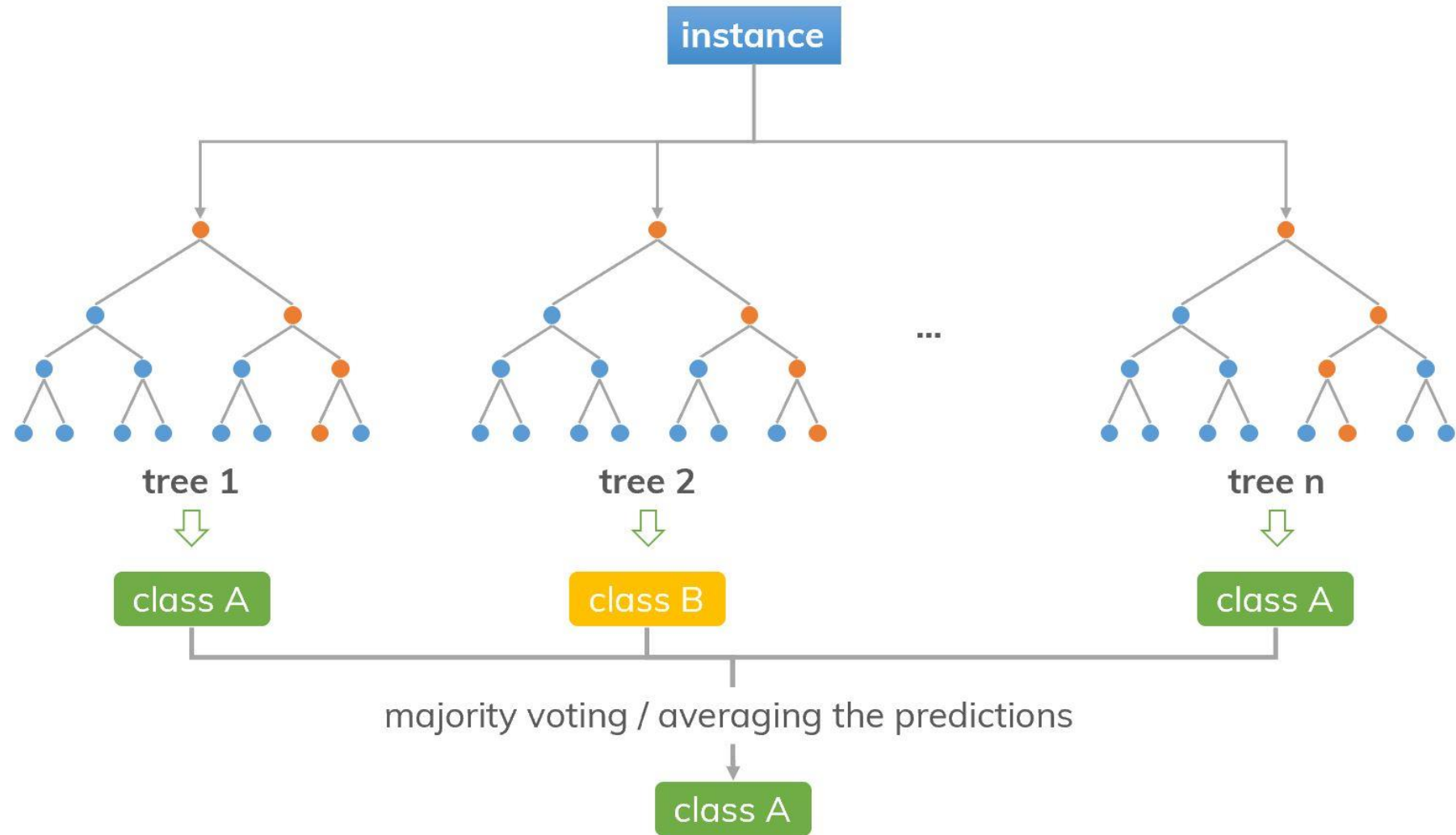


Random Forest



Random Forest Classification

Random Forest Regression

[sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.0 documentation](#)

[sklearn.ensemble.RandomForestRegressor — scikit-learn 1.3.0 documentation](#)

DIFFERENCE BETWEEN RANDOM FORESTS AND DECISION TREES

Compare random forests with decision-trees:


1. Random forests is an ensemble of multiple decision-trees.
2. Decision-trees are computationally faster as compared to random forests.
3. Deep decision-trees may suffer from overfitting. Random forest prevents overfitting by creating trees on random forests.
4. Random forest is difficult to interpret. However, a decision-tree is easily interpretable and can be converted to rules.
5. A subset of features bring diversity and avoid to pick always the same most important features.

