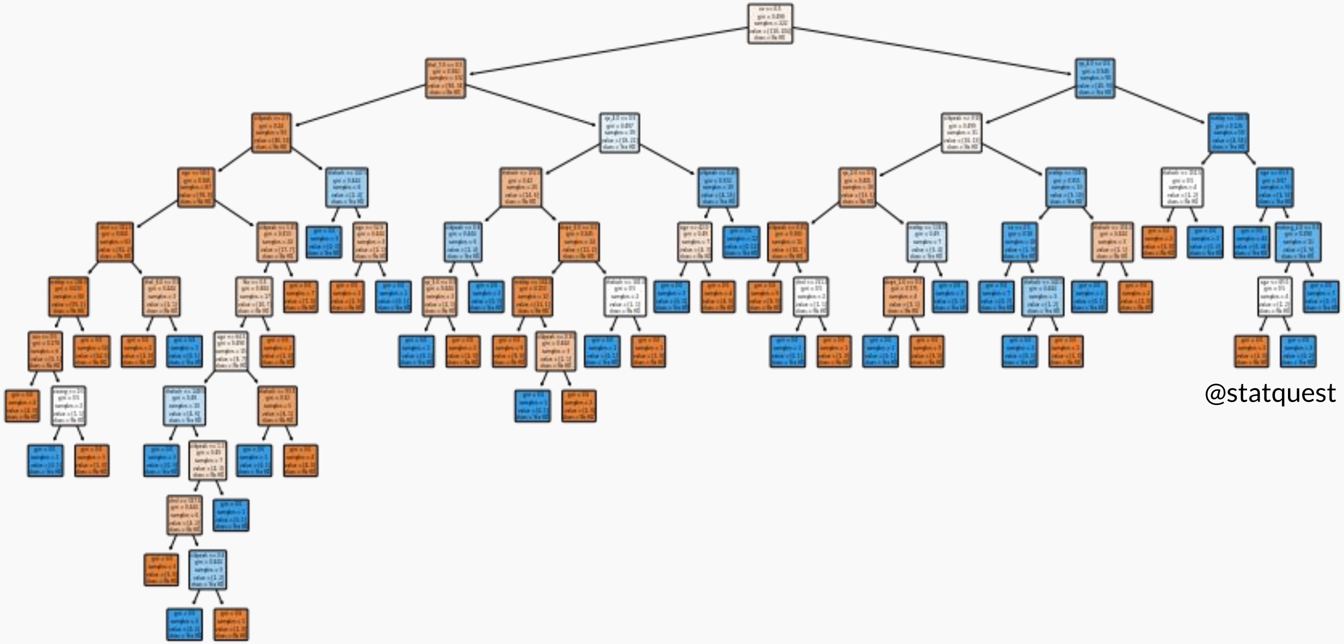
Lesson #04
Decision Trees

and more...

October, 2020

Table of Contents

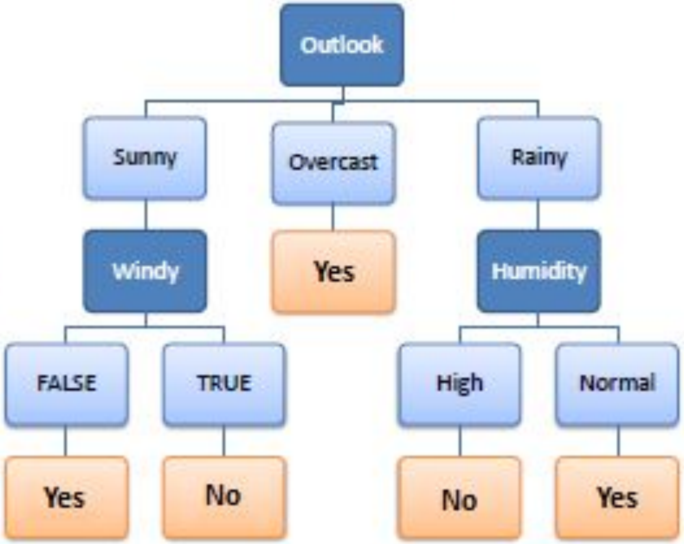
How can a dataset be represented as a tree?



Decision Tree (classification)

Predictors				Target
Outlook	**Temp**	**Humidity**	**Windy**	**Play Golf**
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

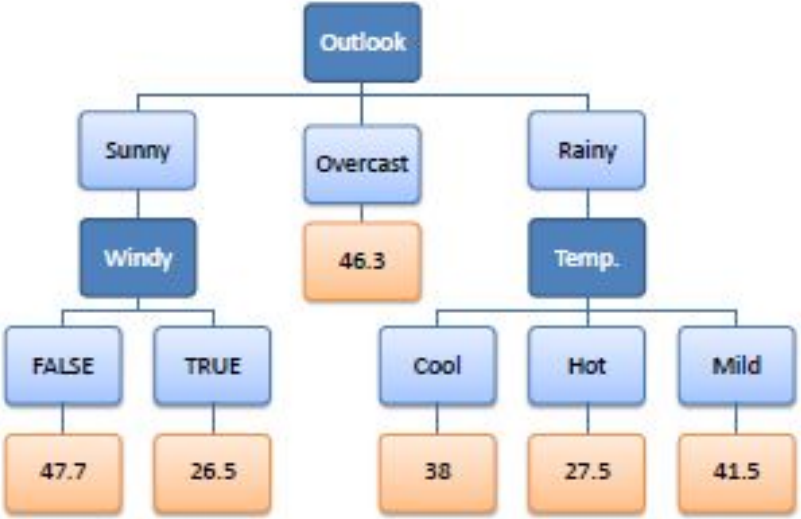
@Rishabh Jain



Decision Tree (regression)

Predictors				Target
Outlook	**Temp**	**Humidity**	**Windy**	**Hours Played**
Rainy	Hot	High	False	26
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	62
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30

@Rishabh Jain




CASE STUDY

Adult Data Set

Download: Data Folder, Data Set Description

Abstract: Predict whether income exceeds $50K/yr based on census data. Also known as "Census Income" dataset.

