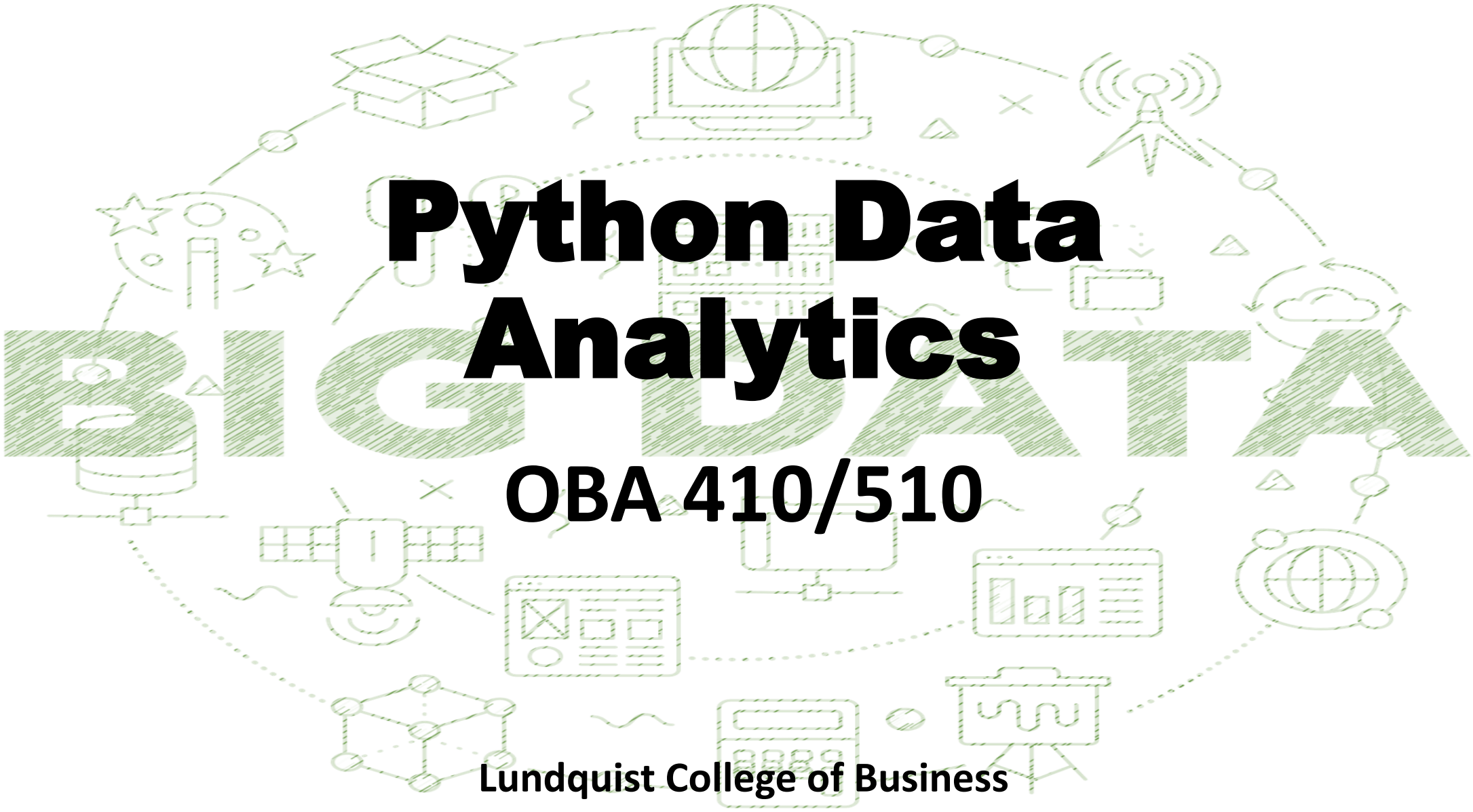


Python Data Analytics

OBA 410/510

Lundquist College of Business

Background image from www. givweb.com

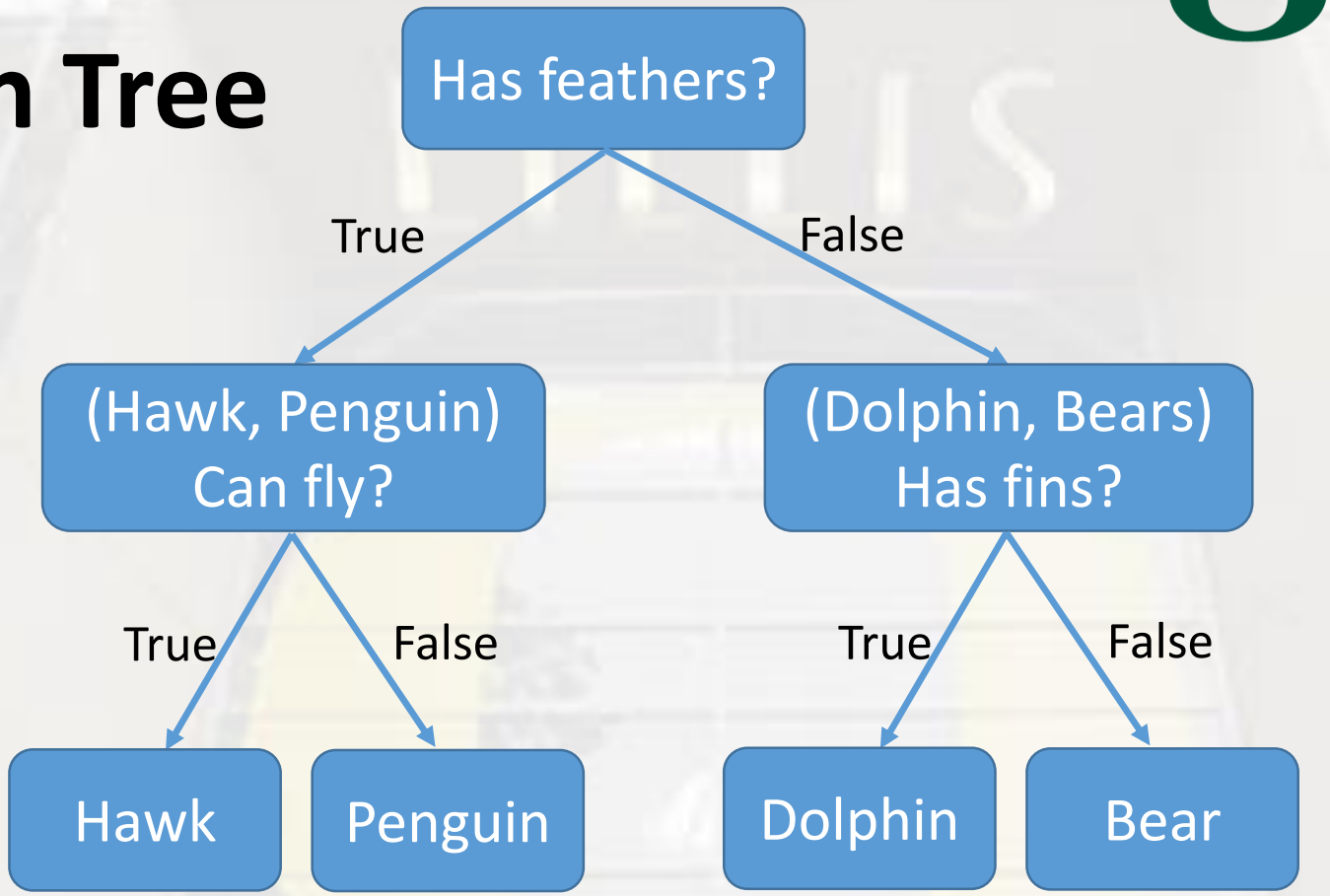
Some of the slides and contents are adopted from Statistical Learning course by Dr. Trevor Hastie

Decision Tree

Decision Tree

- Applied to both Regression and Classification
- Essentially, decision trees learn a hierarchy of if/else questions, leading to a decision
- These questions are similar to the questions you might ask in a game of 20 Questions
- Imagine you want to distinguish between these four animals:
  - Bears, hawks, penguins, and dolphins
- Our goal is to get to the right answer by asking as few if/else questions as possible

Decision Tree

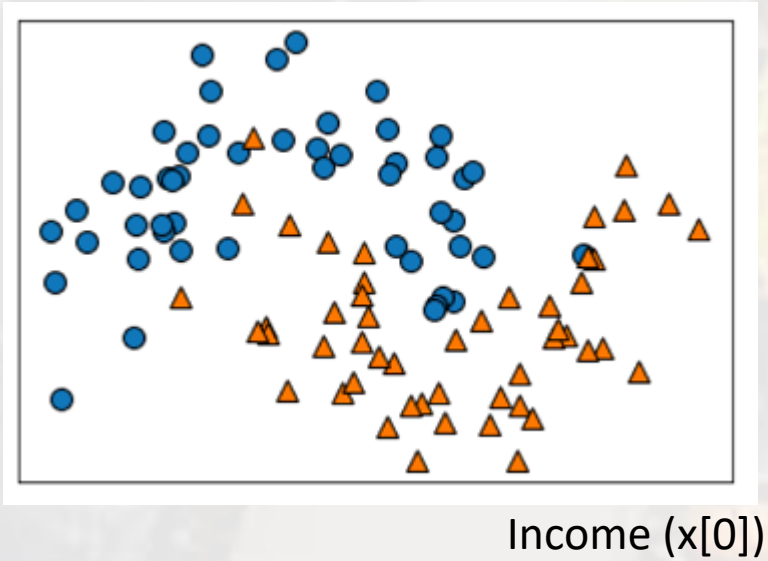


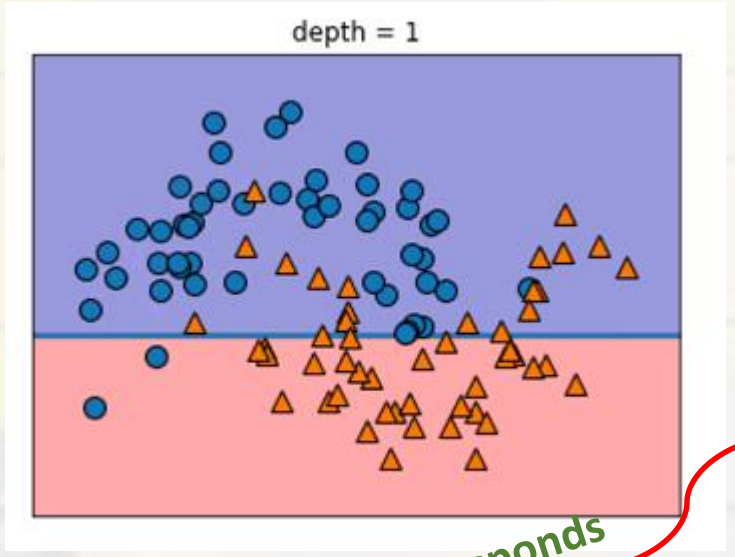
Decision Tree

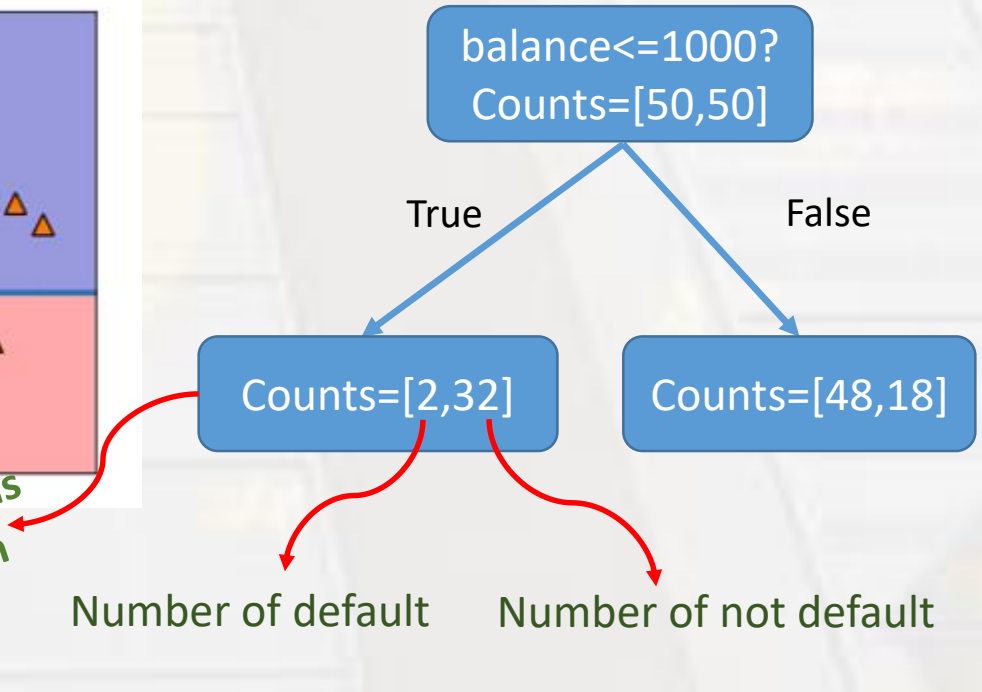
- In decision tree, we have nodes and branches
- Each node either represent a question or a terminal node (which is called ***leaf)*** that has the answer
- Learning a decision tree means learning the sequence of if/else questions that gets us the true answer most quickly
- In machine leaning these questions are called split
- If our features are not in the binary form (similar to the animal example), questions will be in the form of "*is feature smaller/larger than value a?*"

Decision Tree

- To build a decision tree, the algorithm searches over all possible splits and finds the one that is most informative about the target variable
  - First split is whether balance<=1000 (it is shown by the blue line)
  - This gives the best information about the target variable (bests separates defaults from not defaults)