RANDOM FOREST CLASSIFICATION

```
# Data Processing
import pandas as pd
import numpy as np

# Modelling
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score
```

```
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
```

Powered by  OpenAI

TUNING PARAMETERS

Decision Trees have several parameters that can be tuned to improve their performance. Some of the important parameters include:

- **max_depth**: the maximum depth of the tree (*default=None*)
- **min_samples_split**: the minimum number of samples required to split an internal node (*default=2*)
- **min_samples_leaf**: the minimum number of samples required to be at a leaf node (*default=1*)

```
clf = DecisionTreeClassifier(max_depth=5, min_samples_split=10, min_samples_leaf=5)
```

[Using Decision Trees and Random Forests for Machine Learning Classification in Python | by Omardonia | Level Up Coding \(gitconnected.com\)](#)

[Examine the influence of each parameter of the decision tree \(k-dm.work\)](#)

Parameters:

RandomForestClassifier

```
(bootstrap=True,  
class_weight=None,  
criterion='gini',  
max_depth=None,  
max_features='auto',  
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,  
n_estimators=100,  
n_jobs=None,  
oob_score=False,  
random_state=0,  
verbose=0,  
warm_start=False)
```

- Whether bootstrap samples are used when building trees.
- If not given, all classes are supposed to have weight one.
- The function to measure the quality of a split.
- The maximum depth of the tree.
- The number of features to consider when looking for the best split.
- Grow trees with `max_leaf_nodes` in best nodes (relative reduction in impurity).
- A node will be split if splitting impurity greater than or equal to this value.
- The minimum impurity reduction required to perform a split during the tree-building process.
- The minimum weighted fraction of the sum total of weights required to be at a leaf node.
- The minimum number of samples required to split an internal node.
- The minimum number of samples required to be at a leaf node.
- The number of trees in the forest.
- The number of jobs to run in parallel.
- Whether to use out-of-bag samples to estimate the generalization score.
- Controls both the randomness of the bootstrapping of the samples used when building trees.
- Controls the verbosity when fitting and predicting.
- *True* reuse the solution of the previous call to fit and add more estimators to the ensemble, otherwise, just fit a whole new forest.

[sklearn.ensemble.RandomForestClassifier — scikit-learn 1.3.0 documentation](#)

[Random Forest Classifier Tutorial | Kaggle](#)

VISUALIZING THE RESULTS

```
# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```


```
# Export the first three decision trees from the forest
```



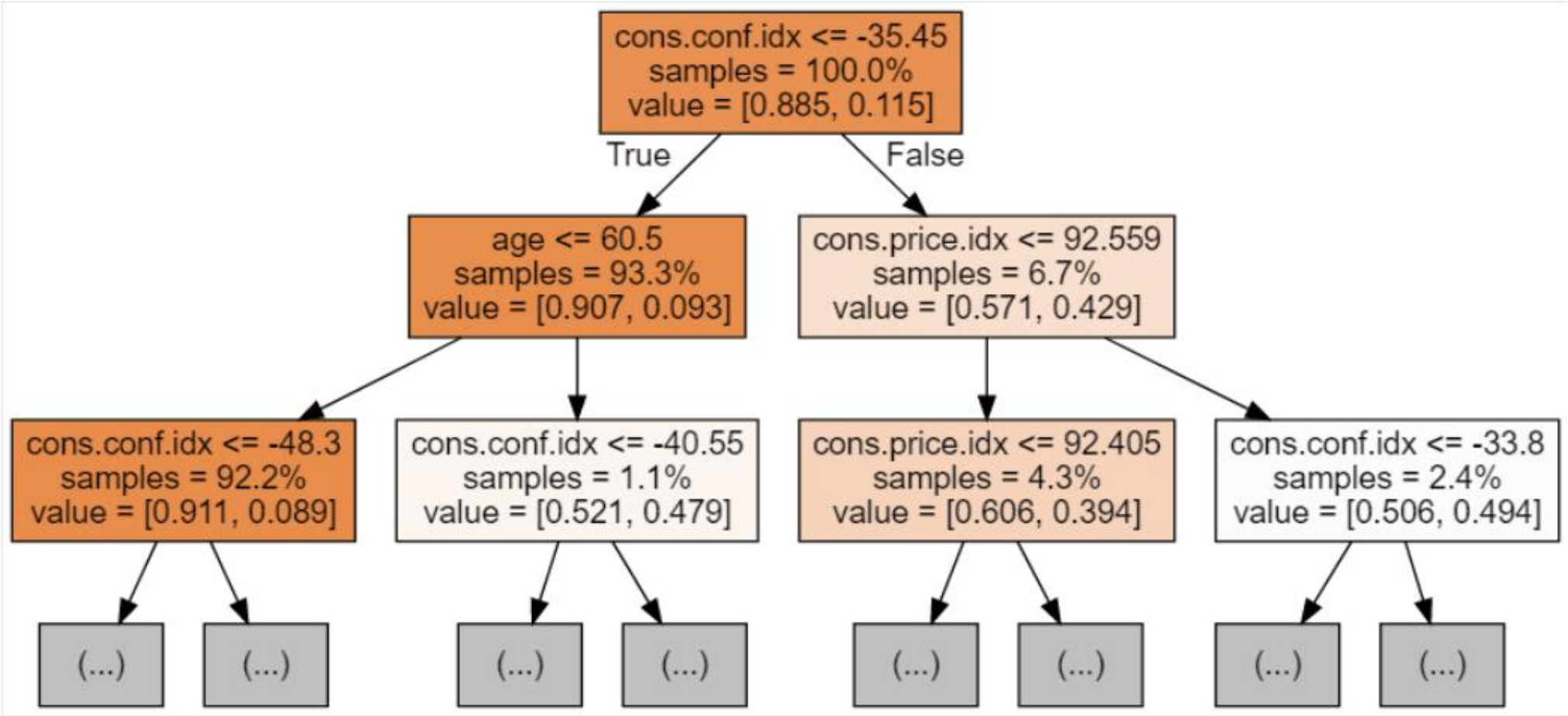
```
for i in range(3):
    tree = rf.estimators_[i]
    dot_data = export_graphviz(tree,
                               feature_names=X_train.columns,
                               filled=True,
                               max_depth=2,
                               impurity=False,
                               proportion=True)