Data Set Characteristics:	Multivariate	**Number of Instances:**	48842	**Area:**	Social
Attribute Characteristics:	Categorical, Integer	**Number of Attributes:**	14	**Date Donated**	1996-05-01
Associated Tasks:	Classification	**Missing Values?**	Yes	**Number of Web Hits:**	1564397

Source:

Donor:

Ronny Kohavi and Barry Becker
Data Mining and Visualization
Silicon Graphics.
e-mail: ronnyk '@' live.com for questions.

Data Set Information:

Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: ((AAGE>16) && (AGI>100) && (AFNLWGT>1)&& (HRSWK>0))

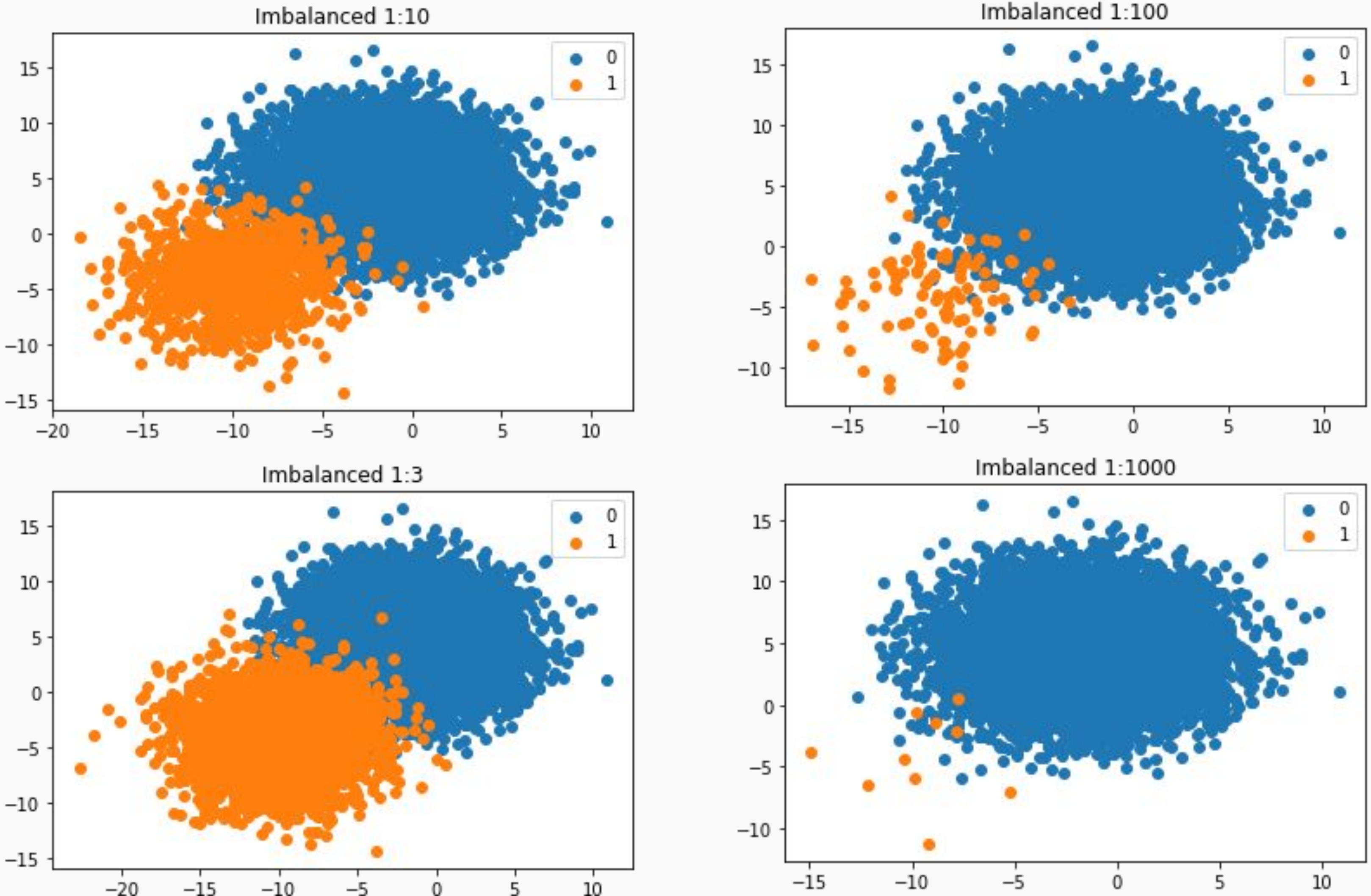
Prediction task is to determine whether a person makes over 50K a year.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education_num   32561 non-null  int64
 5   marital_status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital_gain    32561 non-null  int64
 11  capital_loss    32561 non-null  int64
 12  hours_per_week  32561 non-null  int64
 13  native_country  32561 non-null  object
 14  high_income     32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Cardinality

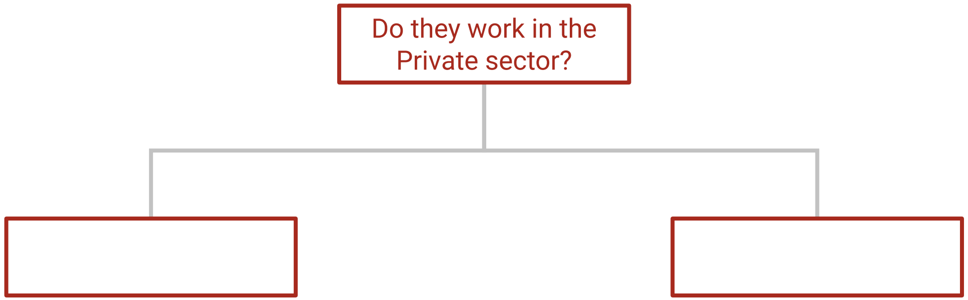
```
workclass          9
education         16
marital_status     7
occupation        15
relationship       6
race               5
sex                2
native_country    42
high_income        2
dtype: int64
```

```
 <=50K    24720
 >50K      7841
Name: high_income, dtype: int64
```


Imbalanced 1:10
0
1
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
Imbalanced 1:100
0
1
15
10
5
0
−5
−10
−15
−10
−5
0
5
10
Imbalanced 1:3
0
1
15
10
5
0
−5
−10
−15
−20
−15
−10
−5
0
5
10
Imbalanced 1:1000
0
1
15
10
5
0
−5
−10
−15
−10
−5
0
5
10

How can we split the tree?

