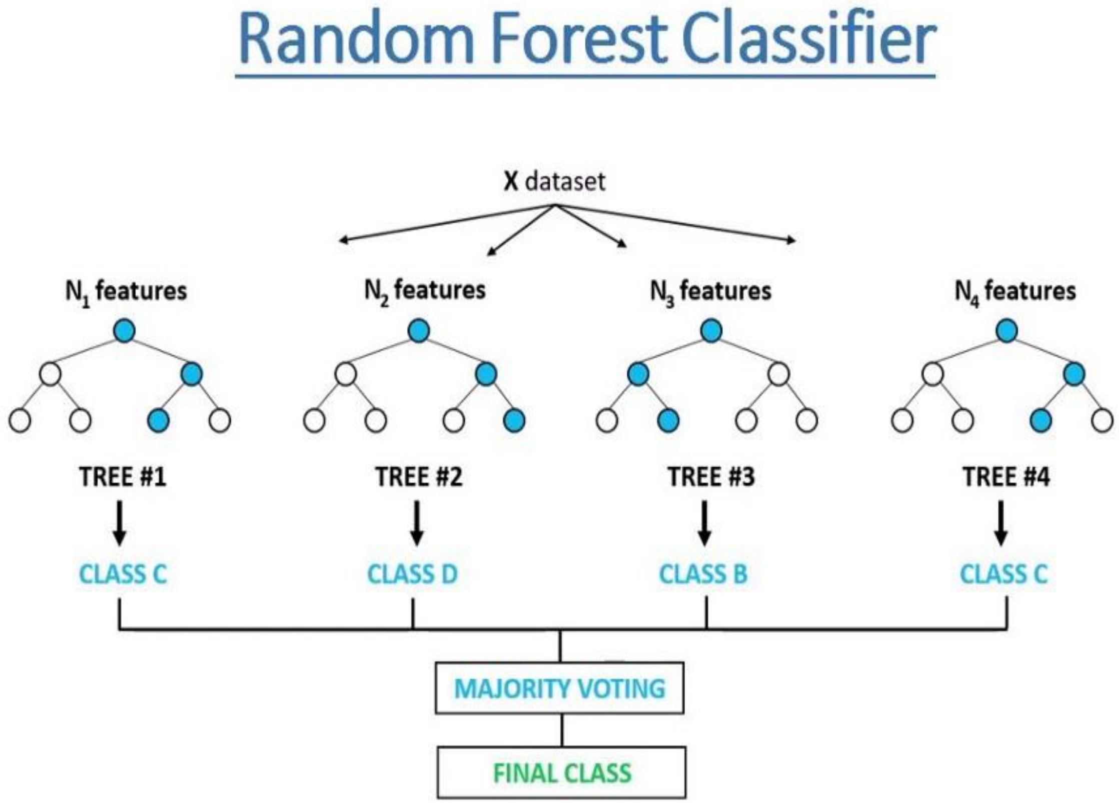
(Decision Trees)