    graph = graphviz.Source(dot_data)
    display(graph)
```

✦ Explain code

Powered by  OpenAI

VISUALIZING THE RESULTS



FEATURE IMPORTANCE

Create a series containing feature importance from the model and feature names from the training data:

```
feature_importances = pd.Series(best_rf.feature_importances_,  
index=X_train.columns).sort_values(ascending=False)
```

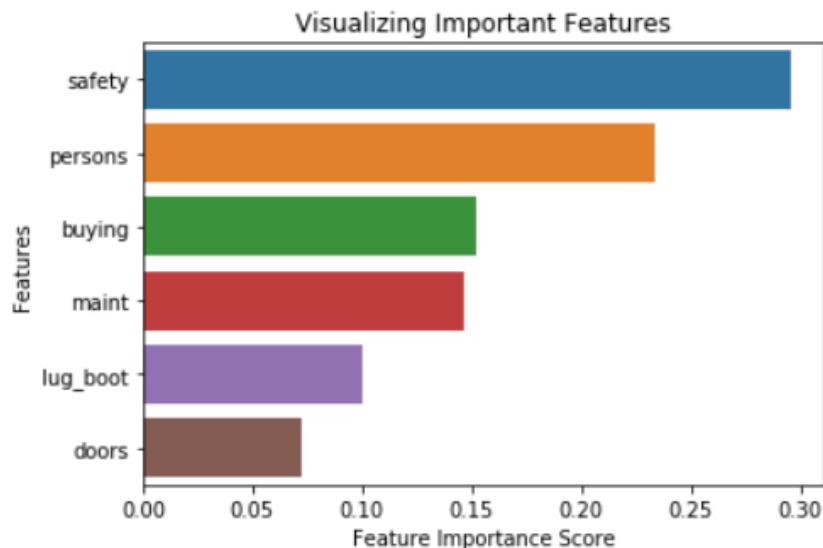
Plot a simple bar chart:

```
feature_importances.plot.bar();
```


FEATURE SCORES

```
feature_scores = pd.Series(clf.feature_importances_,  
                           index=X_train.columns).sort_values(ascending=False)
```

VISUALIZE FEATURE SCORES OF THE FEATURES



Creating a seaborn bar plot

```
sns.barplot(x=feature_scores, y=feature_scores.index)
```

Add labels to the graph