What income do people make?



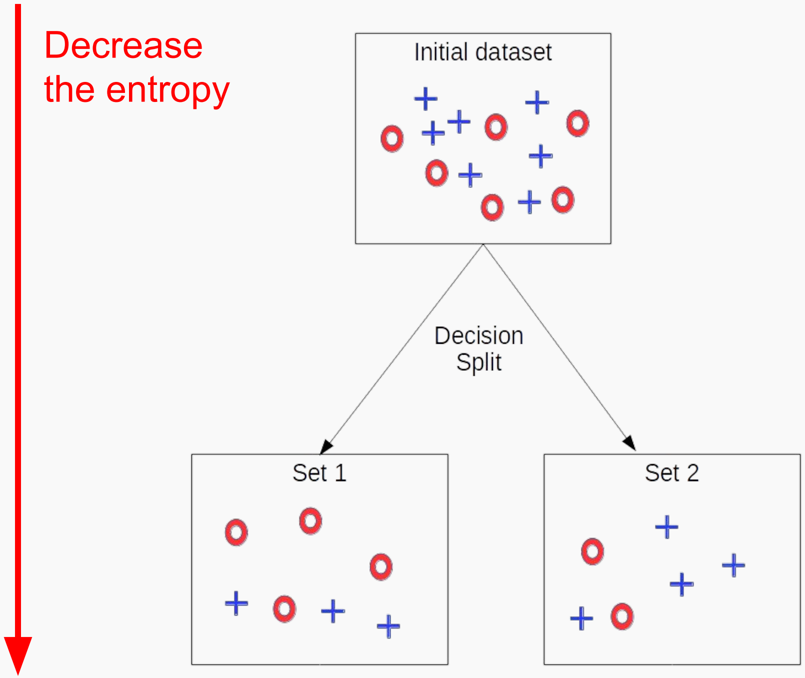
Algorithm used in Decision Trees

1. ID3 (Entropy)
2. Gini Index
3. Chi-Square
4. Reduction in Variance
    a. C4.5, pruning

Entropy

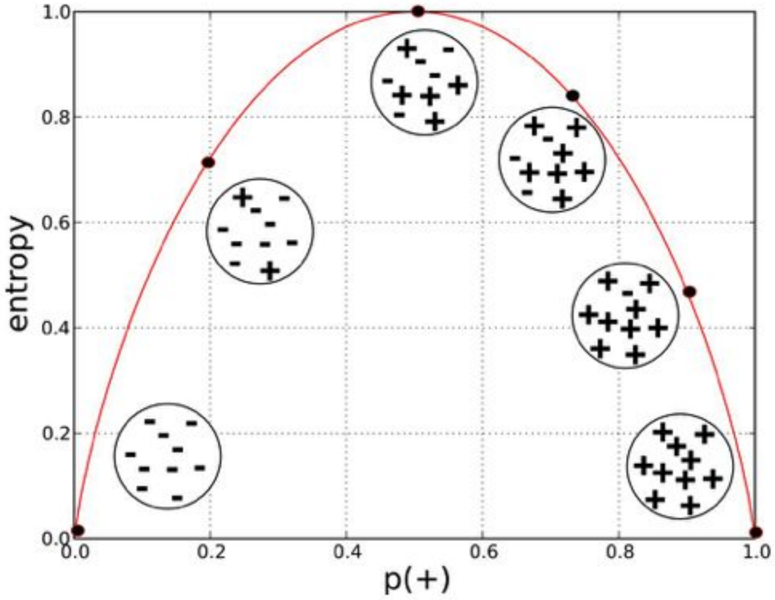
Entropy is an indicator of how messy your data is.

Why Entropy in Decision Trees?



- The goal is to tidy the data.
- You try to separate your data and group the samples together in the classes they belong to.
- You maximize the purity of the groups as much as possible each time you create a new node of the tree
- Of course, at the end of the tree, you want to have a clear answer.

Mathematical definition of entropy



Suppose a set of N items, these items fall into two categories:

- [+] gain > 50k (k)
- [-] gain <= 50k (m)

$$p = \frac{k}{N}, q = \frac{m}{N}$$

$$Entropy = -p \log p - q \log q$$

Generalization

Feature X

$$E(x) = -\sum_{i=1}^{c} P(x_i) \log_b P(x_i)$$

$P(x_i)$ is the fraction of examples in a given class i

```
<=50K     24720
>50K       7841
Name: high_income, dtype: int64
```

```
entropy(df.high_income.value_counts(),base=2)
0.7963839552022132
```

Entropy using the frequency table of two attributes

		High Income		
		<=50k	>50k	
Age	<= 37	14421	2260	16681 (51.22%)
	> 37	10299	5581	15880 (48.77%)

$$E(T|X) = \sum_{c \in X} \frac{|X_c|}{|X|} E(T|X_c)$$

```
cross = pd.crosstab(data.age <= data.age.median(),
                    data.high_income)
```

high_income	<=50K	>50K
age		
False	10299	5581
True	14421	2260

```
0.4877 * entropy(cross.iloc[0],base=2) + 0.5122 \
  * entropy(cross.iloc[1],base=2)

0.7492980618394182
```

Information Gain

$$IG(T,X) = E(T) - E(T|X)$$

Information Gain from X on T

The information gain is based on the **decrease in entropy after a dataset is split** on an attribute.