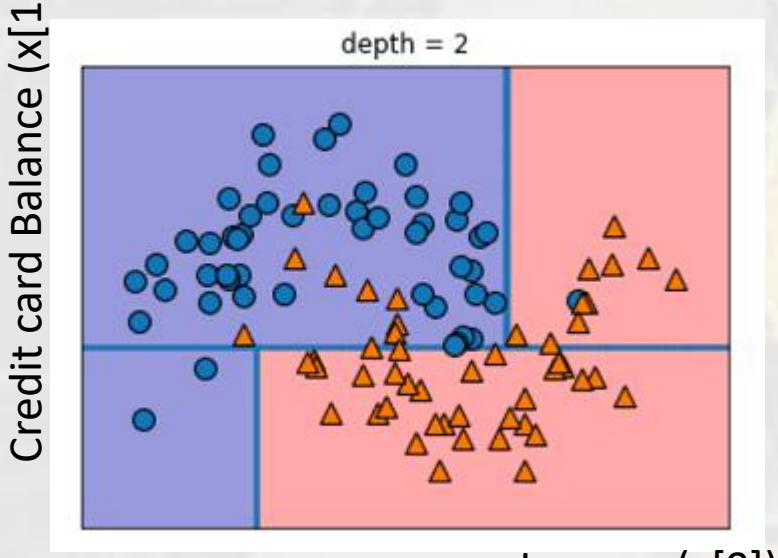


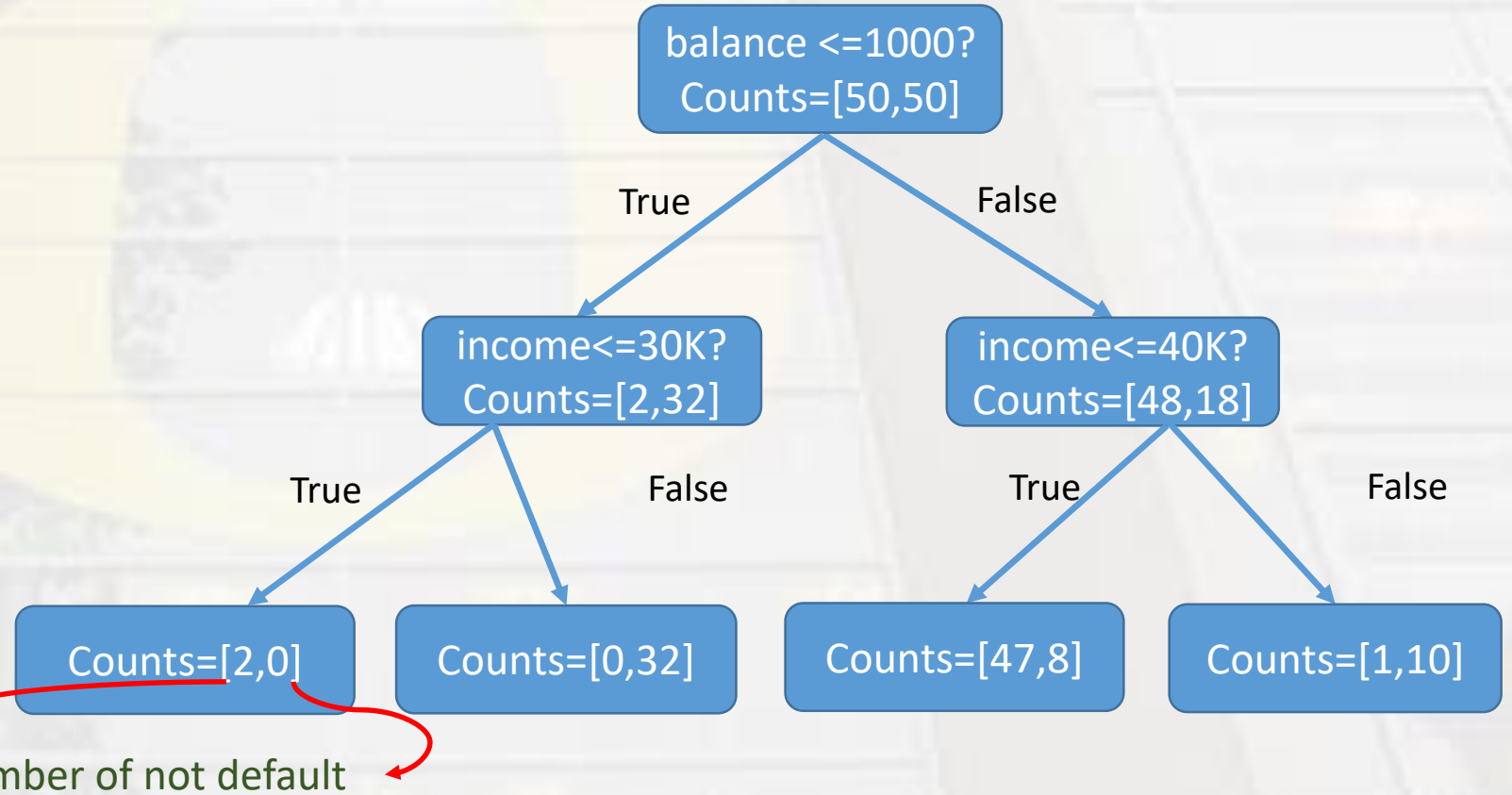




Decision Tree

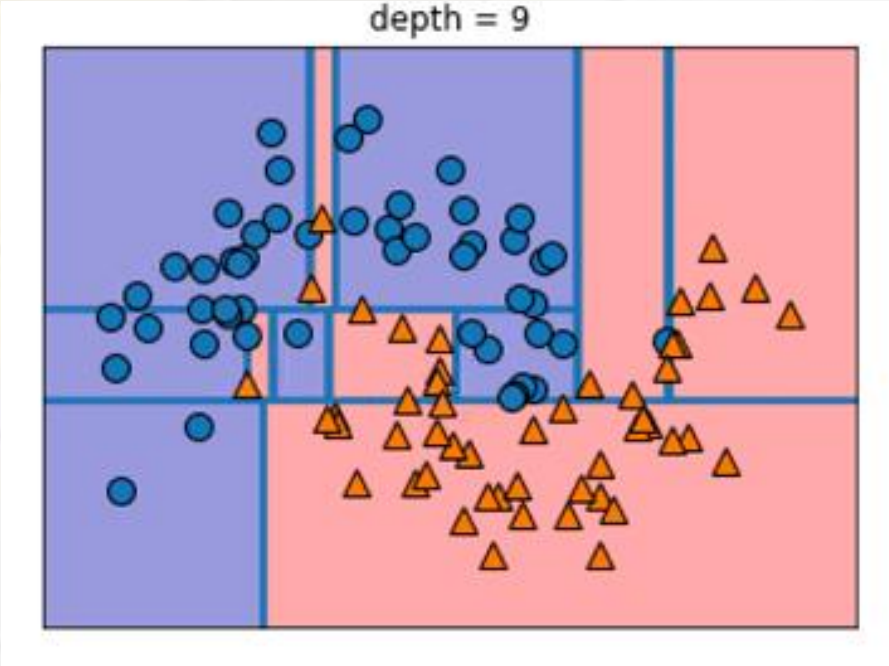
- Even though the first split did a good job separating two classes, the bottom region still contain points belonging to “default” and the top region contains points belonging to “not default”
- We can build a more accurate model by repeating the process of finding the best split in both top and bottom regions





Decision Tree

- Every split concerns only a single feature, therefore, the regions in the resulting partitions always have axis-parallel boundaries
- We can continue this recursive process
- This can continue until each region only contains one class of data
- A leaf containing only one class is called ***pure***
- Prediction on a new data point is made by checking which region of the partition of the feature space the point lies in



Decision Tree in Python

- Diabetes_2_features dataset

```
diabete2=pd.read_csv('diabetes_2_features.csv')
diabete2.head()
```

	Glucose	BloodPressure	Outcome
0	131	109	diabetic
1	114	52	non-diabetic
2	108	50	non-diabetic
3	84	75	non-diabetic
4	69	62	non-diabetic

Decision Tree in Python

- Diabetes_2_features dataset
- random_state in the tree is used for tie-breaking internally

```
X,y=diabete2.iloc[:,:-1],diabete2['Outcome']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=0)
dt.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt.score(X_test,y_test)))
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 1.000
```

Visualizing Decision Trees

- A .dot file will be generated in the same folder that you have your notebook

```
from sklearn.tree import export_graphviz
export_graphviz(dt,out_file='tree_vis.dot',class_names=dt.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

Your decision tree model's name

Choose a name for your .dot file

```
dt.classes_
```

```
array(['diabetic', 'non-diabetic'],
```

```
X.columns
```

```
Index(['Glucose', 'BloodPressure'],
```

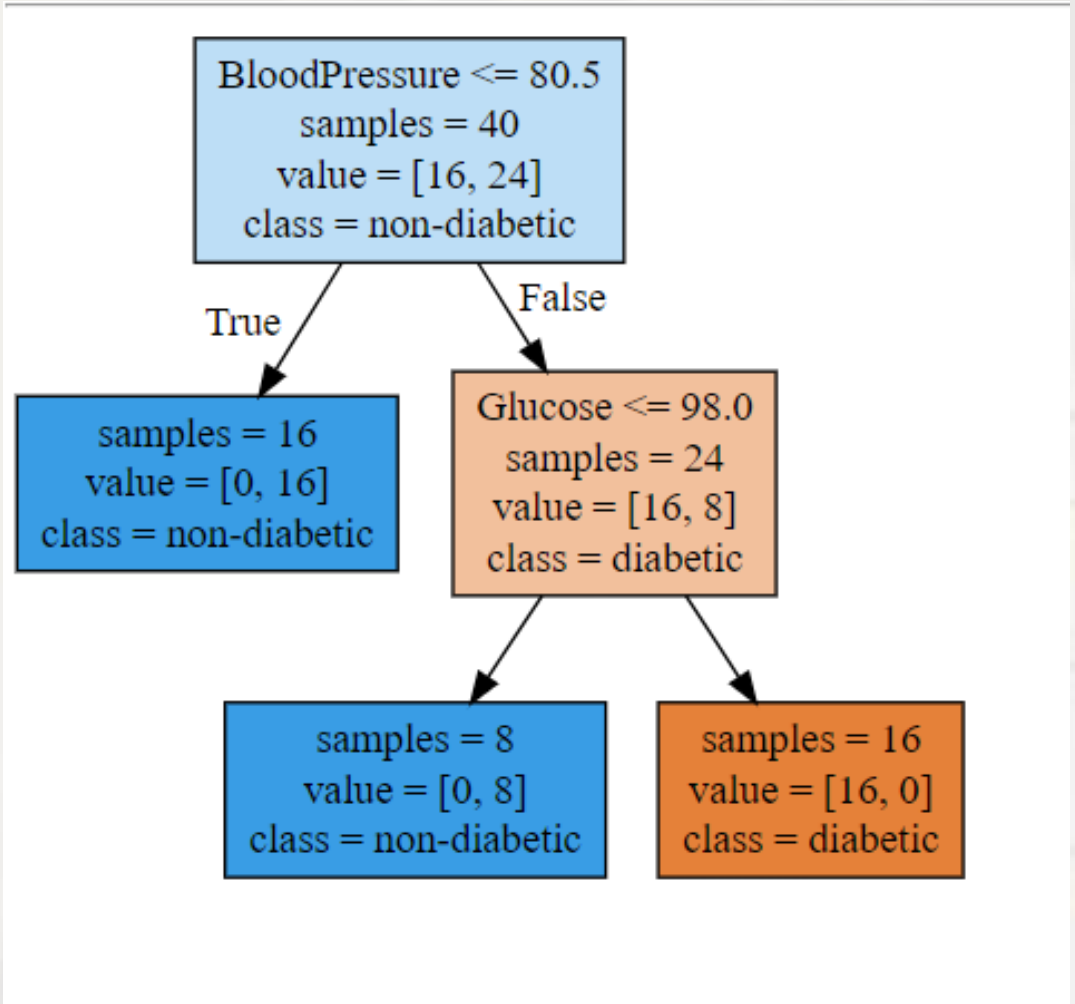
Name of classes in the target variable

Visualizing Decision Trees

- If you do not have graphviz installed on your computer
  - Go to the following website
  - http://webgraphviz.com/
  - Open your .dot file
  - Copy the code in your .dot file and paste it in the text area
  - Click on “Generate graph”

Visualized Decision Tree

- .



Decision Tree in Python

- Diabetes dataset

Only because of visualization of the tree, otherwise there is no need to change the type

```
diabete=pd.read_csv('diabetes.csv')
diabete['Outcome']=diabete['Outcome'].astype('str')
diabete.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Decision Tree in Python

- Diabetes dataset

```
X,y=diabete.iloc[:,:-1],diabete['Outcome']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
from sklearn.tree import DecisionTreeClassifier
dt2=DecisionTreeClassifier(random_state=0)
dt2.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt2.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt2.score(X_test,y_test)))
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.719
```

Decision Tree in Python

- Diabetes dataset

```
from sklearn.tree import export_graphviz
export_graphviz(dt2,out_file='tree_vis2.dot',class_names=dt2.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