```
plt.xlabel('Feature Importance Score')
```

```
plt.ylabel('Features')
```

Add title to the graph

```
plt.title("Visualizing Important Features")
```

Visualize the graph

```
plt.show()
```

CLASSIFICATION REPORT

```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred))
```

	precision	recall	f1-score	support
acc	0.89	0.83	0.86	129
good	0.59	0.85	0.69	20
unacc	0.98	0.97	0.98	397
vgood	0.67	0.72	0.69	25
accuracy			0.93	571
macro avg	0.78	0.84	0.81	571
weighted avg	0.93	0.93	0.93	571

Support

Support is the number of actual occurrences of the class in the specified dataset. Imbalanced support in the training data may indicate structural weaknesses in the reported scores of the classifier and could indicate the need for stratified sampling or rebalancing.

[Understanding a Classification Report For Your Machine Learning Model | by Shivam Kohli | Medium](#)

- **Macro F1** calculates the *F1* separated by class but not using weights for the aggregation:

$$F1_{class1}+F1_{class2}+\cdots+F1_{classN}$$

which results in a bigger penalisation when your model does not perform well with the minority classes (if imbalance)

- **Weighted F1 score** calculates the *F1* score for each class independently but when it adds them together uses a weight that depends on the number of true labels of each class:

$$F1_{class1}*W1+F1_{class2}*W2+\cdots+F1_{classN}*W_N$$

[Random Forest Classifier Tutorial | Kaggle](#)
[classification - macro average and weighted average meaning in classification_report - Data Science Stack Exchange](#)

EVALUATION OF RANDOM FOREST ENSEMBLE FOR REGRESSION

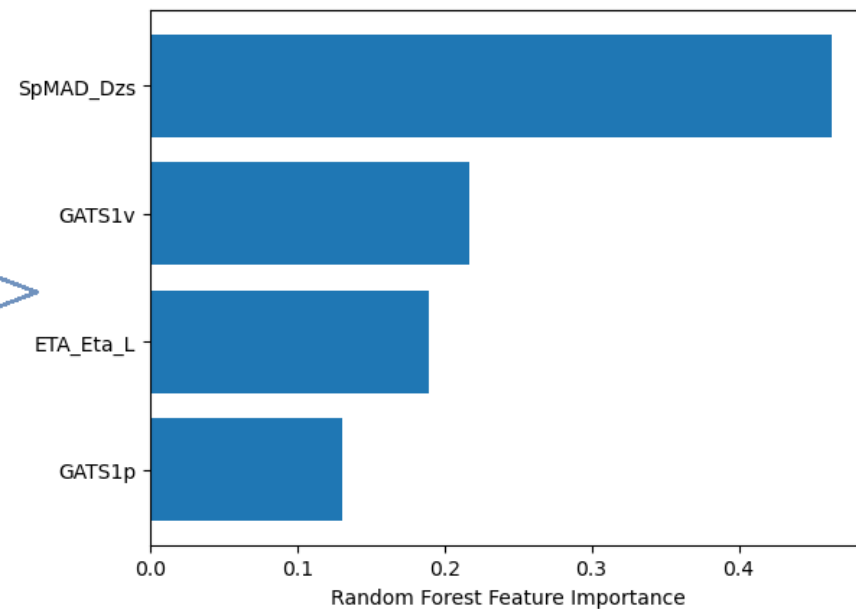
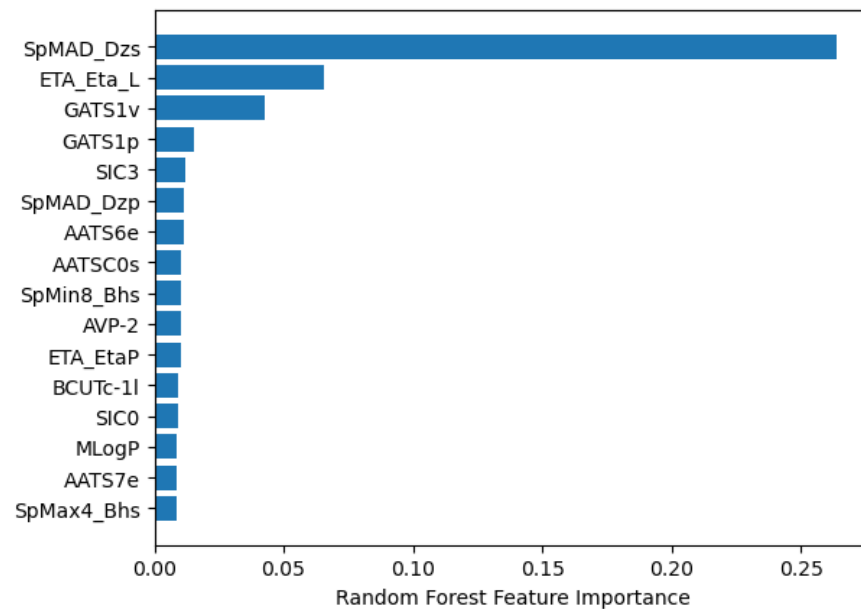
[Definitive Guide to the Random Forest Algorithm with Python and Scikit-Learn \(stackabuse.com\)](#)

[How to create a random forest for regression in Python - Thinking Neuron](#)

[Computing regression accuracy | Python Machine Learning Cookbook - Second Edition \(packtpub.com\)](#)

[Regression Metrics for Machine Learning - MachineLearningMastery.com](#)

[How to Develop a Random Forest Ensemble in Python - MachineLearningMastery.com](#)

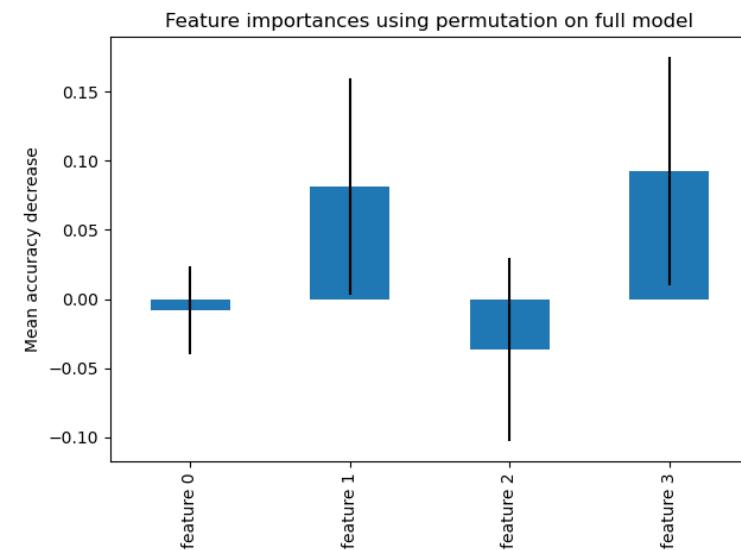


RANDOM FOREST REGRESSION

Practical example.

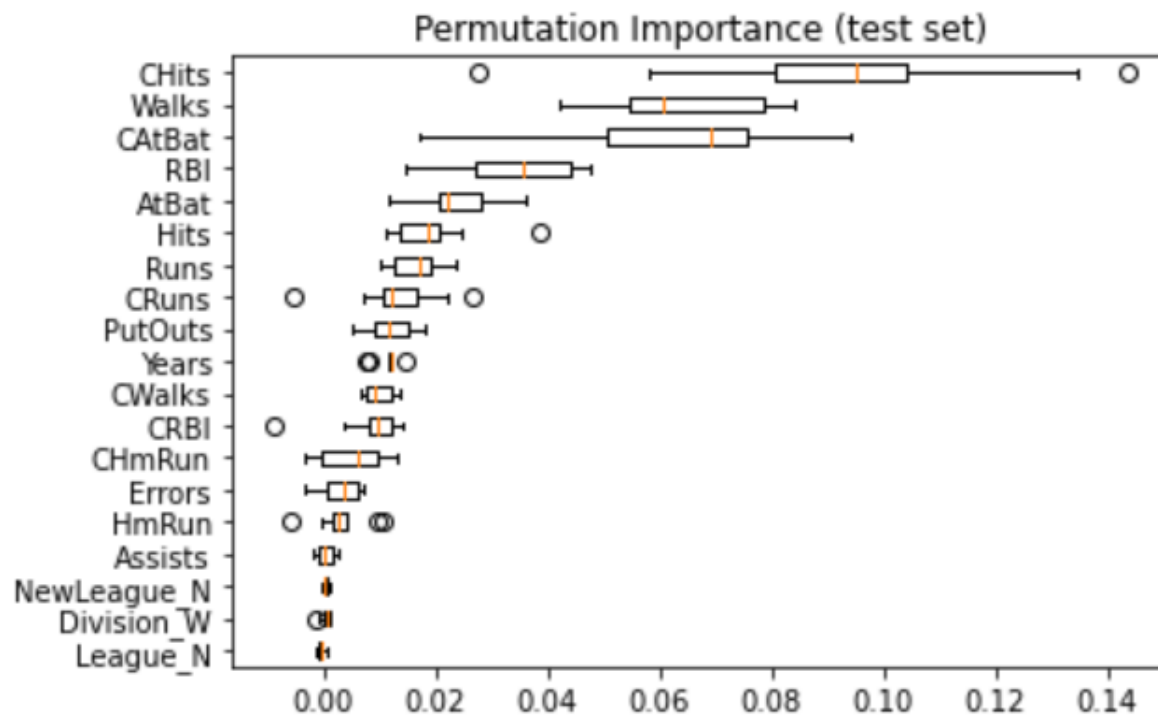
As an input, 1444 features were estimated for detection drugs concentration in samples.

535 features haven't importance after calculation of the Random Forest Regression.



PERMUTATION IMPORTANCE