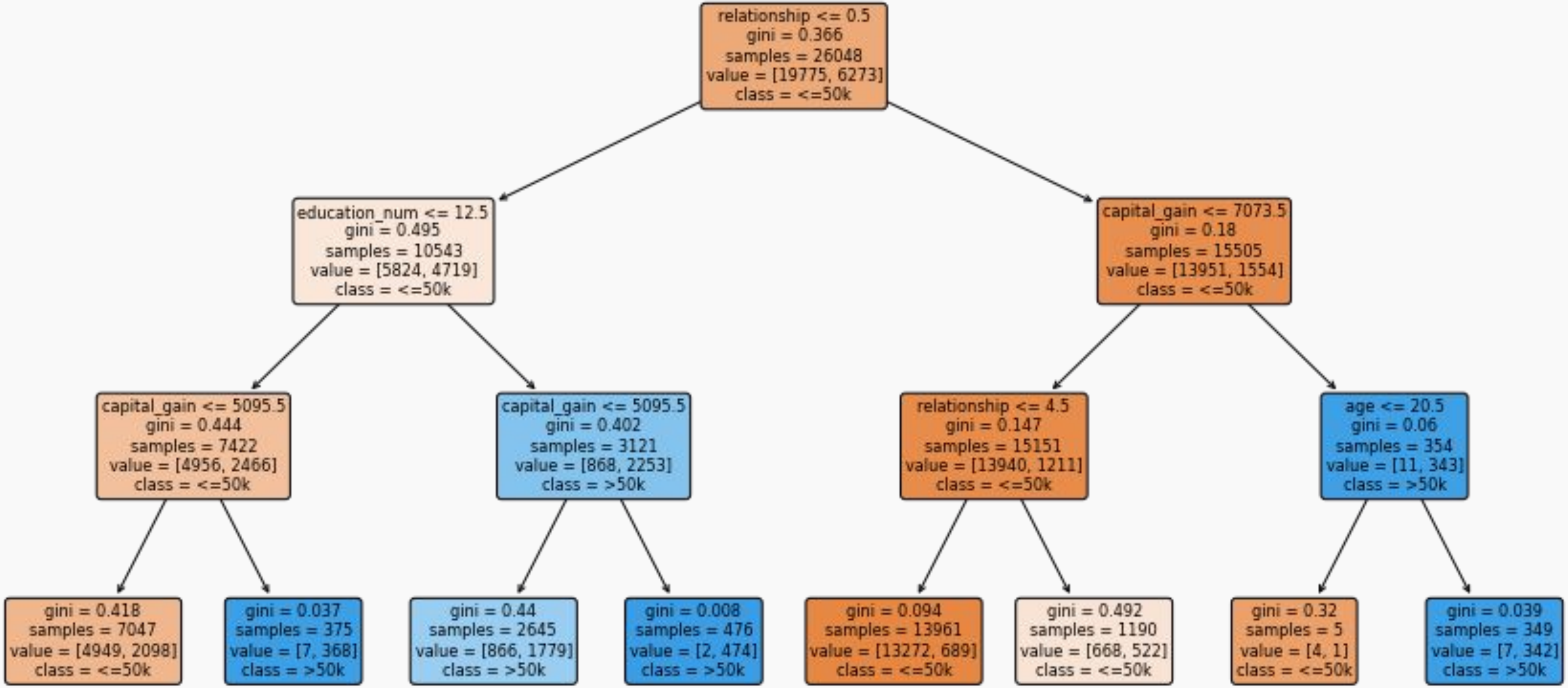
Constructing a decision tree is all about finding attribute that returns the **highest information gain** (i.e., the most homogeneous branches).

$$Gini(x) = 1 - \sum_{i=1}^{c} P(x_i)^2$$

$$Entropy(x) = -\sum_{i=1}^{c} P(x_i) \log_b P(x_i)$$

Gini index or Entropy is the criterion for calculating **Information Gain**. Both of them are measures of impurity of a node.

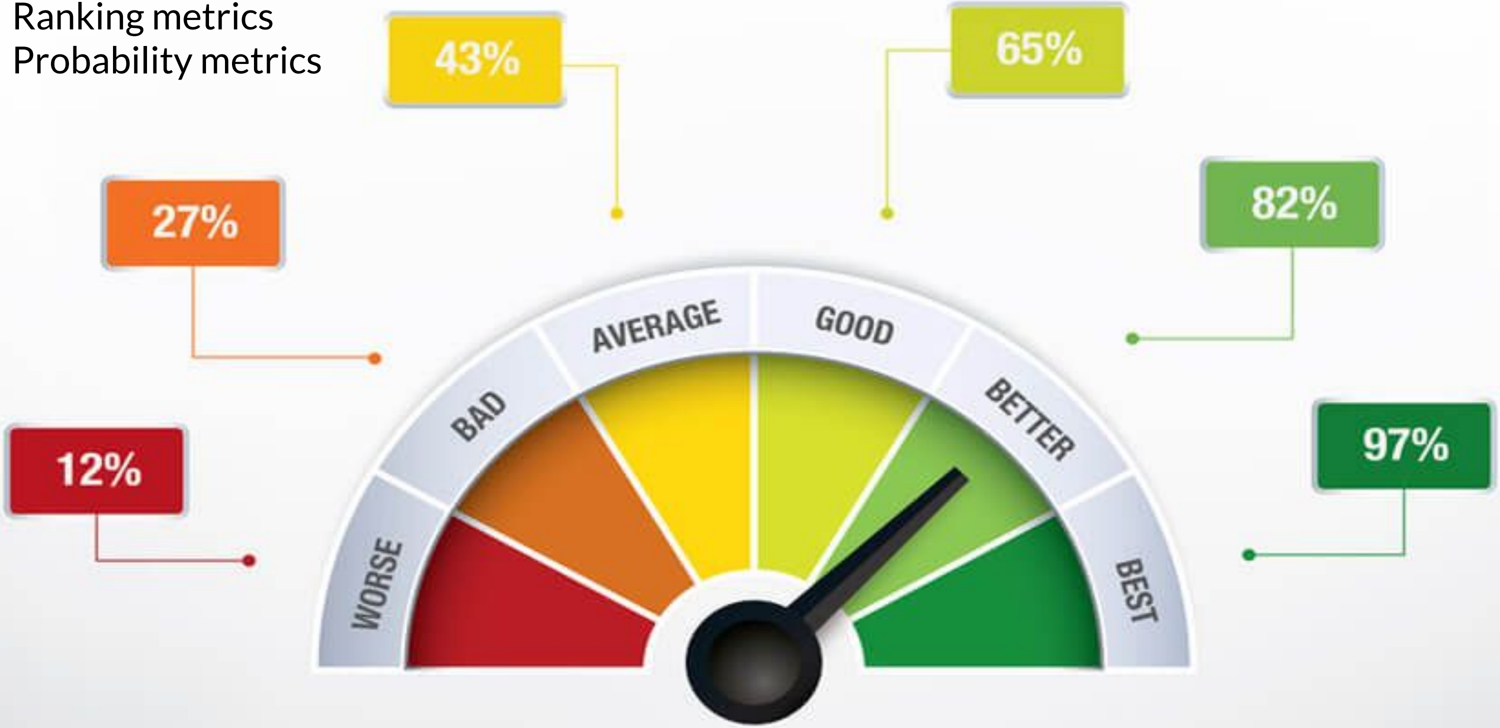
```
from sklearn.tree import plot_tree
```