```
X.columns
```

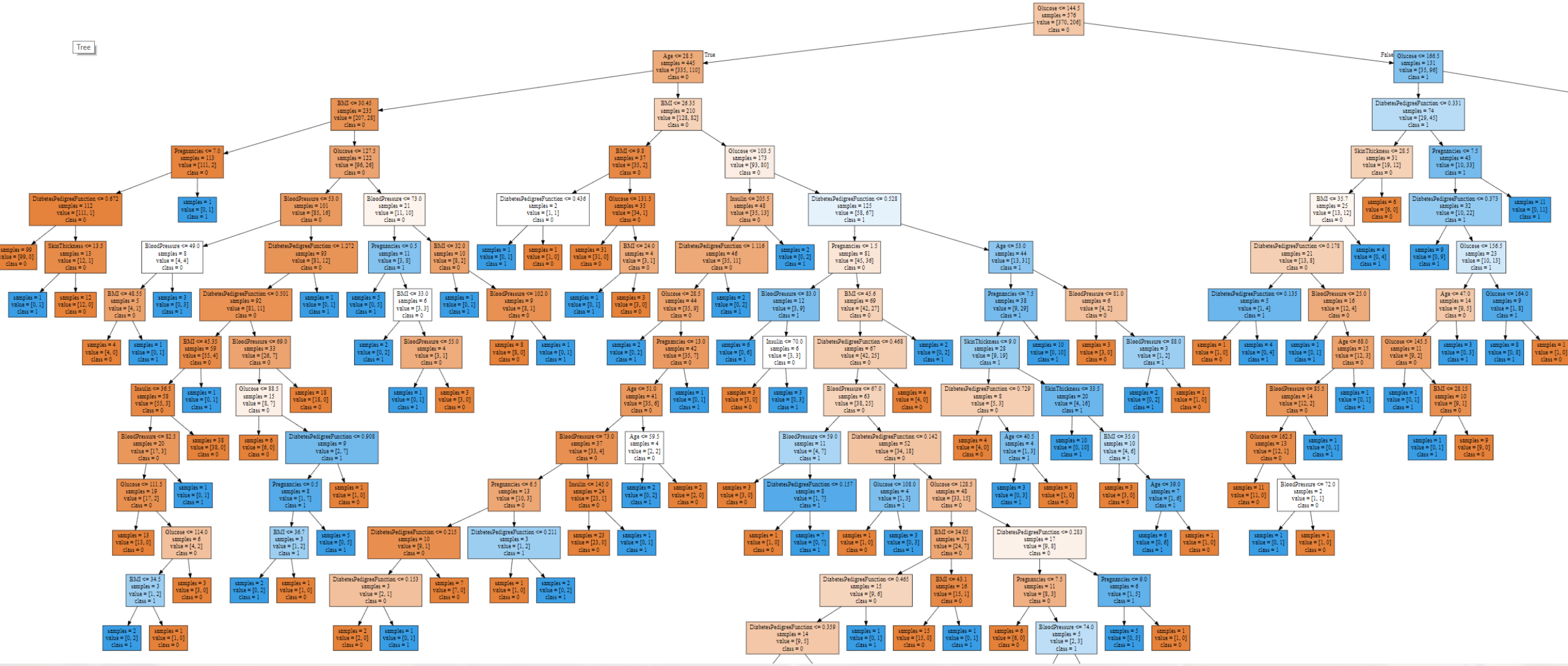
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

```
dt2.classes_
```

```
array(['0', '1'], dtype=object)
```

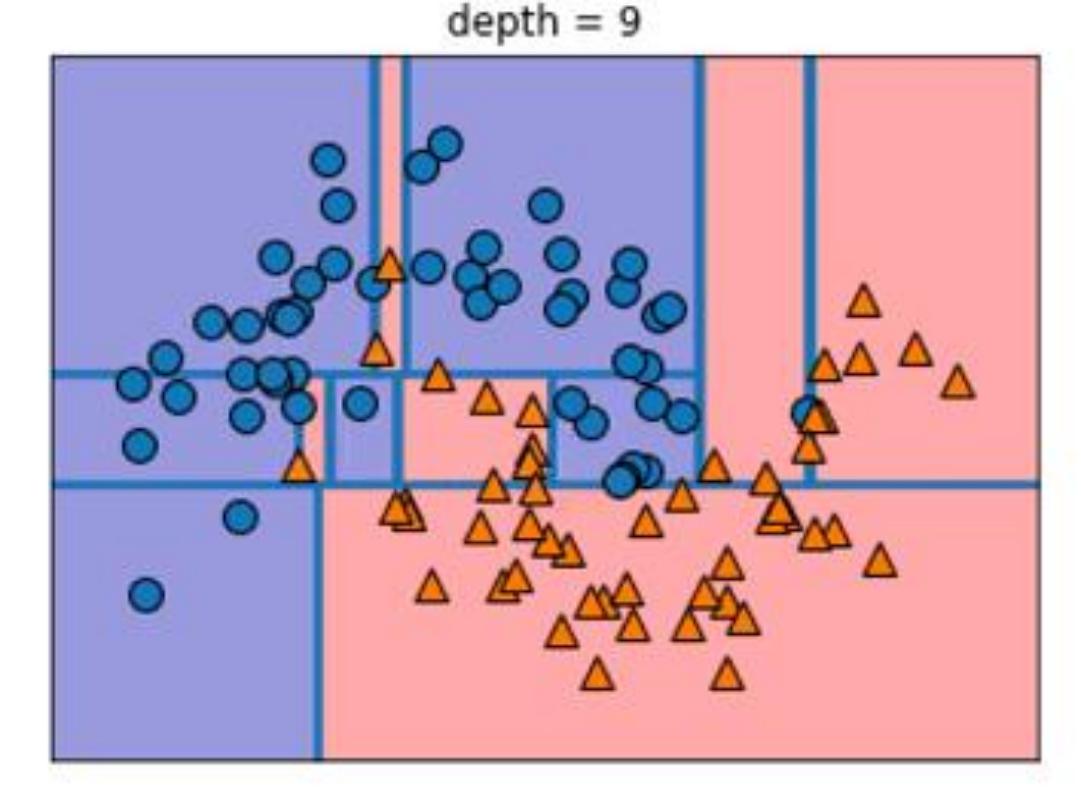
UNIVERSITY OF OREGON
Lundquist College of Business

Visualized Decision Tree



Controlling the complexity of decision trees

- Typically, building a tree as described and continuing until all leaves are pure leads to very complex models
- Therefore, overfitting will be very likely
  - Very high accuracy on training but not very generalizable to unseen data



Controlling the complexity of decision trees

- There are two common strategies to prevent overfitting:
  - Stopping the creation of tree early, ***pre-pruning***
  - Building the tree completely, then removing nodes that contain little information, ***post-pruning***
- scikit-learn implements pre-pruning
- Three possible criteria for pre-pruning:
  - Limiting the maximum depth of tree (***max_depth*** parameter)
  - Limiting maximum number of leaves(***max_leaf_nodes*** parameter)
  - Requiring a minimum number of points in a node to keep splitting it (***min_samples_split*** parameter)

Decision Tree in Python

- Diabetes dataset

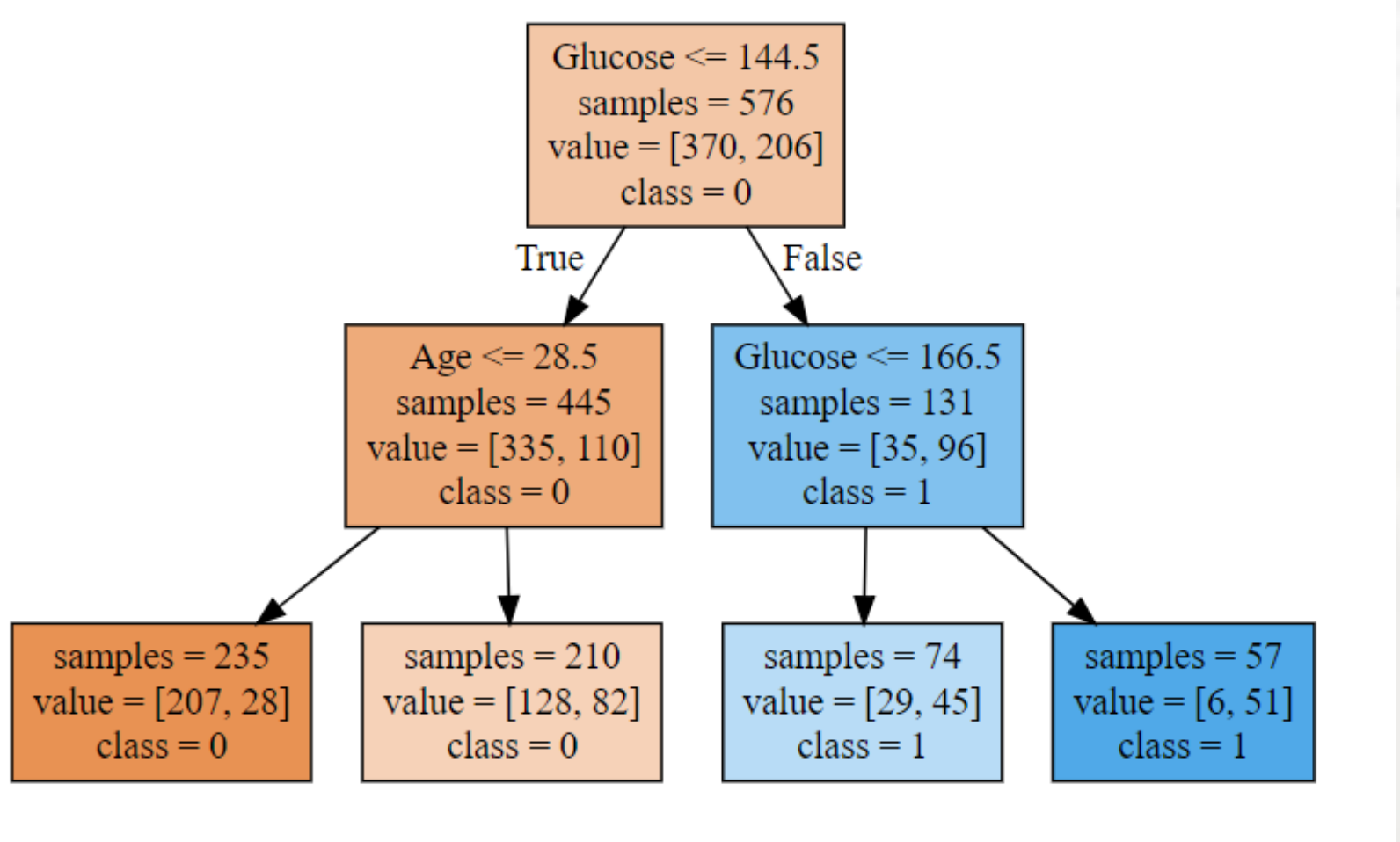
```
dt2=DecisionTreeClassifier(max_depth=2,random_state=0)
dt2.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt2.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt2.score(X_test,y_test)))
```

```
Decision tree acc on train: 0.748
Decision tree acc on test: 0.750
```

```
export_graphviz(dt2,out_file='tree_vis2.dot',class_names=dt2.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

Visualized Decision Tree

- .