```
plt.boxplot(  
    result.importances[sorted_idx].T,  
    vert=False,  
    labels=np.array(feature_names)[sorted_idx],  
)  
  
plt.title("Permutation Importance (test set)")
```



[Random forest in scikit-learn — Introduction to Regression Models \(kirenz.github.io\)](https://kirenz.github.io/)

THE ADVANTAGES OF RANDOM FOREST ALGORITHM:

1. Random forest algorithm can be used to solve both classification and regression problems.
2. It is considered as very accurate and robust model because it uses large number of decision-trees to make predictions. Random forests takes the average of all the predictions made by the decision-trees, which cancels out the biases. So, it does not suffer from the overfitting problem.
3. Random forest classifier can be used for feature selection. It means selecting the most important features out of the available features from the training dataset.

THE ADVANTAGES OF RANDOM FOREST ALGORITHM:

4. Random forest classifier can handle the missing values. There are two ways to handle the missing values. First is to use median values to replace continuous variables and second is to compute the proximity-weighted average of missing values.

Training set:

1. Replace missing values by the average value.
2. Repeat until satisfied:
 - a. Using imputed values calculated so far, train a random forest.
 - b. Compute the proximity matrix.
 - c. Using the proximity as the weight, impute missing values as the weighted average of non-missing values.

Test set:

1. If labels exist, use the imputation derived from test data.
2. If data is unlabeled, replicate the test set with a copy for each class label and proceed as with labeled data.

Age(yrs)	Ht>5	wt(lbs)	obese
20	no	130	no
24	yes	160	yes
26	no	150	no
25			no

Original data with missing values

Age(yrs)	Ht>5	wt(lbs)	obese
20	no	130	no
24	yes	160	yes
26	no	150	no
25	no	150	no

Filled missing values using strawman imputation