NATHAN VAN SCHYNDEL, HAYDEN MUSCHA, AZURE ELLER, VICTOR PHAM

Random Forest

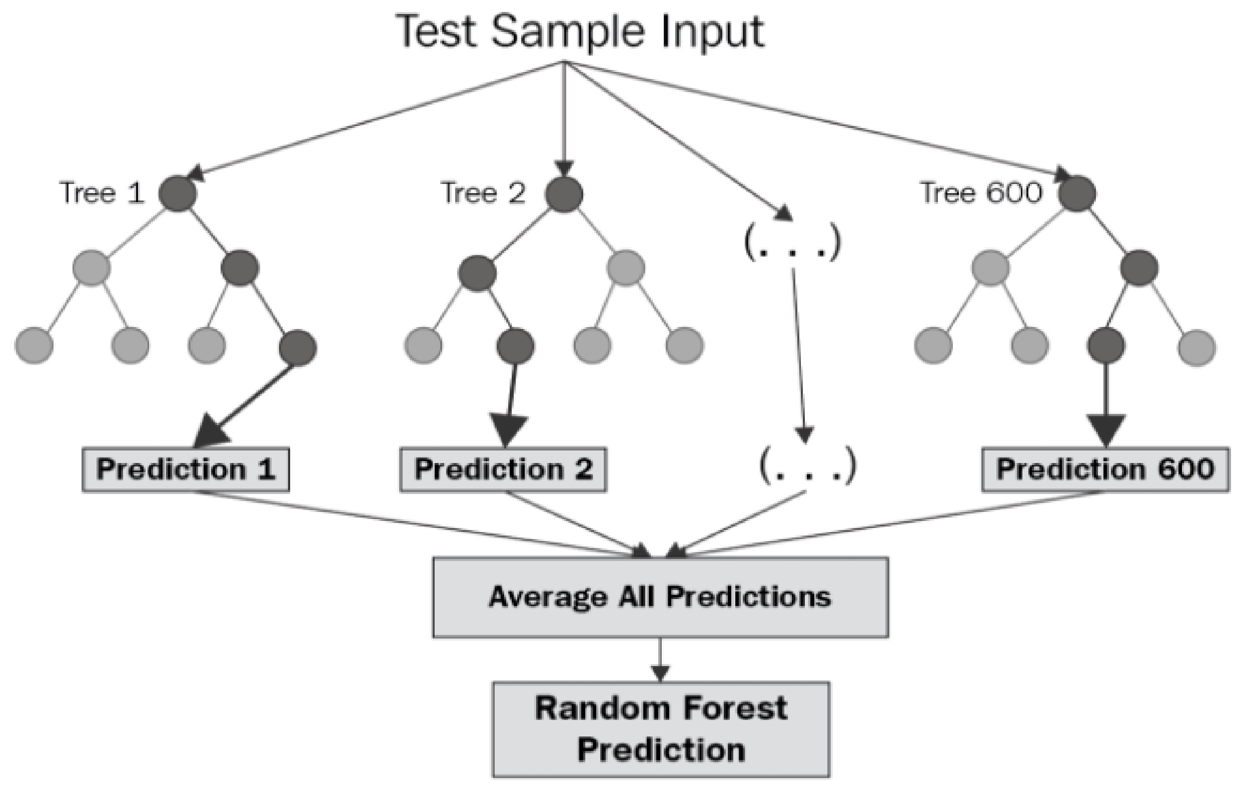
- Root Node
- Splitting
- Leaf
- Branch
- Tree





Forest Structure

- Number of trees
- Depth
- Voting Principle
  - Gini Impurity



Random Forest

- Statistical decisions

Forest: Class Comparison

- Supervised
- Non-Linear or Linear
- Categorical and Continuous Specialty

Random Forest: Advantages

- Reduces Overfitting
- Flexible
- Categorical and Continuous
- Fills missing values
- No Normalization

Random Forest: Disadvantages

- Computationally heavy
- Time intensive
- Interpretability and variable significance

Data processing steps

- No Standardization
- Deal with null values, create dummy variables if necessary
- Split data from target variable
- Check feature importance