Decision Tree in Python

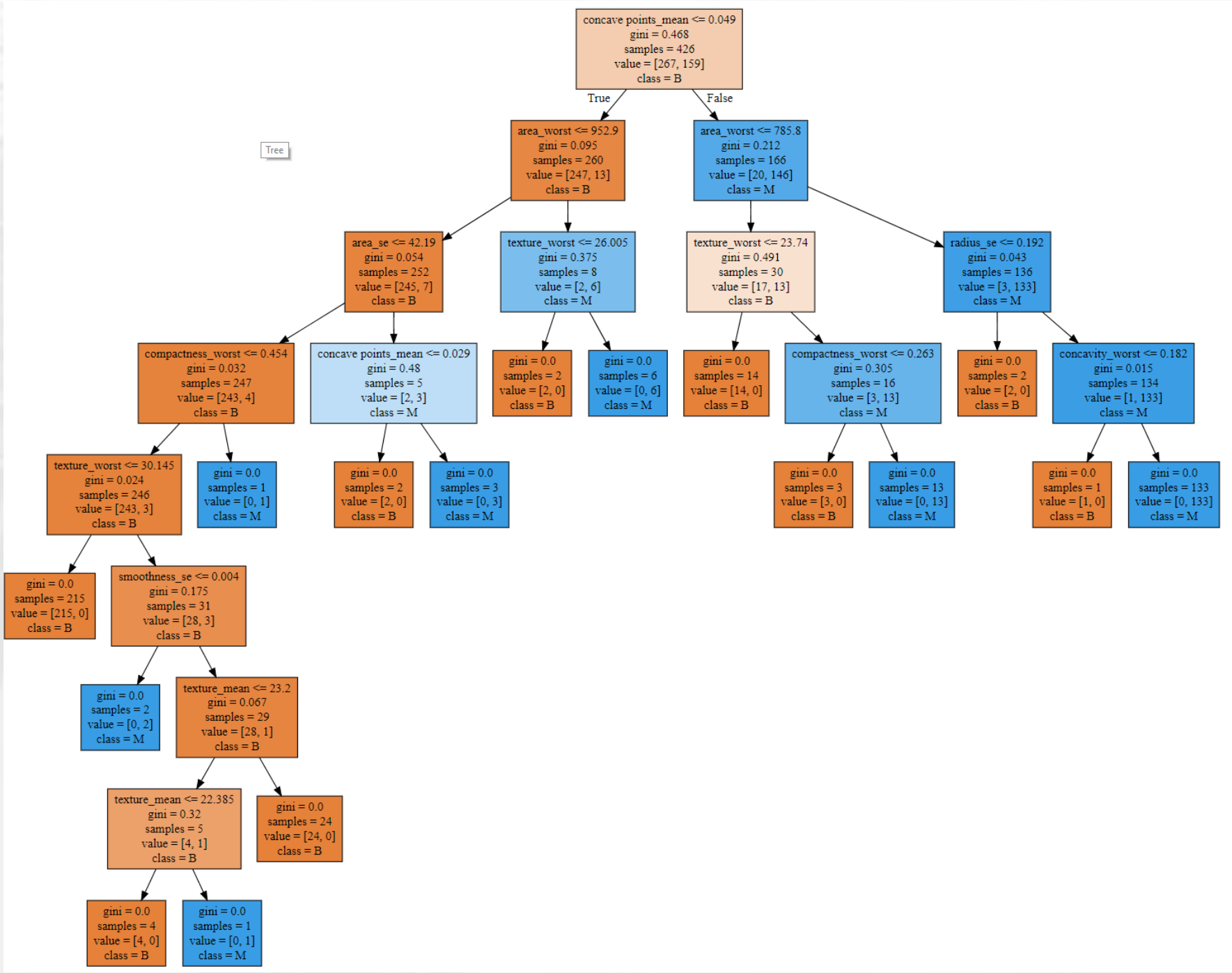
- Breast cancer dataset

```
cancer=pd.read_csv('breast_cancer_data.csv',index_col=0)
cancer.head()
```

```
X,y=cancer.iloc[:,1:],cancer['diagnosis']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt3=DecisionTreeClassifier(random_state=0)
dt3.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt3.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt3.score(X_test,y_test)))
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.881
```

```
export_graphviz(dt3,out_file='tree_vis3.dot',class_names=dt3.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

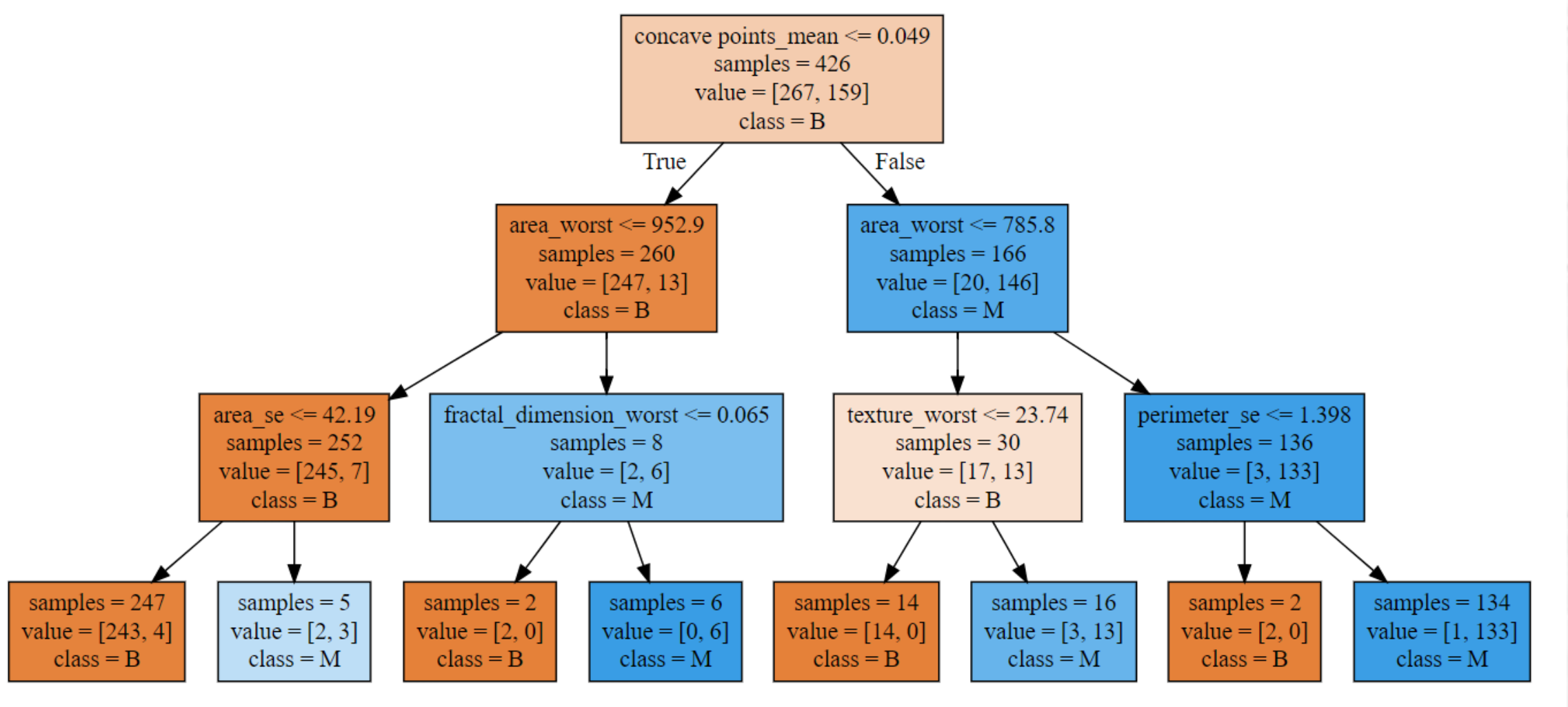

Tree
concave points_mean <= 0.049
gini = 0.468
samples = 426
value = [267, 159]
class = B
True
False
area_worst <= 952.9
gini = 0.095
samples = 260
value = [247, 13]
class = B
area_worst <= 785.8
gini = 0.212
samples = 166
value = [20, 146]
class = M
area_se <= 42.19
gini = 0.054
samples = 252
value = [245, 7]
class = B
texture_worst <= 26.005
gini = 0.375
samples = 8
value = [2, 6]
class = M
texture_worst <= 23.74
gini = 0.491
samples = 30
value = [17, 13]
class = B
radius_se <= 0.192
gini = 0.043
samples = 136
value = [3, 133]
class = M
compactness_worst <= 0.454
gini = 0.032
samples = 247
value = [243, 4]
class = B
concave points_mean <= 0.029
gini = 0.48
samples = 5
value = [2, 3]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 6
value = [0, 6]
class = M
gini = 0.0
samples = 14
value = [14, 0]
class = B
compactness_worst <= 0.263
gini = 0.305
samples = 16
value = [3, 13]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
concavity_worst <= 0.182
gini = 0.015
samples = 134
value = [1, 133]
class = M
texture_worst <= 30.145
gini = 0.024
samples = 246
value = [243, 3]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 3
value = [0, 3]
class = M
gini = 0.0
samples = 3
value = [3, 0]
class = B
gini = 0.0
samples = 13
value = [0, 13]
class = M
gini = 0.0
samples = 1
value = [1, 0]
class = B
gini = 0.0
samples = 133
value = [0, 133]
class = M
gini = 0.0
samples = 215
value = [215, 0]
class = B
smoothness_se <= 0.004
gini = 0.175
samples = 31
value = [28, 3]
class = B
gini = 0.0
samples = 2
value = [0, 2]
class = M
texture_mean <= 23.2
gini = 0.067
samples = 29
value = [28, 1]
class = B
texture_mean <= 22.385
gini = 0.32
samples = 5
value = [4, 1]
class = B
gini = 0.0
samples = 24
value = [24, 0]
class = B
gini = 0.0
samples = 4
value = [4, 0]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M

Decision Tree in Python

- Breast cancer dataset

```
dt3=DecisionTreeClassifier(max_depth=3,random_state=0)
dt3.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt3.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt3.score(X_test,y_test)))
export_graphviz(dt3,out_file='tree_vis3.dot',class_names=dt3.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

```
Decision tree acc on train: 0.977
Decision tree acc on test: 0.937
```


concave points_mean <= 0.049
samples = 426
value = [267, 159]
class = B
True
False
area_worst <= 952.9
samples = 260
value = [247, 13]
class = B
area_worst <= 785.8
samples = 166
value = [20, 146]
class = M
area_se <= 42.19
samples = 252
value = [245, 7]
class = B
fractal_dimension_worst <= 0.065
samples = 8
value = [2, 6]
class = M
texture_worst <= 23.74
samples = 30
value = [17, 13]
class = B
perimeter_se <= 1.398
samples = 136
value = [3, 133]
class = M
samples = 247
value = [243, 4]
class = B
samples = 5
value = [2, 3]
class = M
samples = 2
value = [2, 0]
class = B
samples = 6
value = [0, 6]
class = M
samples = 14
value = [14, 0]
class = B
samples = 16
value = [3, 13]
class = M
samples = 2
value = [2, 0]
class = B
samples = 134
value = [1, 133]
class = M

Gini Index

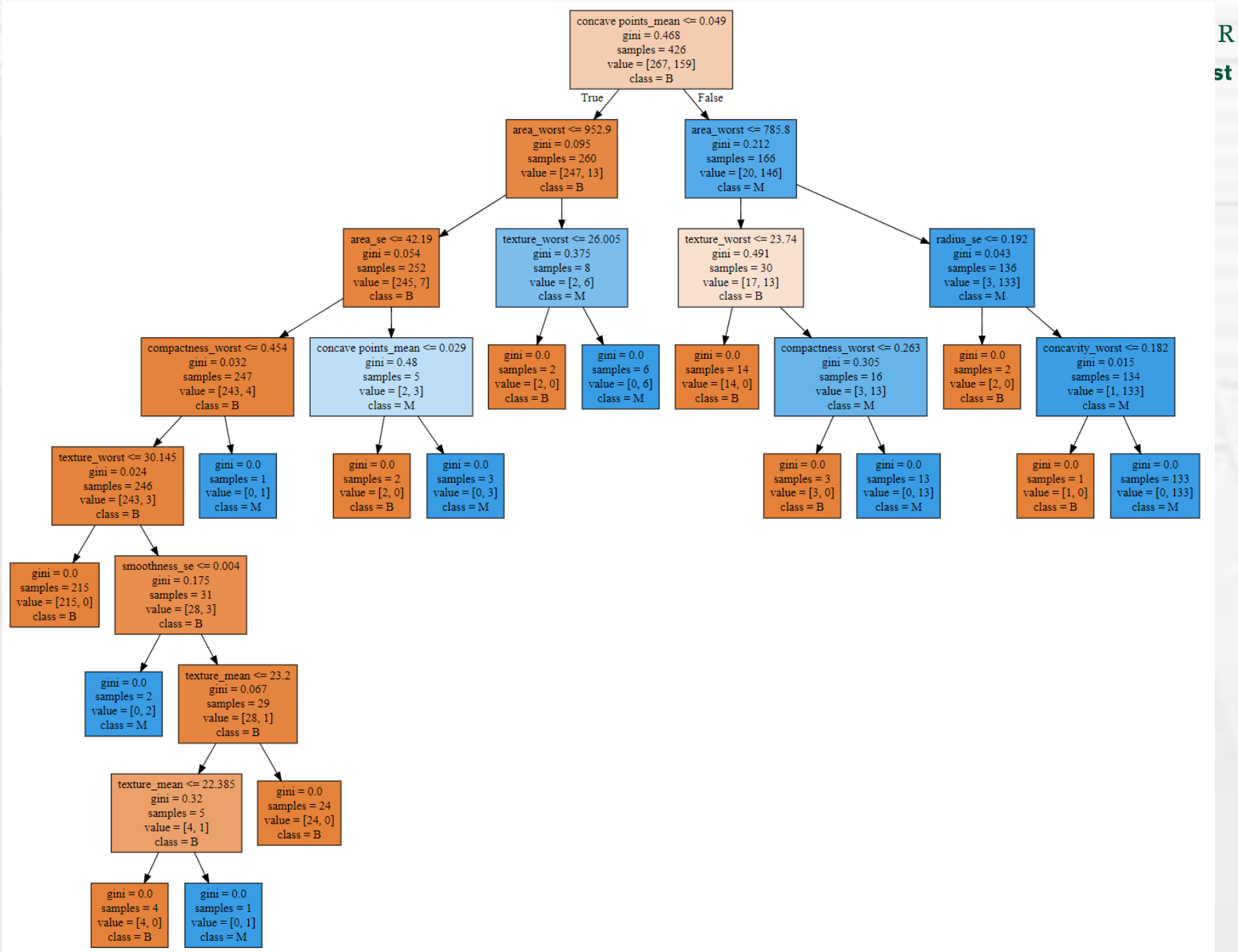
- Gini Index: a measure of node impurity in classification decision tree
- A small value indicates that a node contains predominantly observations from single class
- Therefore, lower values for Gini, less impurity (more purity) in a node
- If in ***export_graphviz*** you set the ***impurity*** parameter to ***True***, you will see the Gini index for all nodes in the decision tree

Gini Index

- Breast cancer dataset

```
X,y=cancer.iloc[:,1:],cancer['diagnosis']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt3=DecisionTreeClassifier(random_state=0)
dt3.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt3.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt3.score(X_test,y_test)))
export_graphviz(dt3,out_file='tree_vis3.dot',class_names=dt3.classes_,
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.881
```


concave points_mean <= 0.049
gini = 0.468
samples = 426
value = [267, 159]
class = B
True
False
area_worst <= 952.9
gini = 0.095
samples = 260
value = [247, 13]
class = B
area_worst <= 785.8
gini = 0.212
samples = 166
value = [20, 146]
class = M
area_se <= 42.19
gini = 0.054
samples = 252
value = [245, 7]
class = B
texture_worst <= 26.005
gini = 0.375
samples = 8
value = [2, 6]
class = M
texture_worst <= 23.74
gini = 0.491
samples = 30
value = [17, 13]
class = B
radius_se <= 0.192
gini = 0.043
samples = 136
value = [3, 133]
class = M
compactness_worst <= 0.454
gini = 0.032
samples = 247
value = [243, 4]
class = B
concave points_mean <= 0.029
gini = 0.48
samples = 5
value = [2, 3]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 6
value = [0, 6]
class = M
gini = 0.0
samples = 14
value = [14, 0]
class = B
compactness_worst <= 0.263
gini = 0.305
samples = 16
value = [3, 13]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
concavity_worst <= 0.182
gini = 0.015
samples = 134
value = [1, 133]
class = M
texture_worst <= 30.145
gini = 0.024
samples = 246
value = [243, 3]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M
gini = 0.0
samples = 2
value = [2, 0]
class = B
gini = 0.0
samples = 3
value = [0, 3]
class = M
gini = 0.0
samples = 3
value = [3, 0]
class = B
gini = 0.0
samples = 13
value = [0, 13]
class = M
gini = 0.0
samples = 1
value = [1, 0]
class = B
gini = 0.0
samples = 133
value = [0, 133]
class = M
gini = 0.0
samples = 215
value = [215, 0]
class = B
smoothness_se <= 0.004
gini = 0.175
samples = 31
value = [28, 3]
class = B
gini = 0.0
samples = 2
value = [0, 2]
class = M
texture_mean <= 23.2
gini = 0.067
samples = 29
value = [28, 1]
class = B
texture_mean <= 22.385
gini = 0.32
samples = 5
value = [4, 1]
class = B
gini = 0.0
samples = 24
value = [24, 0]
class = B
gini = 0.0
samples = 4
value = [4, 0]
class = B
gini = 0.0
samples = 1
value = [0, 1]
class = M