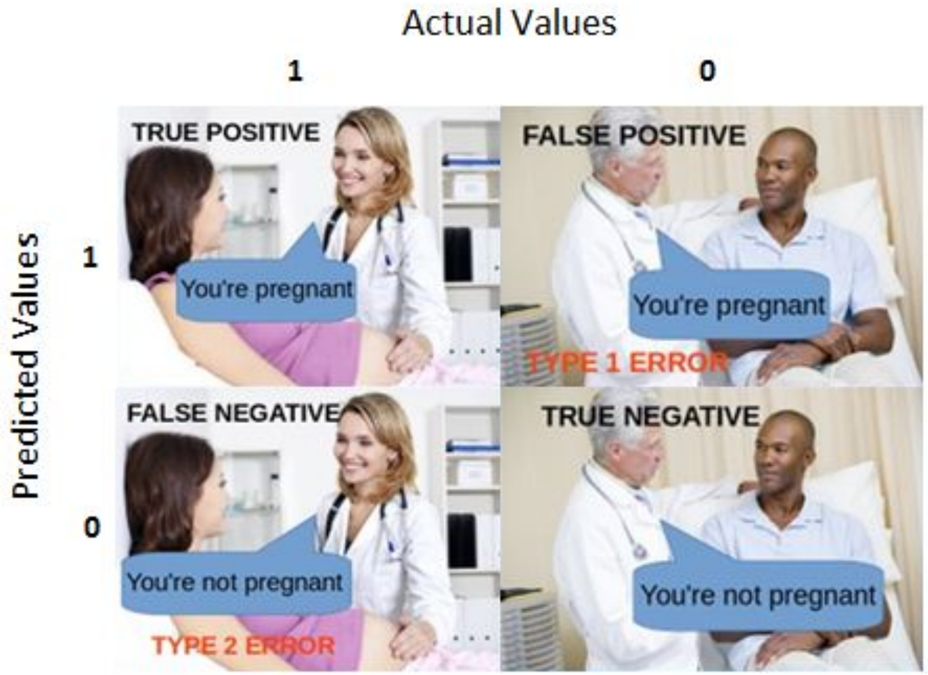


Evaluating Classifiers

A classier is only as good as the metric used to evaluate it

1. Threshold metrics
2. Ranking metrics
3. Probability metrics





$$precision = \frac{TP}{(TP+FP)}$$

$$precision = \frac{TN}{(TN+FN)}$$

<u>Threshold Metrics</u> are those that quantify the classification prediction errors

$$Accuracy = \frac{TP+TN}{TP+TN+FN+FP}$$

$$TPR = \frac{\#\text{true positives}}{\#\text{true positives} + \#\text{false negatives}}$$

Recall

$$TNR = \frac{\#\text{true negatives}}{\#\text{true negatives} + \#\text{false positives}}$$

$$F_1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

```python
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
```

```
accuracy_score(y_test, predict)
0.8432366037156457
```

```
classification_report(y_test,predict)
              precision    recall  f1-score   support

           0       0.86      0.95      0.90      4945
           1       0.77      0.50      0.60      1568

    accuracy                           0.84      6513
   macro avg       0.81      0.73      0.75      6513
weighted avg       0.84      0.84      0.83      6513
```

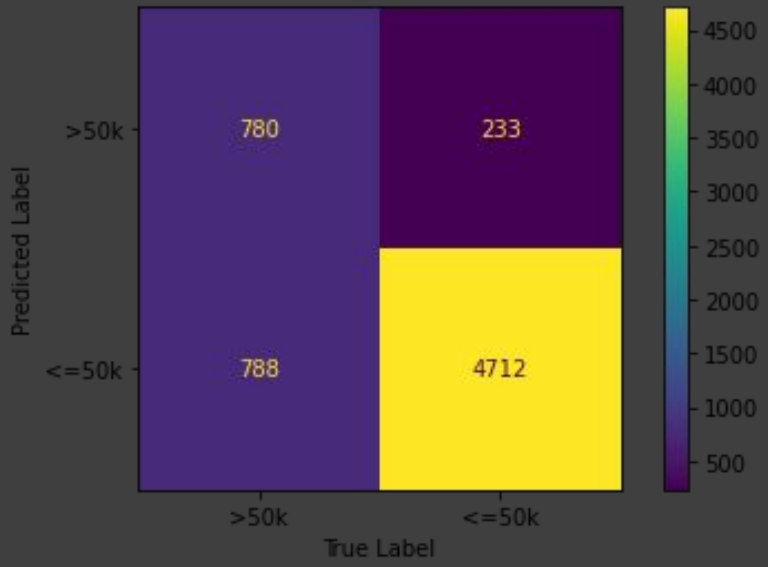
```
confusion_matrix(y_test,predict)
array([[4712,  233],
       [ 788,  780]])
```

```
confusion_matrix(y_test,predict,
                 labels=[1,0])
array([[ 780,  788],
       [ 233, 4712]])
```

```
confusion_matrix(predict,y_test,
                 labels=[1,0])
array([[ 780,  233],
       [ 788, 4712]])
```

```python
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
```

```python
ConfusionMatrixDisplay(confusion_matrix(predict,y_test,labels=[1,0]),
                       display_labels=[">50k","<=50k"]).plot(values_format=".0f",ax=ax)
```