Hyperparameters: Regression

n_estimators - int, default=100

criterion - squared error, absolute error, poisson - default = squared error

max_depth- int, default=None

min_samples_split - int or float, default=2

min_samples_leaf - int or float, default=1

min_weight_fraction_leaf - float, default=0.0

max_features - {“sqrt”, “log2”, None}, int or float, default=1.0

max_leaf_nodes - int, default=None

min_impurity_decrease - float, default=0.0

bootstrap - bool, default=True

Hyperparameters: Classification

n_estimators - int, default=100

criterion - gini, entropy, log_loss, default = gini

max_depth- int, default=None

min_samples_split - int or float, default=2

min_samples_leaf - int or float, default=1

min_weight_fraction_leaf - float, default=0.0

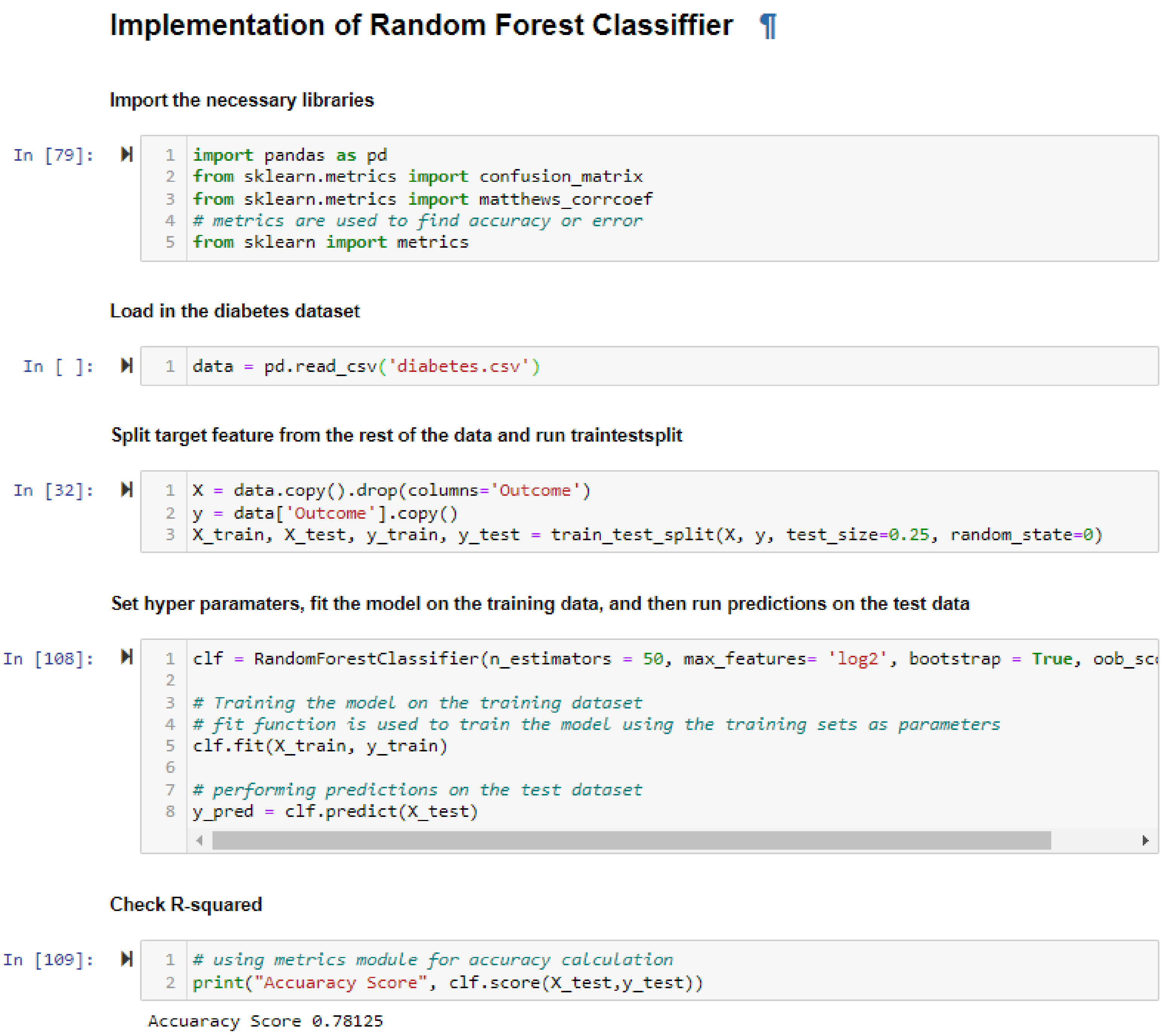
max_features - {“sqrt”, “log2”, None}, int or float, default=sqrt

max_leaf_nodes - int, default=None

min_impurity_decrease - float, default=0.0

bootstrap - bool, default=True

Sample Code: Random Forest Classifier on Diabetes Dataset



Implementation of Random Forest Classiffier ¶

Import the necessary libraries

In [79]:

```
import pandas as pd
from sklearn.metrics import confusion_matrix
from sklearn.metrics import matthews_corrcoef
# metrics are used to find accuracy or error
from sklearn import metrics
```

Load in the diabetes dataset

In [ ]:

```
data = pd.read_csv('diabetes.csv')
```

Split target feature from the rest of the data and run traintestsplit

In [32]:

```
X = data.copy().drop(columns='Outcome')
y = data['Outcome'].copy()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

Set hyper paramaters, fit the model on the training data, and then run predictions on the test data

In [108]:

```
clf = RandomForestClassifier(n_estimators = 50, max_features= 'log2', bootstrap = True, oob_sc

# Training the model on the training dataset
# fit function is used to train the model using the training sets as parameters
clf.fit(X_train, y_train)