Gini Index

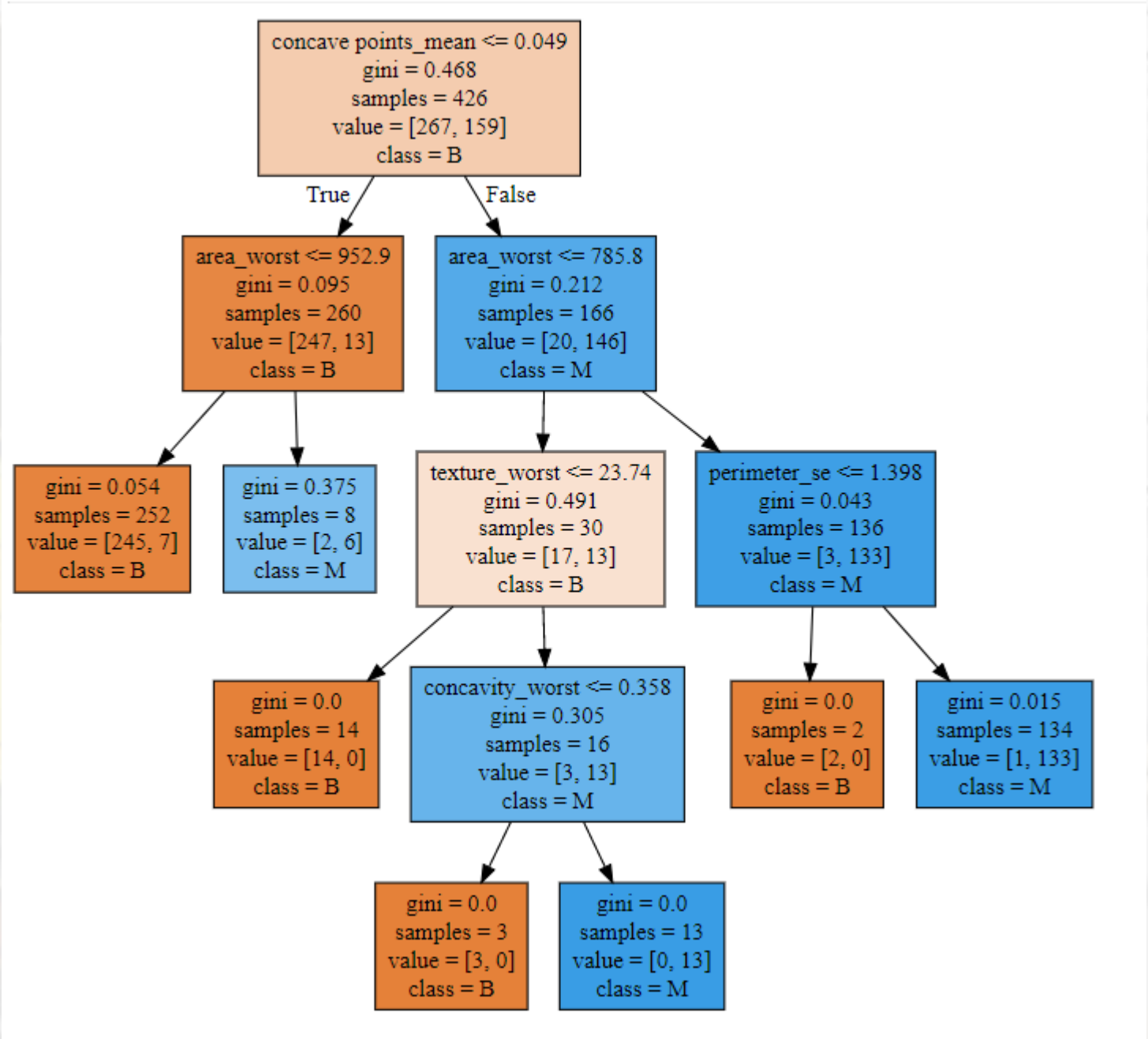
- Breast cancer dataset

```
dt3=DecisionTreeClassifier(max_leaf_nodes=7,random_state=0)
dt3.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt3.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt3.score(X_test,y_test)))
export_graphviz(dt3,out_file='tree_vis3.dot',class_names=dt3.classes_,
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 0.977
Decision tree acc on test: 0.958
```

Visualized Decision Tree

•



Feature Importance in Trees

- Instead of going over the whole tree and all features, we can get some valuable summaries
- The most commonly used summary is feature importance
- Feature importance rates how important each feature is for a decision a tree makes
- It is a number between 0 and 1 for each feature
  - 0 means “not used at all”
  - 1 means “perfectly predicts the target”
- The summation of feature importances is 1

Implementation in Python

- .

```
dt3.feature_importances_
```

```
array([0.        , 0.0096886 , 0.        , 0.        , 0.        ,
       0.        , 0.        , 0.71160121, 0.        , 0.        ,
       0.01948008, 0.        , 0.        , 0.01676117, 0.017502  ,
       0.        , 0.        , 0.        , 0.        , 0.        ,
       0.        , 0.06706044, 0.        , 0.11373562, 0.        ,
       0.03421113, 0.00995974, 0.        , 0.        , 0.        ])
```

Implementation in Python

```
feature_imp=pd.DataFrame(data=dt3.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

	importance
concave points_mean	0.711601
area_worst	0.113736
texture_worst	0.067060
compactness_worst	0.034211
radius_se	0.019480
smoothness_se	0.017502
area_se	0.016761
concavity_worst	0.009960
texture_mean	0.009689
radius_worst	0.000000

Feature Importance in Trees

- Usually, the feature used in the top split is by far the most important feature
- If a feature has a low value in feature_importance_, it does not necessarily mean that this feature is uninformative
- It might mean that this feature was not picked by the tree, likely because another feature had the same information

Feature Importance in Trees

- Diabetes dataset

```
X,y=diabete.iloc[:,:-1],diabete['Outcome']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
from sklearn.tree import DecisionTreeClassifier
dt2=DecisionTreeClassifier(random_state=0)
dt2.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt2.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt2.score(X_test,y_test)))
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.719
```

```
from sklearn.tree import export_graphviz
export_graphviz(dt2,out_file='tree_vis2.dot',class_names=dt2.classes_,
                feature_names=X.columns,impurity=False,filled=True)
```

```
dt2.feature_importances_
```

```
array([0.06554869, 0.31731807, 0.11957134, 0.04184143, 0.03422922,
       0.14935508, 0.15604779, 0.11608838])
```

Featu

- Diabetes dataset

```
dt2=DecisionTreeClassifier(random_state=0,min_samples_split=100)
dt2.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt2.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt2.score(X_test,y_test)))
```

```
Decision tree acc on train: 0.781
Decision tree acc on test: 0.781
```

```
dt2.feature_importances_
```

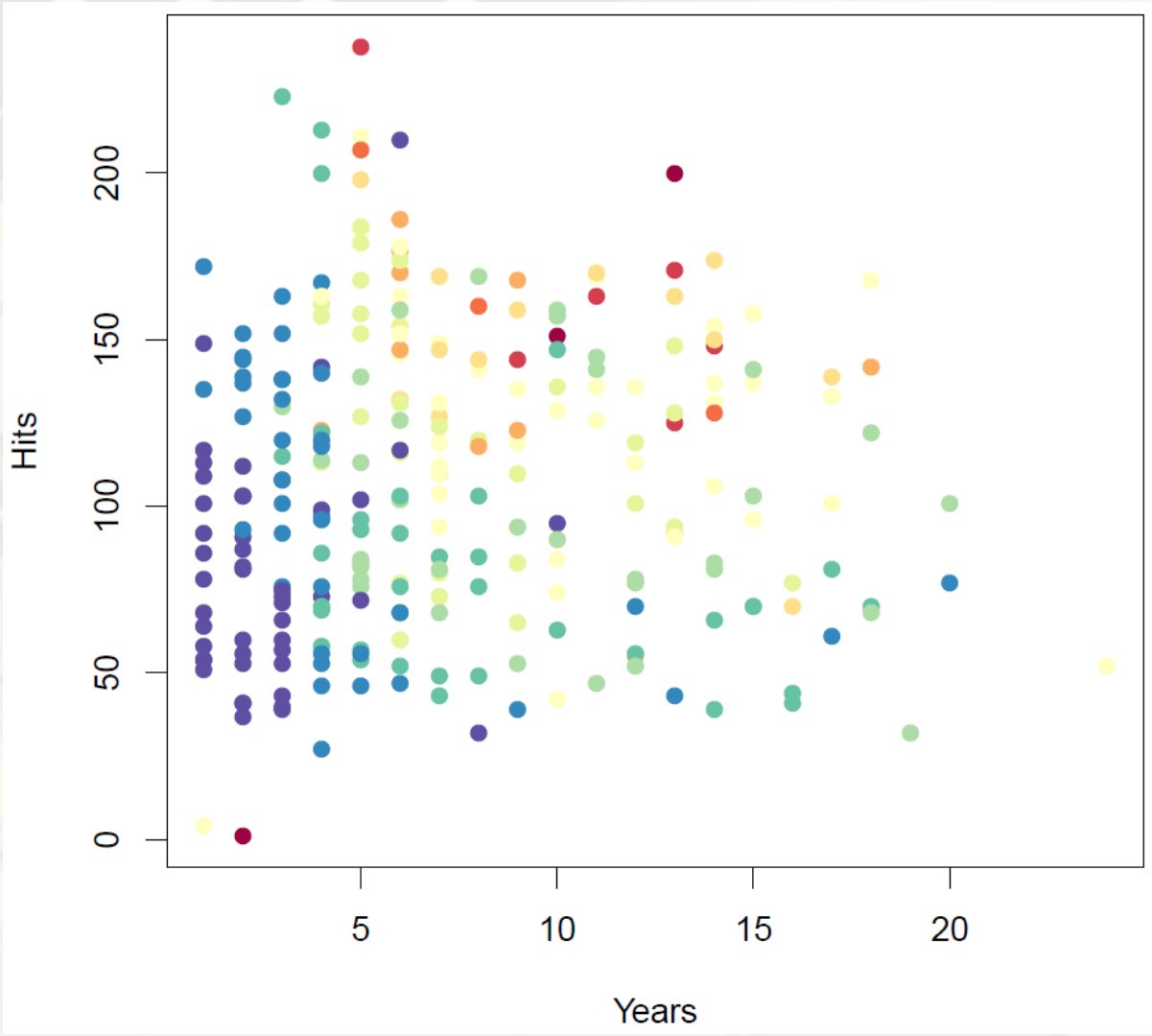
```
array([0.0194008 , 0.61195884, 0.02020199, 0.        , 0.        ,
       0.14594473, 0.03979489, 0.16269875])
```

```
X.columns
feature_imp=pd.DataFrame(data=dt2.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

	importance
Glucose	0.611959
Age	0.162699
BMI	0.145945
DiabetesPedigreeFunction	0.039795
BloodPressure	0.020202
Pregnancies	0.019401
SkinThickness	0.000000
Insulin	0.000000

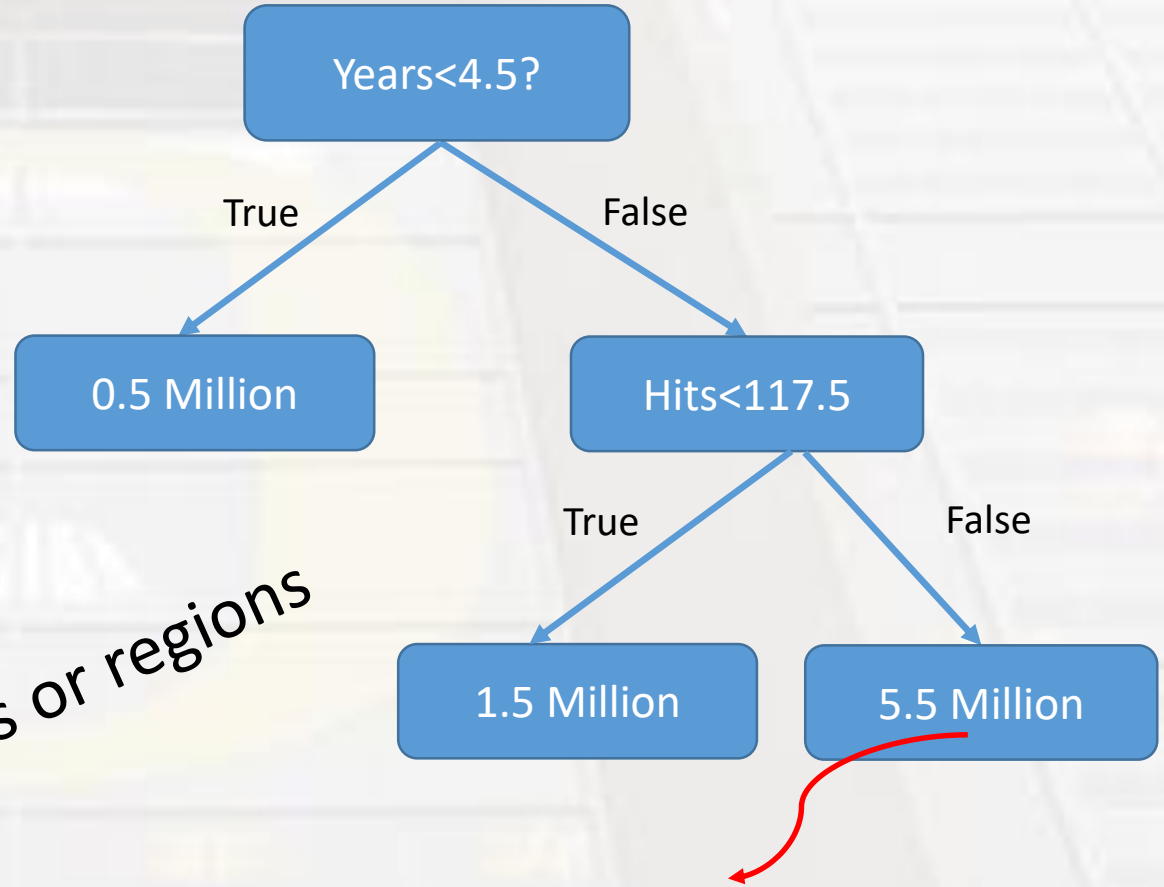
Decision Trees as Regression

- Baseball salary data : How to predict baseball player salaries
  - Target : Salary,
  - Predictors : Years, Hits
  - Salary is color-coded from low (blue, green) to high (yellow, red)