Additional Threshold Metrics

$$G\text{-}mean = \sqrt{\frac{TP}{TP + FN} \times \frac{TN}{TN + FP}}$$

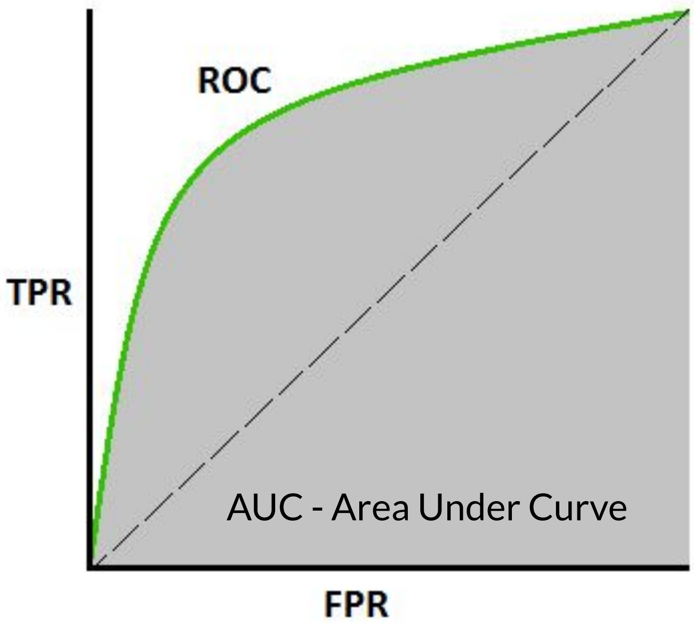
$$Fbeta\text{-}measure = \frac{(1 + \beta^2) \times Precision \times Recall}{\beta^2 \times Precision + Recall}$$

$$\beta == \begin{cases} 0.5, & \text{more weight on precision, less weight on recall} \\ 1, & \text{balance the weight on precision and recall} \\ 2, & \text{less weight on precision, more weight on recall} \end{cases}$$

Ranking Metrics for Classifier Evaluation

- Rank metrics are more concerned with evaluating classifiers based on **how effective they are at separating classes**.
- These metrics require that a classifier predicts a **score** or a **probability of class membership**. Therefore, instead of a simple positive or negative prediction, the score **introduces a level of granularity.**

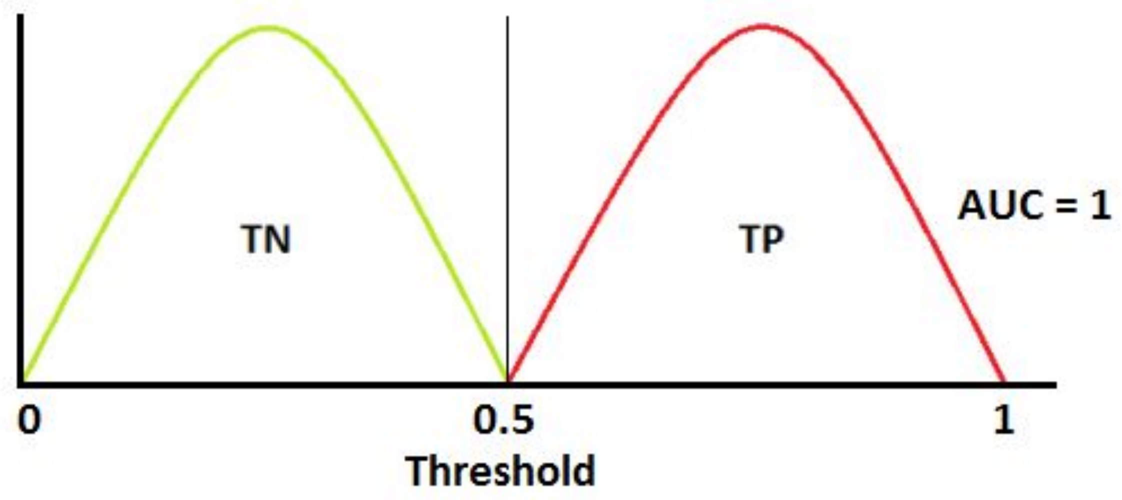
Receiver Operating Characteristic (ROC)

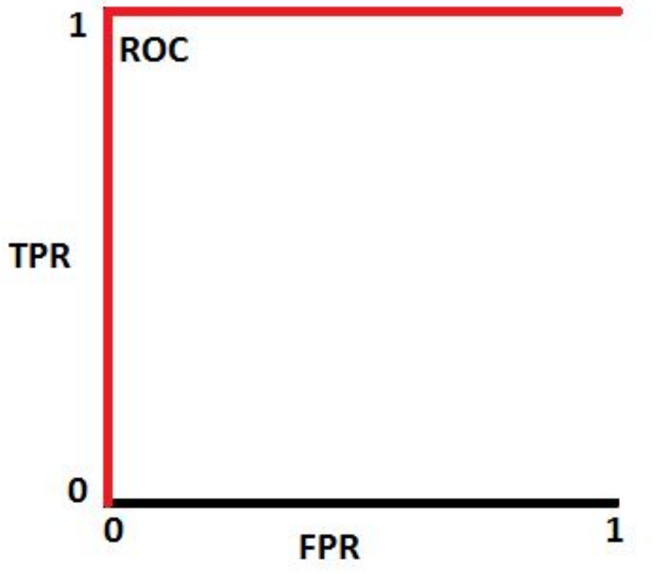


TPR
(measure the impact of True Positive)