# performing predictions on the test dataset
y_pred = clf.predict(X_test)
```

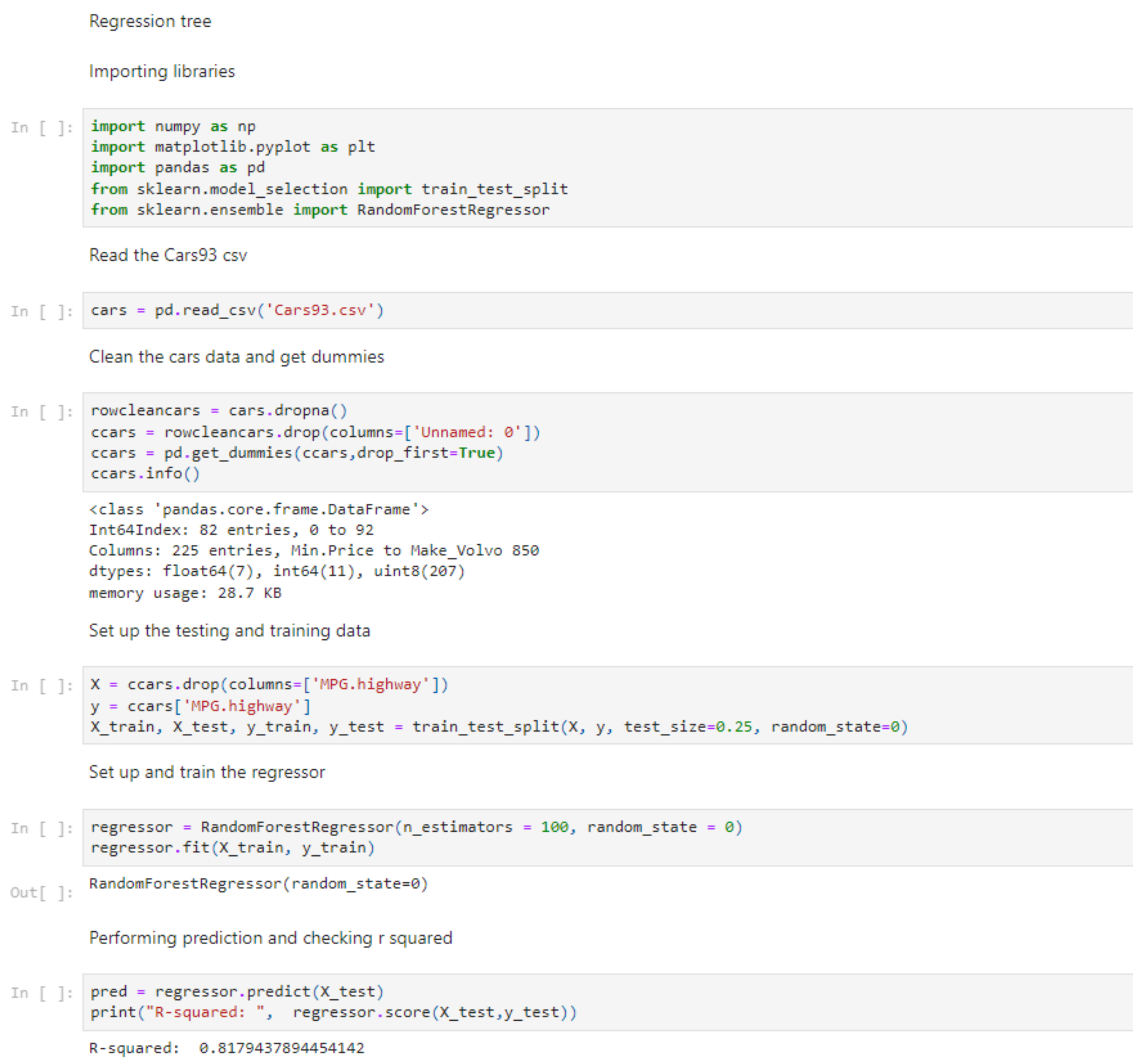
Check R-squared

In [109]:

```
# using metrics module for accuracy calculation
print("Accuaracy Score", clf.score(X_test,y_test))
```

```
Accuaracy Score 0.78125
```

Sample Code: Random Forest Regression Tree on Cars93



Regression tree

Importing libraries

```
In [ ]: import numpy as np
        import matplotlib.pyplot as plt
        import pandas as pd
        from sklearn.model_selection import train_test_split
        from sklearn.ensemble import RandomForestRegressor
```

Read the Cars93 csv

```
In [ ]: cars = pd.read_csv('Cars93.csv')
```

Clean the cars data and get dummies

```
In [ ]: rowcleancars = cars.dropna()
        ccars = rowcleancars.drop(columns=['Unnamed: 0'])
        ccars = pd.get_dummies(ccars,drop_first=True)
        ccars.info()

        <class 'pandas.core.frame.DataFrame'>
        Int64Index: 82 entries, 0 to 92
        Columns: 225 entries, Min.Price to Make_Volvo 850
        dtypes: float64(7), int64(11), uint8(207)
        memory usage: 28.7 KB
```

Set up the testing and training data

```
In [ ]: X = ccars.drop(columns=['MPG.highway'])
        y = ccars['MPG.highway']
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=0)
```

Set up and train the regressor

```
In [ ]: regressor = RandomForestRegressor(n_estimators = 100, random_state = 0)
        regressor.fit(X_train, y_train)

Out[ ]: RandomForestRegressor(random_state=0)
```

Performing prediction and checking r squared

```
In [ ]: pred = regressor.predict(X_test)
        print("R-squared: ",  regressor.score(X_test,y_test))

        R-squared:  0.8179437894454142
```

Summary

- An advancement of decision trees
- Can be used for both Regression and Classification
- Regression and Classification run with different criterion
- Easy to use
- Can be prone to overfitting if the hyperparameters are not managed