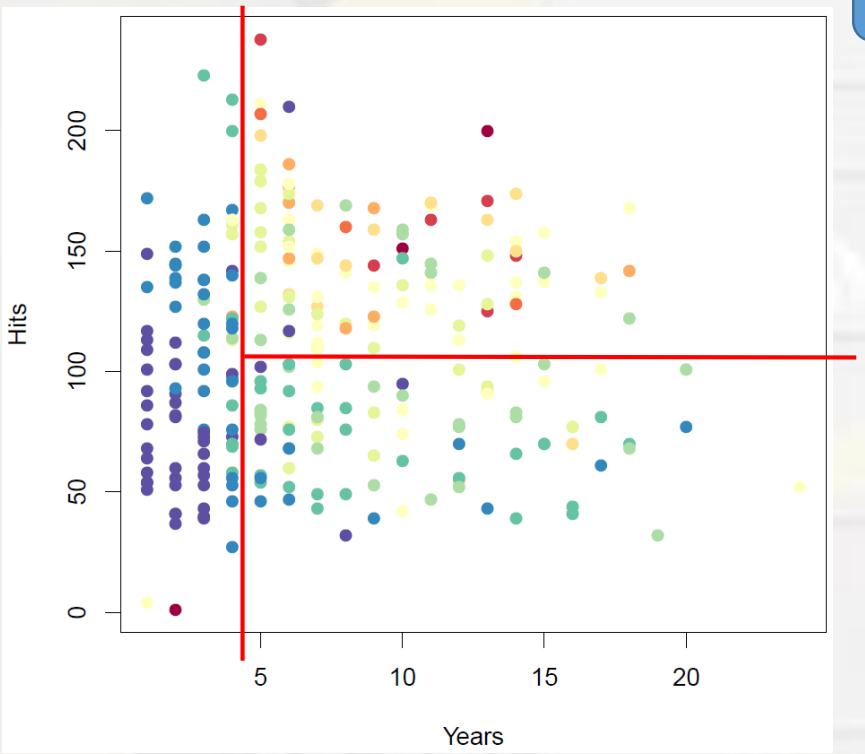
Decision Trees as Regression

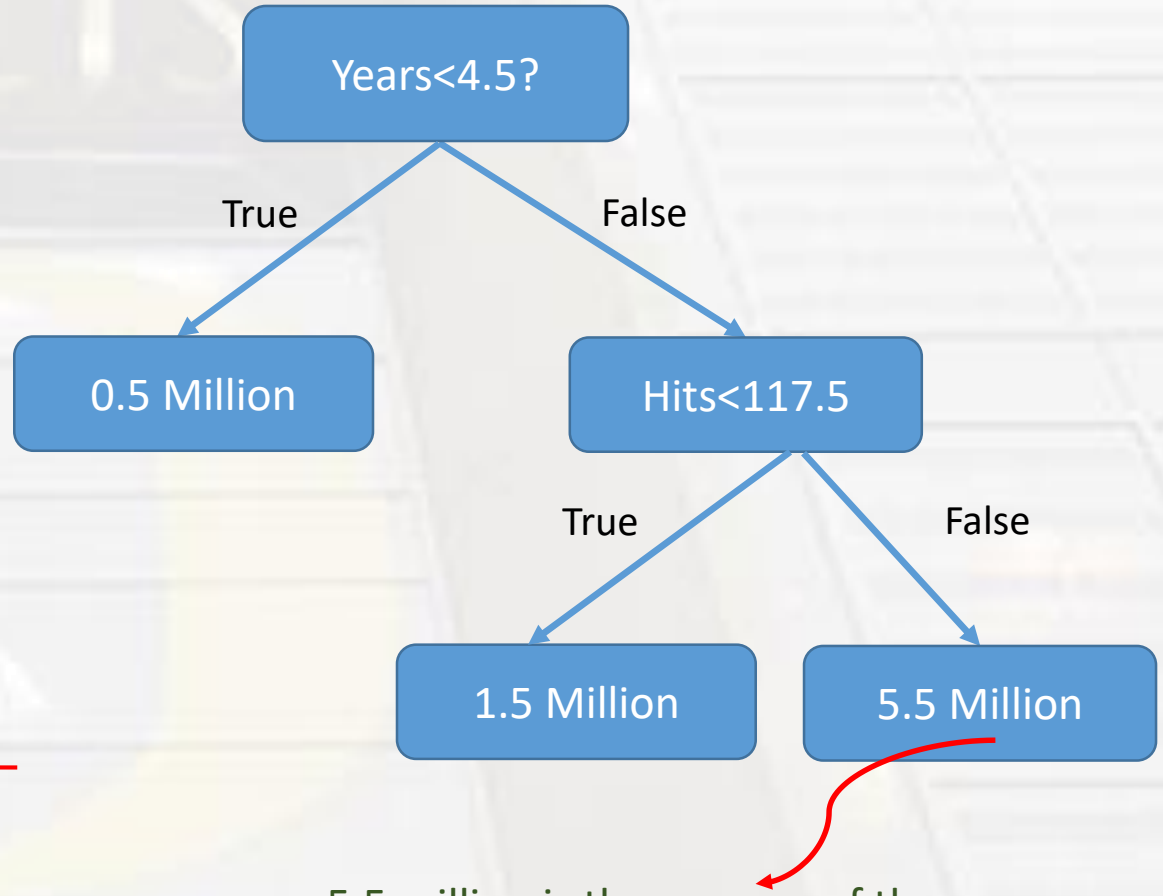
- A regression tree for predicting the salary of a baseball player, based on the number of years that he has played in the major leagues and the number of hits that he made in the previous year.



Interpretation

- Years is the most important variable
- Hits do not play an important role for less experienced players
- Hits play role for high experienced players





5.5 million is the average of the response for the observations that fall in this node

MSE in Regression Trees

- In developing a regression tree, the algorithm tries to minimize the MSE
- Mean squared error (MSE) measures the deviation from the true target values in each node
- $e_i = y_i - \hat{y}_i$ prediction error for each data point
- MSE= $\frac{1}{n}\sum_{i=1}^{n} e_i^2$

Regression Trees in Python

- boston_housing_data

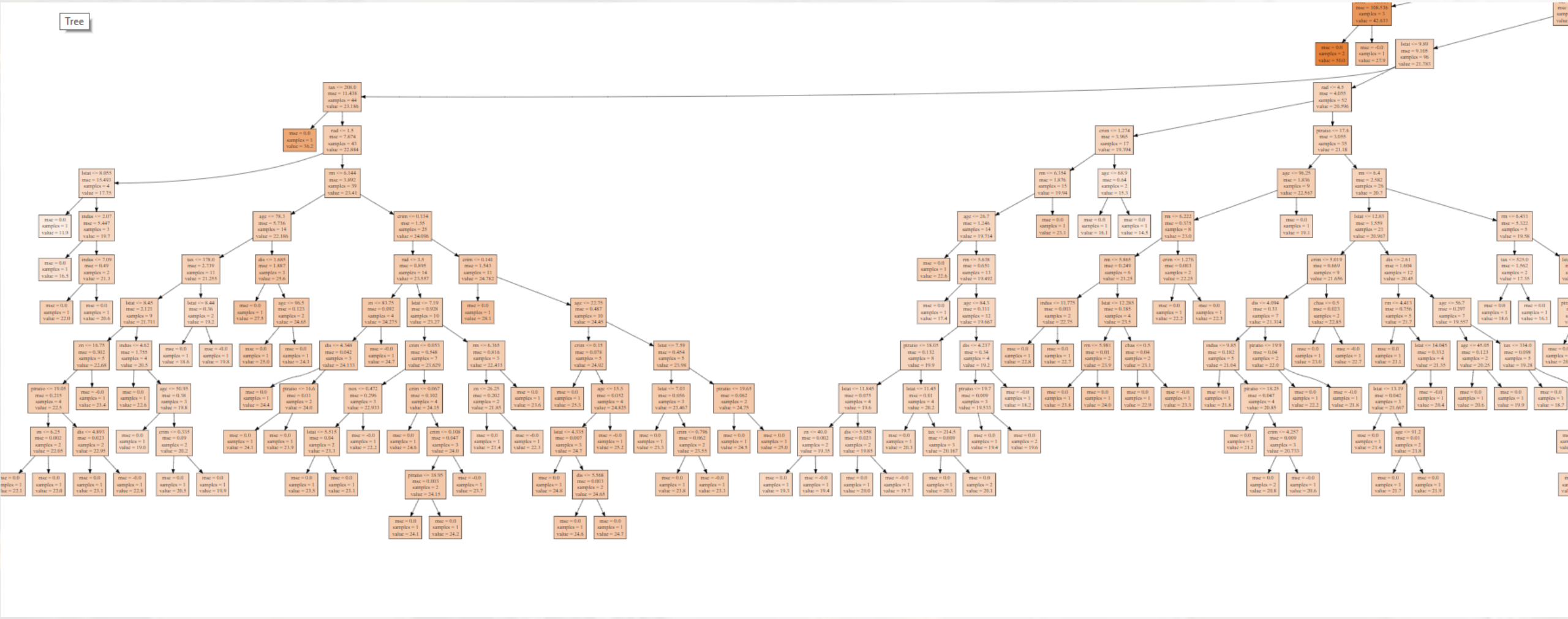
```
boston=pd.read_csv('boston_housing_data.csv',index_col=0)
from sklearn.tree import DecisionTreeRegressor
X,y=boston.iloc[:,:-1],boston['target_medv']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt4=DecisionTreeRegressor(random_state=0)
dt4.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt4.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt4.score(X_test,y_test)))
export_graphviz(dt4,out_file='tree_vis4.dot',
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.871
```

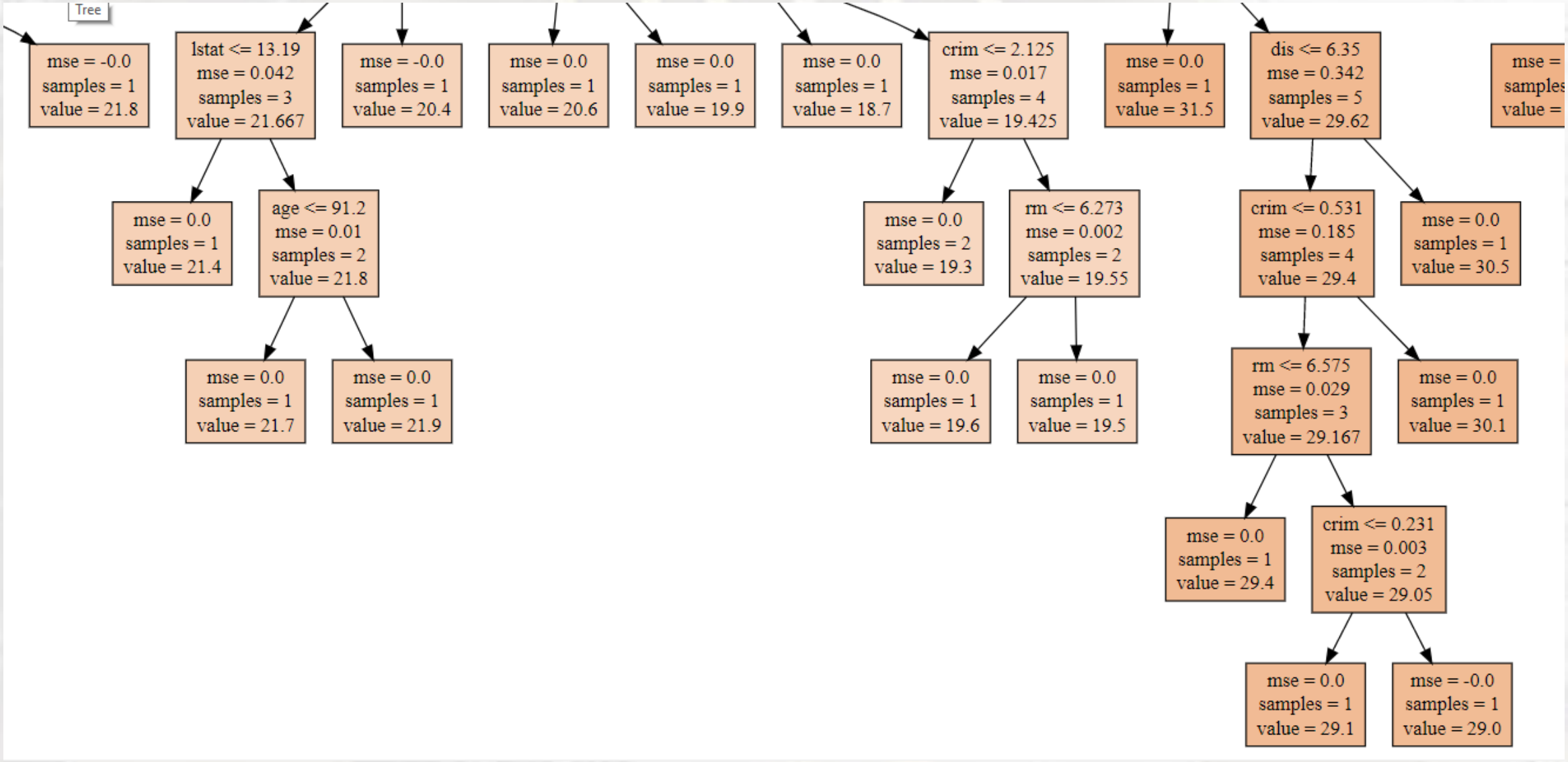
UNIVERSITY OF OREGON
Lundquist College of Business