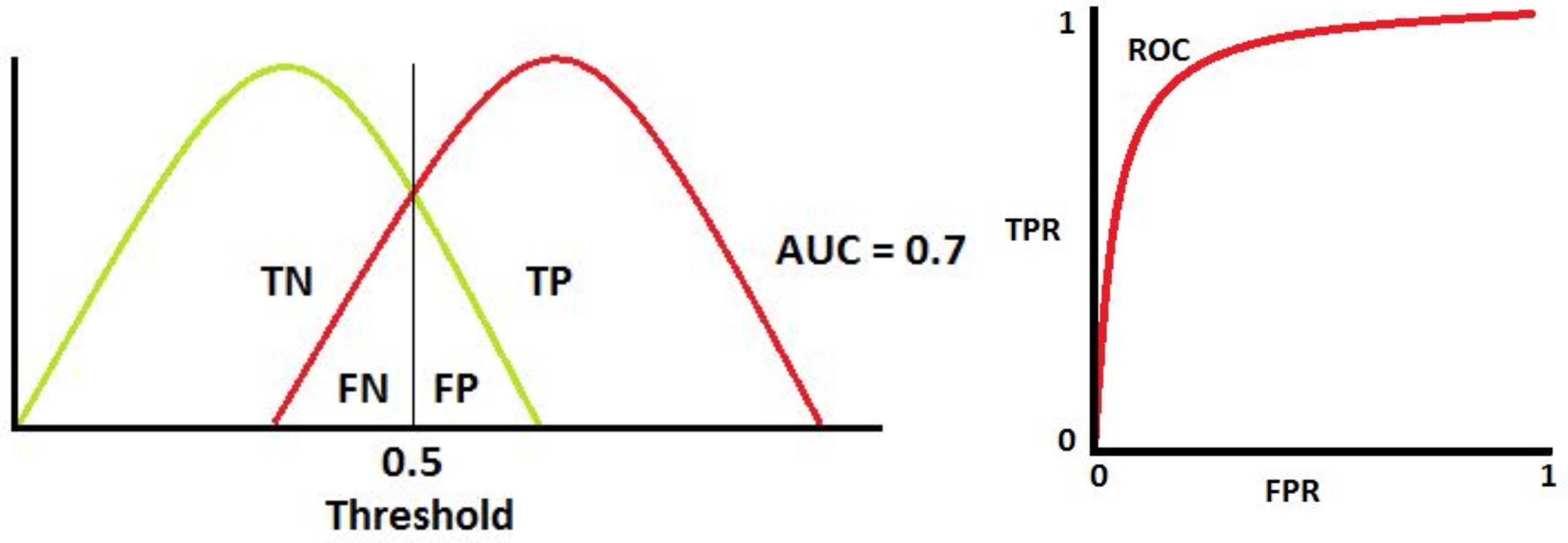
FPR = 1 - TNR
(measure the impact of False Positive)

How to speculate the performance of model?

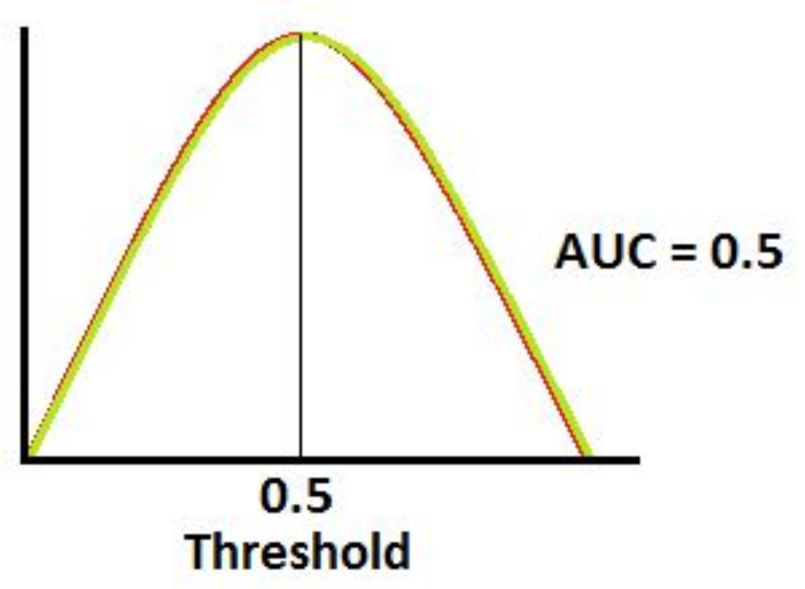


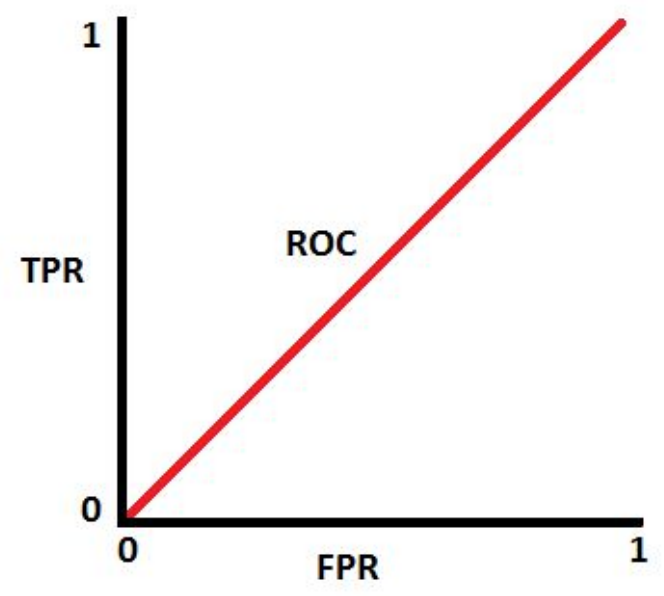


How to speculate the performance of model?