The weighted frequency of *no* is greater than *yes*, so *no* would be the good option to fill the missing value of observation 4 height column.

Similarly, we need to calculate the weighted frequency for the weight column to refine the filling. To calculate this, we need to multiply the observation and weights of the values for each observation as follows.

Observation 1 = $130 * 0.1$ (first red cell in matrix E) / $(0.1+0.1+0.8) = 130 * 0.1 = 13$

Observation 2 = $160 * 0.1$ (blue cell in matrix E) / $(0.1+0.1+0.8) = 160 * 0.1 = 16$

Observation 3 = $150 * 0.8$ (second red cell matrix E) / $0.1+0.1+0.8 = 150 * 0.8 = 120$

Sum of all above is 149 so the revised value to fill the weight of observation 4 is **149**.

Considering general scale, 149 is rounded to **150**.

To summarize we filled the missing values and revised them using random forests and this process is continued 6-7 times until the values do not change after recalculation.

	1	2	3	4
1				
2				
3				
4				

Yes = 1/3 No = 2/3

A	1	2	3	4
1				
2				
3				1
4			1	

B	1	2	3	4
1				
2			1	1
3		1		2
4		1	2	

The weight of *yes* = proximity of *yes* / all proximities of the sample = 0.1
(blue colored cell in matrix E) / $(0.1+0.1+0.8) = 0.1$

The weight of *no* = $0.1+0.8$ (red colored cells in matrix E) / $0.1+0.1+0.8 = 0.9$

The weighted frequency of *yes* = frequency of *yes* * the weight for *yes* = $\frac{1}{3} * 0.1 = \mathbf{0.03}$

The weighted frequency of *no* = $\frac{2}{3} * 0.9 = \mathbf{0.6}$

C	1	2	3	4
1				
2			1	1
3		1		3
4		1	3	

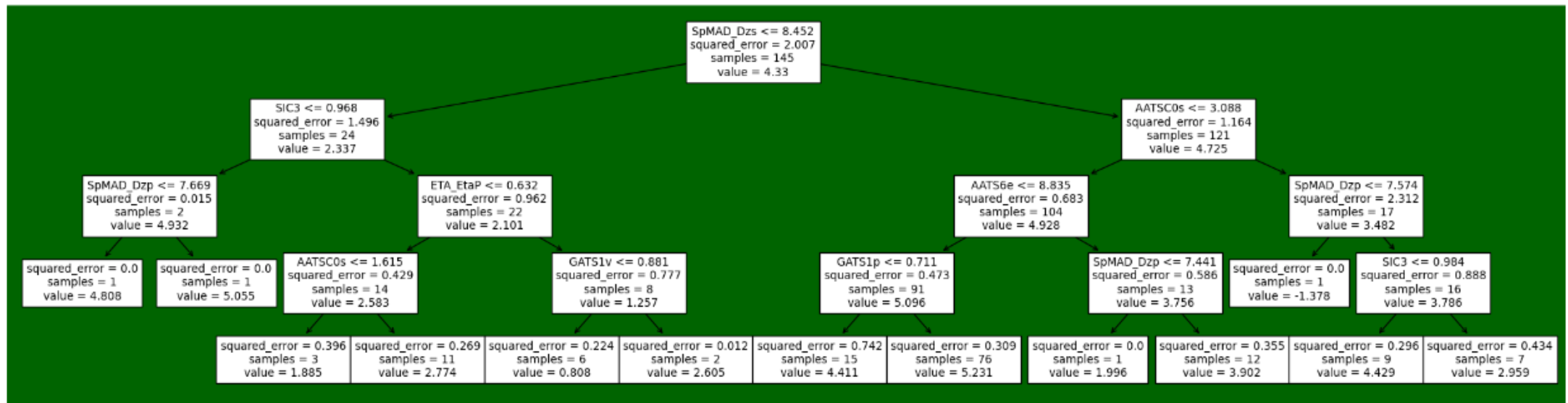
D	1	2	3	4
1		2	1	1
2	2		1	1
3	1	1		8
4	1	1	8	

E	1	2	3	4
1		0.2	0.1	0.1
2	0.2		0.1	0.1
3	0.1	0.1		0.8
4	0.1	0.1	0.8	

[Way to handle missing values-Proximity Imputation \(numpyninja.com\)](https://pypi.org/project/numpy-ninja/)

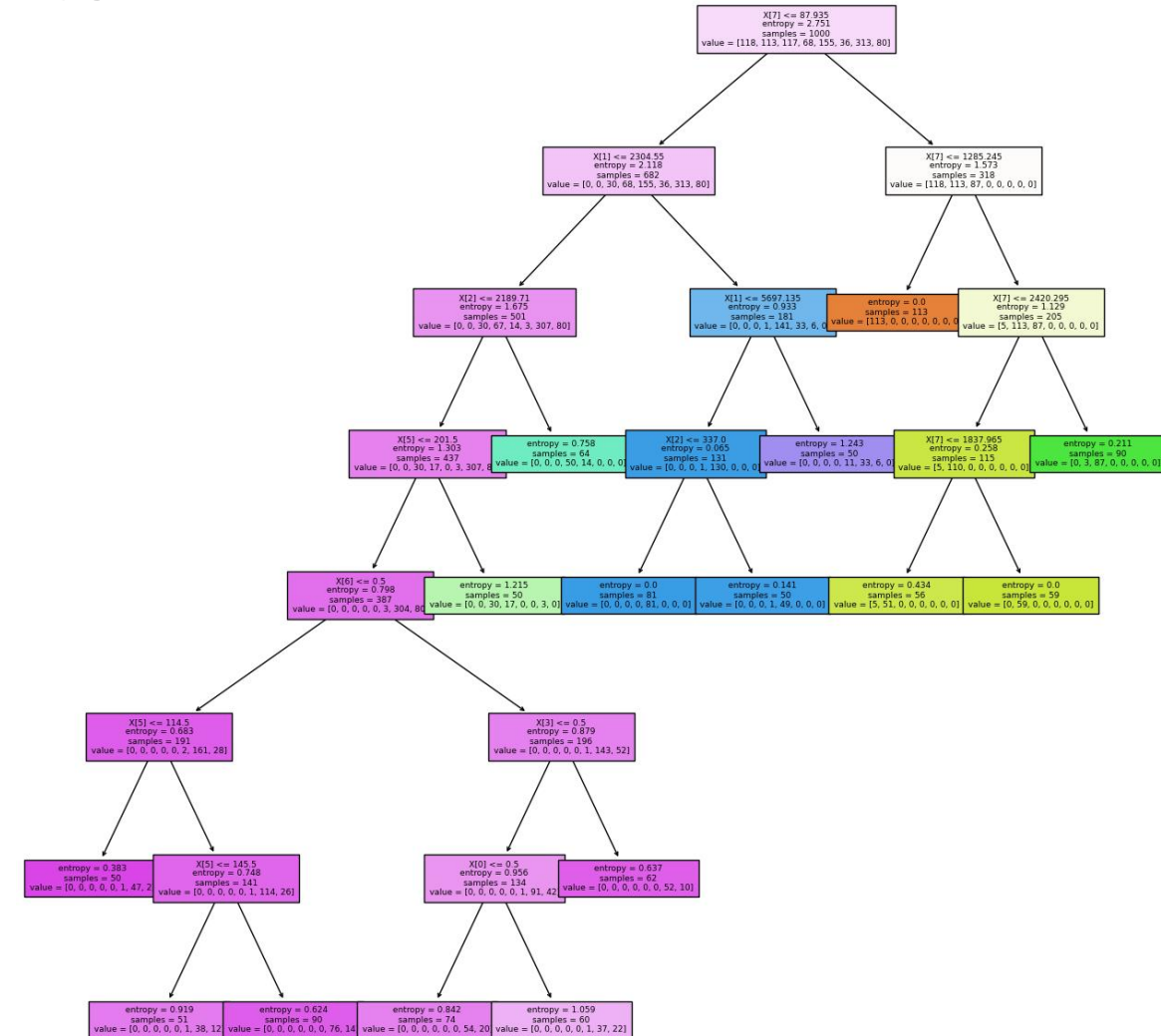
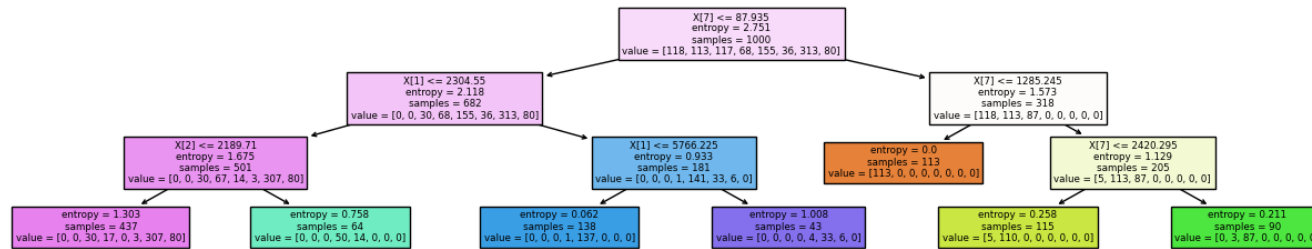
VISUALIZING THE RESULTS