Regression Trees in Python

- boston_housing_data



Regression Trees in Python



Regression Trees in Python

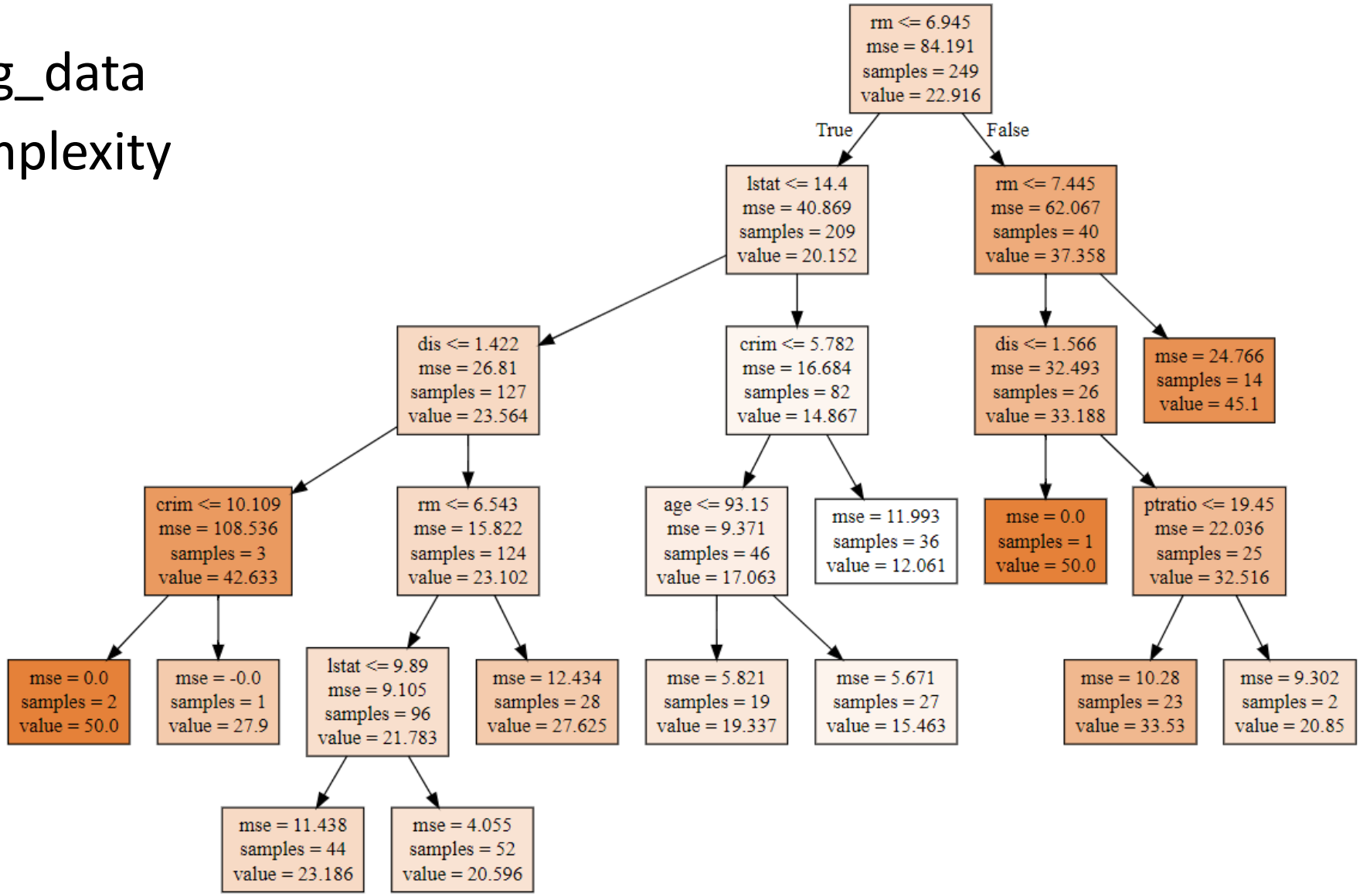
- boston_housing_data (control for complexity of the tree)

```
boston=pd.read_csv('boston_housing_data.csv',index_col=0)
from sklearn.tree import DecisionTreeRegressor
X,y=boston.iloc[:,:-1],boston['target_medv']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt4=DecisionTreeRegressor(random_state=0,max_leaf_nodes=12)
dt4.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt4.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt4.score(X_test,y_test)))
export_graphviz(dt4,out_file='tree_vis4.dot',
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 0.887
Decision tree acc on test: 0.899
```

Regression Trees in Python

- boston_housing_data (control for complexity of the tree)



Regression Trees in Python

- WestRoxbury

```
house=pd.read_csv('WestRoxbury.csv')
X,y=house.iloc[:,1:],house['TOTAL VALUE ']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt5=DecisionTreeRegressor(random_state=0)
dt5.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt5.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt5.score(X_test,y_test)))
export_graphviz(dt5,out_file='tree_vis5.dot',
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 1.000
Decision tree acc on test: 0.658
```

Regression Trees in Python

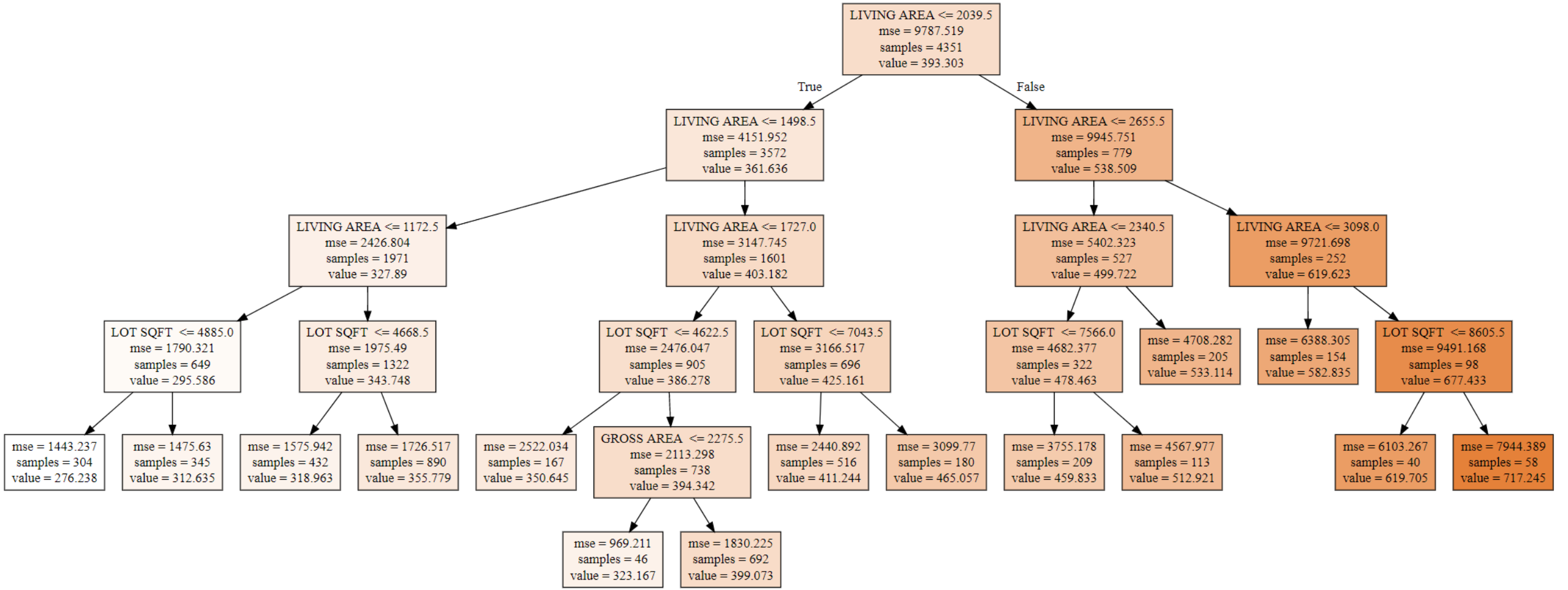
- WestRoxbury (control for complexity of the tree)

```
house=pd.read_csv('WestRoxbury.csv')
X,y=house.iloc[:,1:],house['TOTAL VALUE ']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt5=DecisionTreeRegressor(random_state=0,max_leaf_nodes=15)
dt5.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt5.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt5.score(X_test,y_test)))
export_graphviz(dt5,out_file='tree_vis5.dot',
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 0.749
Decision tree acc on test: 0.687
```

Regression Trees in Python

- WestRoxbury (control for complexity of the tree)



Feature Importance

```
]: feature_imp=pd.DataFrame(data=dt5.feature_importances_,index=X.columns,columns=['importance'])
   feature_imp.sort_values('importance',ascending=False)
```

]:

	importance
LIVING AREA	0.939365
LOT SQFT	0.052848
GROSS AREA	0.007787
YR BUILT	0.000000
FLOORS	0.000000
ROOMS	0.000000

Strength, Weaknesses, and Parameters

- Pre-pruning parameters to control the model's complexity:
  - max_depth, max_leaf_nodes, min_samples_split
- Decision trees can easily be visualized and understood by non-experts
- They work pretty well with features in different scales
- The main downside is that, a lot of times even with the use of pre-pruning, they tend to over-fit and have poor generalization

Random Forests

Random Forests

- The main drawback of decision trees: tending to overfit the training data
- Random forests are a way to address this problem
- A random forest is essentially a collection of trees
- Each tree is slightly different from the other trees
- The idea behind random forests is that each tree might do a good job of predicting, but they tend to overfit on different parts of the data
- If we build many trees, we can reduce the overfitting by averaging their results

Random Forests

- To implement this strategy, we need to build many decision trees
- Each tree should do an acceptable job of predicting the target and also should be slightly different from others
- Random forests get their name from injecting randomness into the tree building to ensure each tree is different
- Two ways to randomize the development of trees:
  - By randomly selecting the data points (records) used to build each tree
  - By randomly selecting the features to split the data space

Random Forests-Classification

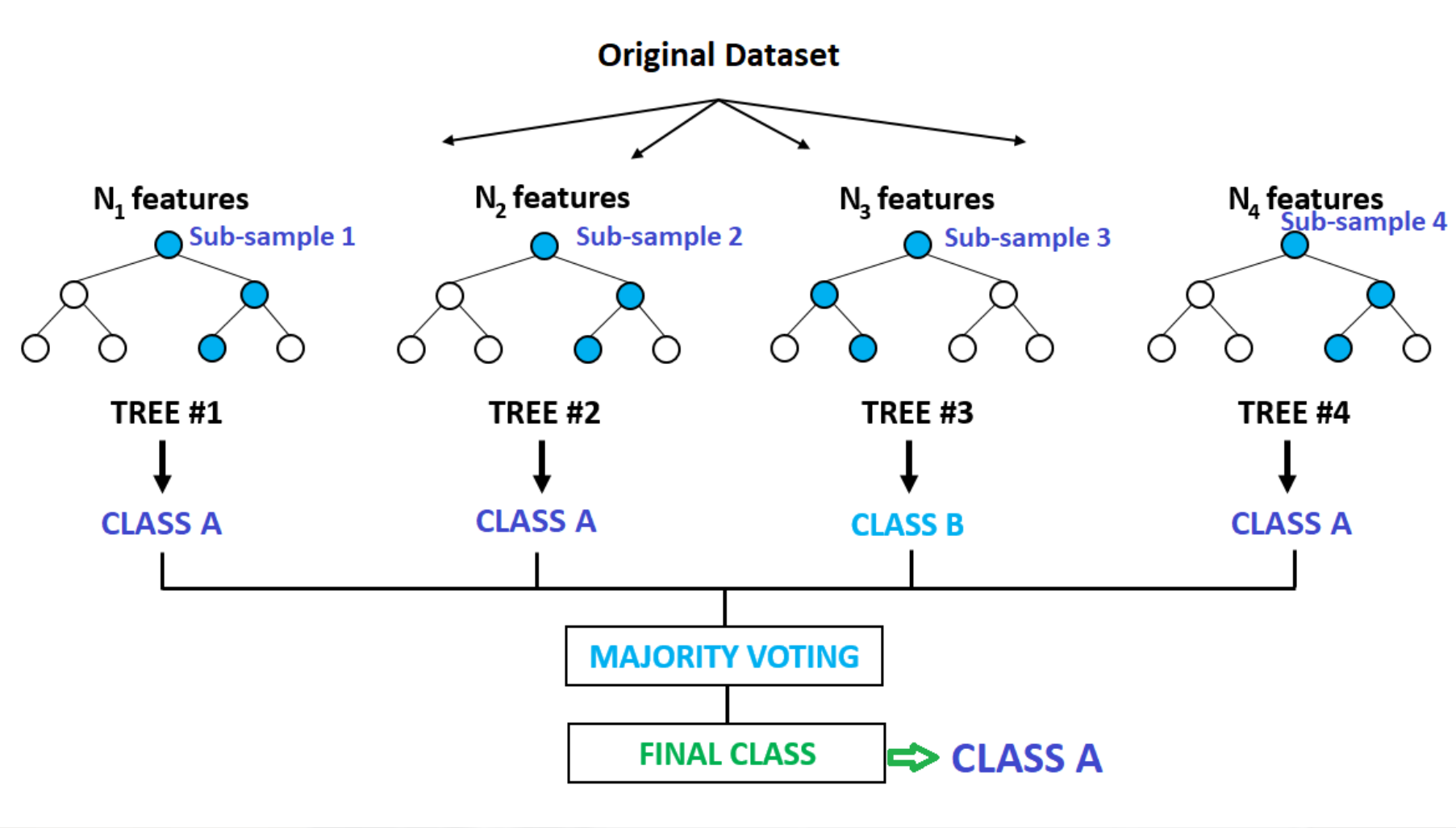
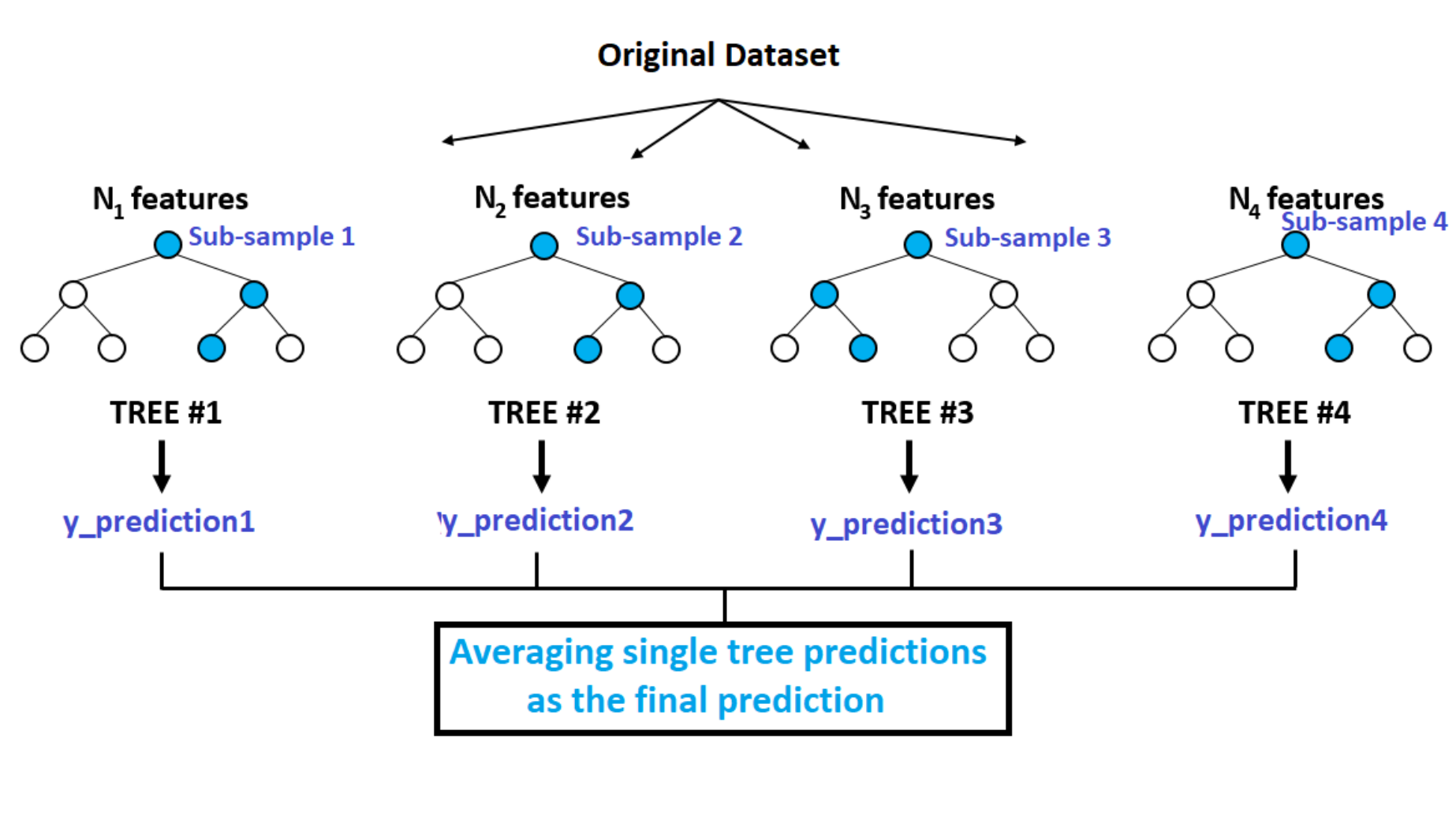