```
plt.figure(figsize=(24,6),facecolor='darkgreen')
_ = tree.plot_tree(res,feature_names=X.columns)
```



Libraries for plotting results

import matplotlib.pyplot as plt
from sklearn.tree import plot_tree

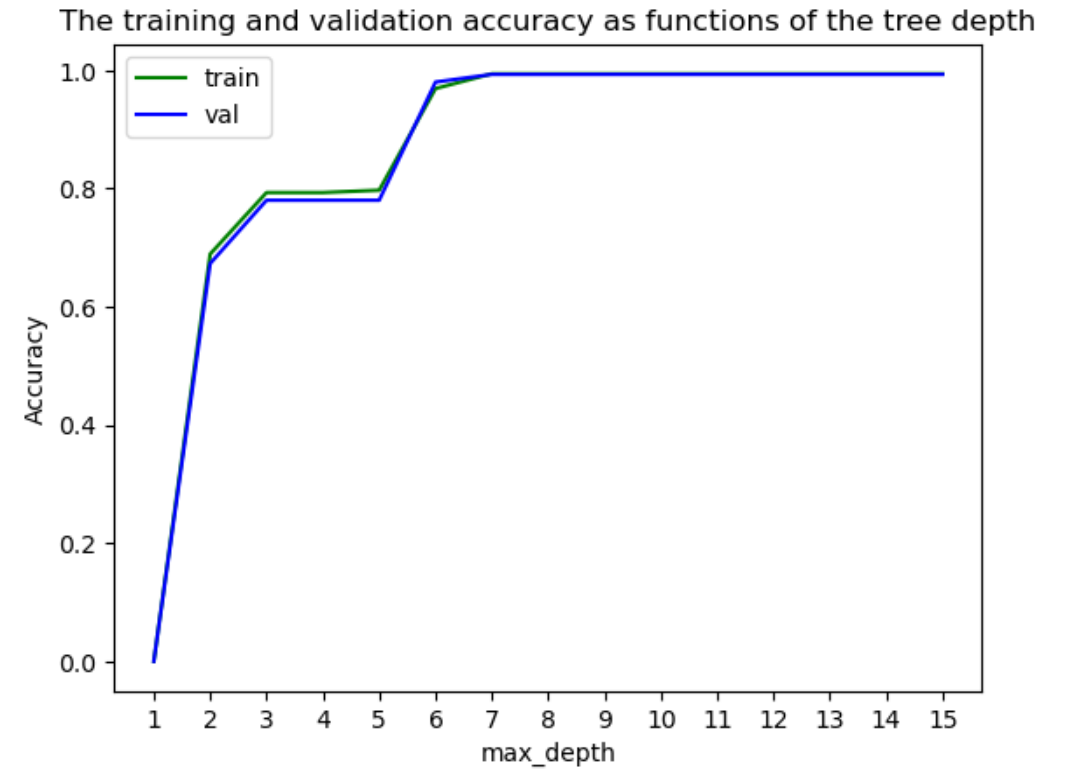
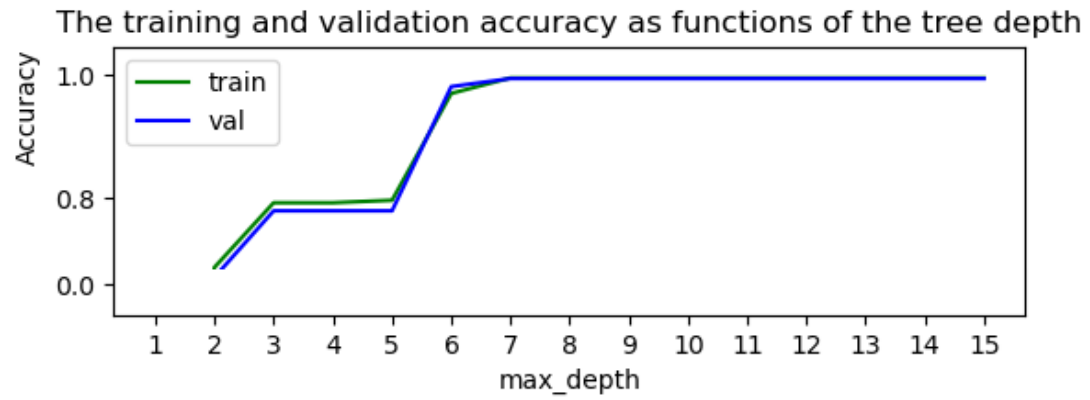


```
# DecisionTreeClassifier with entropy with parameter max_depth=2:  
model = DecisionTreeClassifier(criterion='entropy',  
                               min_samples_split=124)  
model.fit(X, y)
```

DecisionTreeClassifier(criterion='entropy', min_samples_split=124)

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
#Depicting the results  
plt.figure(figsize=(16, 16))  
plot_tree(model, fontsize=6.5, filled=True)  
plt.show()
```



Accuracy representation and counted results (correction):

```
import numpy as np
np.round(accuracy, decimals=3)
```