Photo adopted from: https://medium.com/dataseries/machine-learning-series-507f9b21c9e5 (Edited afterwards)

Random Forests-Regression



Random Forests-Regression

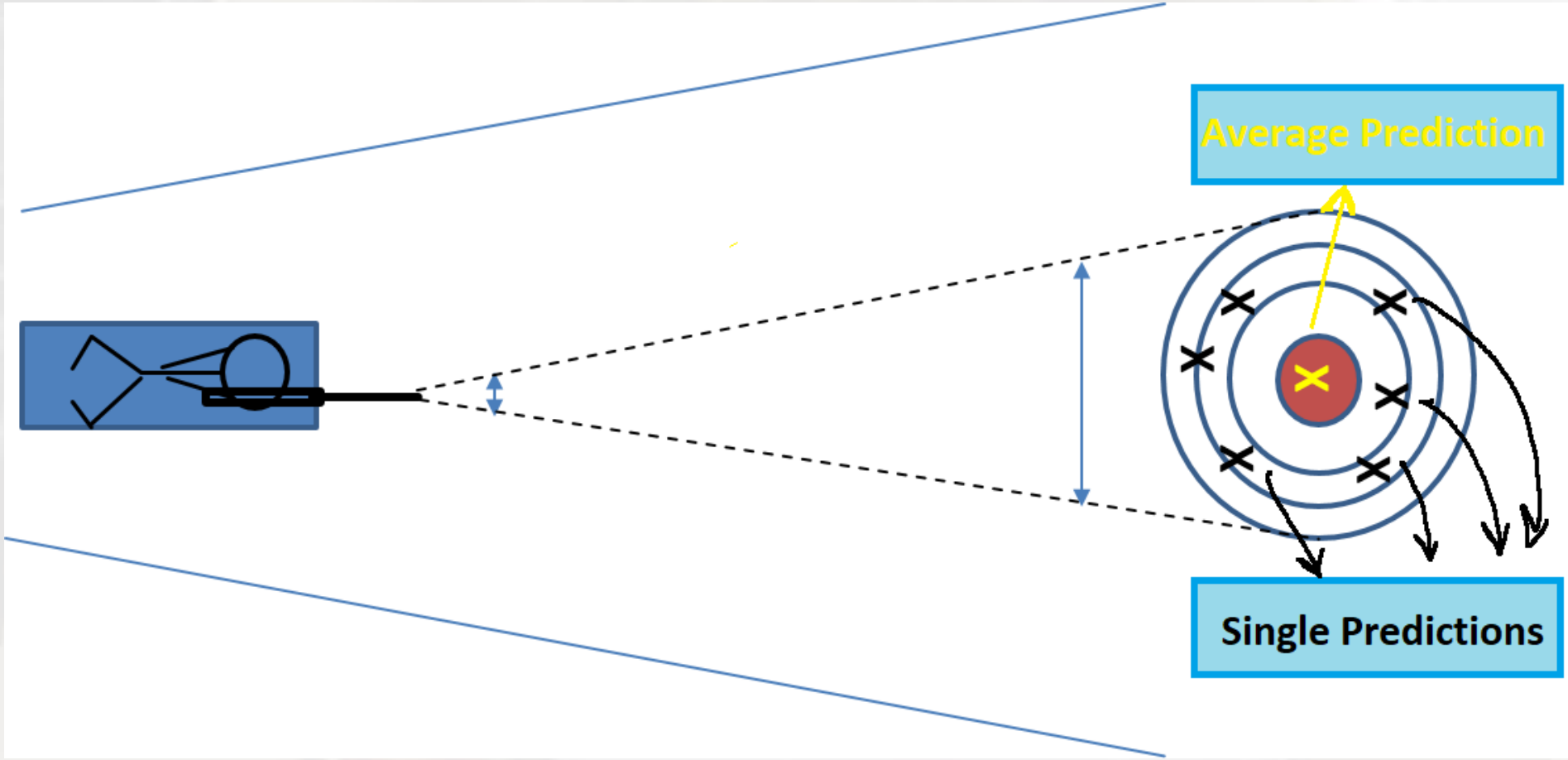


Photo adopted from: https://www.python-course.eu/Random_Forests.php (Edited afterwards)

Random Forests

- To build a random forest, you need to decide on the number of trees to build
  - This will be done using the ***n_estimators*** parameter
  - Usually higher ***n_estimators*** is better and reduces over-fitting, however, it will require more time and memory
- To make a prediction using the random forest, the algorithm first make a prediction for every tree in the forest, then
  - For regression, we will average these results to get our final prediction
  - For classification, we will average the output probabilities of all trees and use the average probability to make the final prediction

Random Forests in Python

- Pre-pruned decision tree on breast cancer data

```
cancer=pd.read_csv('breast_cancer_data.csv',index_col=0)
X,y=cancer.iloc[:,1:],cancer['diagnosis']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
from sklearn.tree import DecisionTreeClassifier
dt=DecisionTreeClassifier(random_state=0,max_leaf_nodes=7)
dt.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt.score(X_test,y_test)))
```

```
Decision tree acc on train: 0.977
Decision tree acc on test: 0.958
```

Random Forests in Python

- Random forest on breast cancer data

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf=RandomForestClassifier(random_state=0,n_estimators=100)
rf.fit(X_train,y_train)
print('Random Forest acc on train: {:.3f}'.format(rf.score(X_train,y_train)))
print('Random Forest acc on test: {:.3f}'.format(rf.score(X_test,y_test)))
```

```
Random Forest acc on train: 1.000
Random Forest acc on test: 0.972
```

Random Forests in Python

- Random forests also provide feature importance
- They are computed by aggregating the feature importnaces over the trees in the forest
- Typically, feature importances provided by random forests are more reliable than the ones provided by single trees
- We can get them by attribute, ***feature_importances_***

Random Forests in Python

```
rf.feature_importances_
```

```
array([0.00254465, 0.01873653, 0.00472889, 0.00482262, 0.00580242,
       0.0012574 , 0.00917384, 0.20171831, 0.0034509 , 0.00194409,
       0.00316387, 0.00486969, 0.00385039, 0.02024855, 0.00276736,
       0.00292138, 0.00635463, 0.00117784, 0.00228943, 0.00610244,
       0.05289488, 0.02836345, 0.30421444, 0.0602248 , 0.00947784,
       0.0077201 , 0.01738994, 0.20376772, 0.00608876, 0.0019328 ])
```

```
feature_imp=pd.DataFrame(data=rf.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

	importance
perimeter_worst	0.160277
concave points_worst	0.117118
radius_worst	0.116573
concave points_mean	0.103054
area_worst	0.063469
concavity_mean	0.055395
perimeter_mean	0.053727

Random Forests in Python

- Pre-pruned decision tree on diabetes data

```
X,y=diabete.iloc[:,:-1],diabete['Outcome']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
from sklearn.tree import DecisionTreeClassifier
dt2=DecisionTreeClassifier(random_state=0,min_samples_split=100)
dt2.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt2.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt2.score(X_test,y_test)))
```

```
Decision tree acc on train: 0.781
Decision tree acc on test: 0.781
```

```
dt2.feature_importances_
```

```
array([0.0194008 , 0.61195884, 0.02020199, 0.        , 0.        ,
       0.14594473, 0.03979489, 0.16269875])
```

Random Forests in Python

```
rf2=RandomForestClassifier(random_state=0,n_estimators=700)
rf2.fit(X_train,y_train)
print('Random Forest acc on train: {:.3f}'.format(rf2.score(X_train,y_train)))
print('Random Forest acc on test: {:.3f}'.format(rf2.score(X_test,y_test)))
```

```
Random Forest acc on train: 1.000
Random Forest acc on test: 0.792
```

```
X.columns
feature_imp=pd.DataFrame(data=rf2.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

	importance
Glucose	0.248904
BMI	0.167602
Age	0.144347
DiabetesPedigreeFunction	0.125590
BloodPressure	0.087404
Pregnancies	0.081092

- Random forest on diabetes data

Regression Random Forests in Python

- Pre-pruned decision tree on Boston housing data

```
boston=pd.read_csv('boston_housing_data.csv',index_col=0)
from sklearn.tree import DecisionTreeRegressor
X,y=boston.iloc[:,:-1],boston['target_medv']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt4=DecisionTreeRegressor(random_state=0,max_leaf_nodes=12)
dt4.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt4.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt4.score(X_test,y_test)))
```

```
Decision tree acc on train: 0.887
Decision tree acc on test: 0.899
```

```
dt4.feature_importances_
```

```
array([0.04466081, 0.        , 0.        , 0.        , 0.        ,
       0.64341902, 0.00899586, 0.07585795, 0.        , 0.        ,
       0.01590292, 0.21116344])
```

Regression Random Forests in Python

```
from sklearn.ensemble import RandomForestRegressor
```

```
rf3=RandomForestRegressor(random_state=0,n_estimators=100)
rf3.fit(X_train,y_train)
print('Random Forest acc on train: {:.3f}'.format(rf3.score(X_train,y_train)))
print('Random Forest acc on test: {:.3f}'.format(rf3.score(X_test,y_test)))
```

```
Random Forest acc on train: 0.975
Random Forest acc on test: 0.919
```

```
rf3.feature_importances_
```

```
array([0.04645699, 0.00139178, 0.00789442, 0.00187887, 0.01385366,
       0.39824902, 0.0171968 , 0.04987211, 0.00870603, 0.02069426,
       0.01095033, 0.42285575])
```

- Random forest on Boston housing data

Regression Random Forests in Python


```
house=pd.read_csv('WestRoxbury.csv')
X,y=house.iloc[:,1:],house['TOTAL VALUE ']
X_train, X_test,y_train,y_test=train_test_split(X,y,random_state=0)
dt5=DecisionTreeRegressor(random_state=0,max_leaf_nodes=15)
dt5.fit(X_train,y_train)
print('Decision tree acc on train: {:.3f}'.format(dt5.score(X_train,y_train)))
print('Decision tree acc on test: {:.3f}'.format(dt5.score(X_test,y_test)))
export_graphviz(dt5,out_file='tree_vis5.dot',
                feature_names=X.columns,impurity=True,filled=True)
```

```
Decision tree acc on train: 0.749
Decision tree acc on test: 0.687
```

```
feature_imp=pd.DataFrame(data=dt5.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

	importance
LIVING AREA	0.939365
LOT SQFT	0.052848
GROSS AREA	0.007787
YR BUILT	0.000000

- Pre-pruned decision tree on WestRoxbury data

Regression Random Forests in Python

```
rf4=RandomForestRegressor(random_state=0,n_estimators=200)
rf4.fit(X_train,y_train)
print('Random Forest acc on train: {:.3f}'.format(rf4.score(X_train,y_train)))
print('Random Forest acc on test: {:.3f}'.format(rf4.score(X_test,y_test)))
feature_imp=pd.DataFrame(data=rf4.feature_importances_,index=X.columns,columns=['importance'])
feature_imp.sort_values('importance',ascending=False)
```

```
Random Forest acc on train: 0.976
Random Forest acc on test: 0.808
```

	importance
LIVING AREA	0.726795
LOT SQFT	0.093983
GROSS AREA	0.066999
YR BUILT	0.034342
FLOORS	0.016210
FIREPLACE	0.012872
REMODEL	0.011802
ROOMS	0.011325

- Random forest on WestRoxbury data