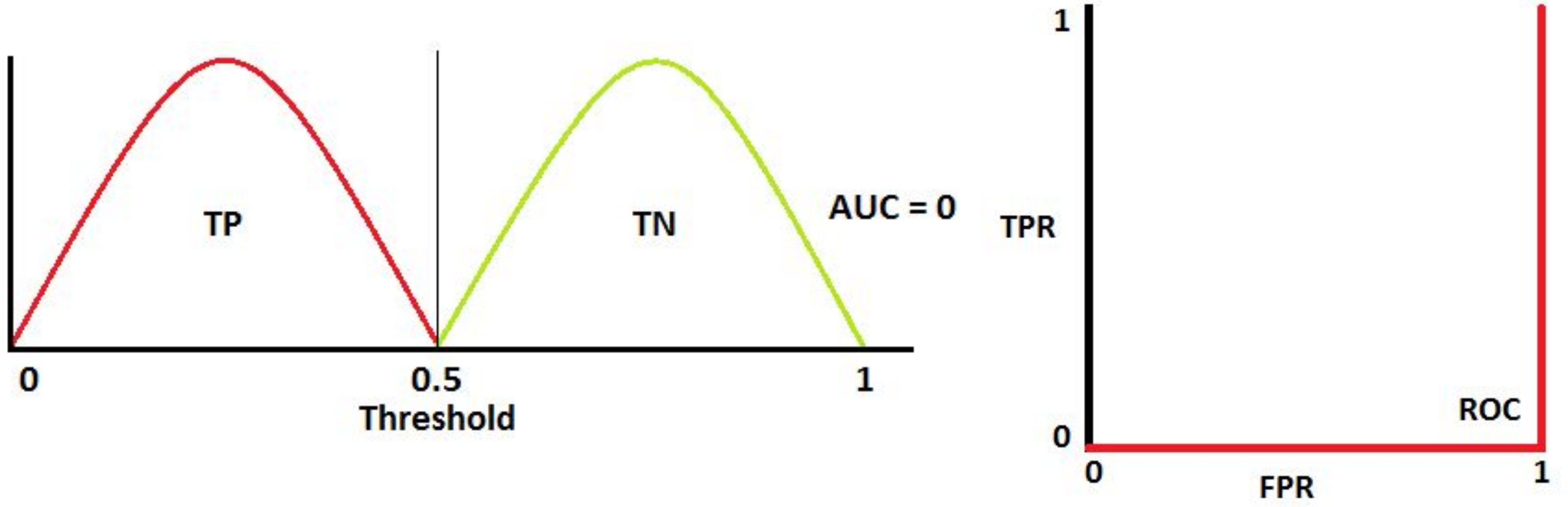


How to speculate the performance of model?





How to speculate the performance of model?



Probabilistic Metrics for Classifier Evaluation

Probabilistic metrics are designed specially **to quantify the uncertainty** in a classier's predictions. These are useful for problems where **we are less interested in incorrect vs. correct class predictions** and more interested in the **uncertainty** the model has in predictions and penalizing those predictions that are wrong but highly condent.

Cross-entropy

$$LogLoss = -((1 - y) \times \log(1 - \hat{y}) + y \times \log \hat{y})$$

The score summarizes the average difference between two probability distributions. A perfect classifier has a log loss of 0.0, with worse values being positive up to infinity.

The score can be generalized to multiple classes by simply adding the terms:

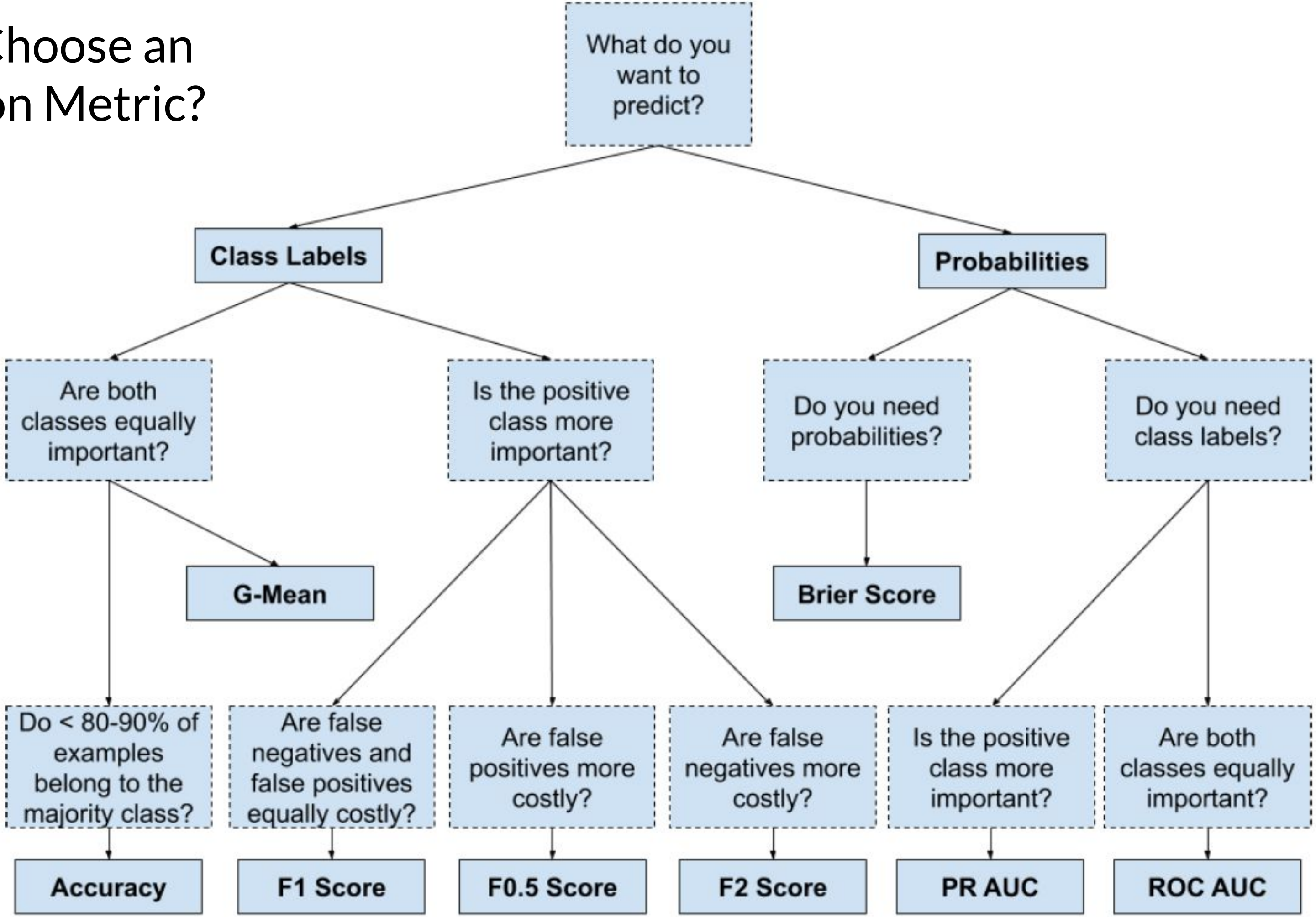
$$LogLoss = -\sum_{c \in C} y_c \times \log \hat{y_c}$$

Brier Score

Another popular score for predicted probabilities is the Brier score. The benefit of the Brier score is that it is **focused on the positive class**, which for imbalanced classification is the minority class. This makes it more preferable than log loss, which is focused on the entire probability distribution.

$$Brier\_Score = \frac{1}{N} \times \sum_{i=1}^{N} (\hat{y}_i - y_i)^2$$

How to Choose an Evaluation Metric?



@jason brownlee

scikit-learn 0.23

Applying Decision Trees

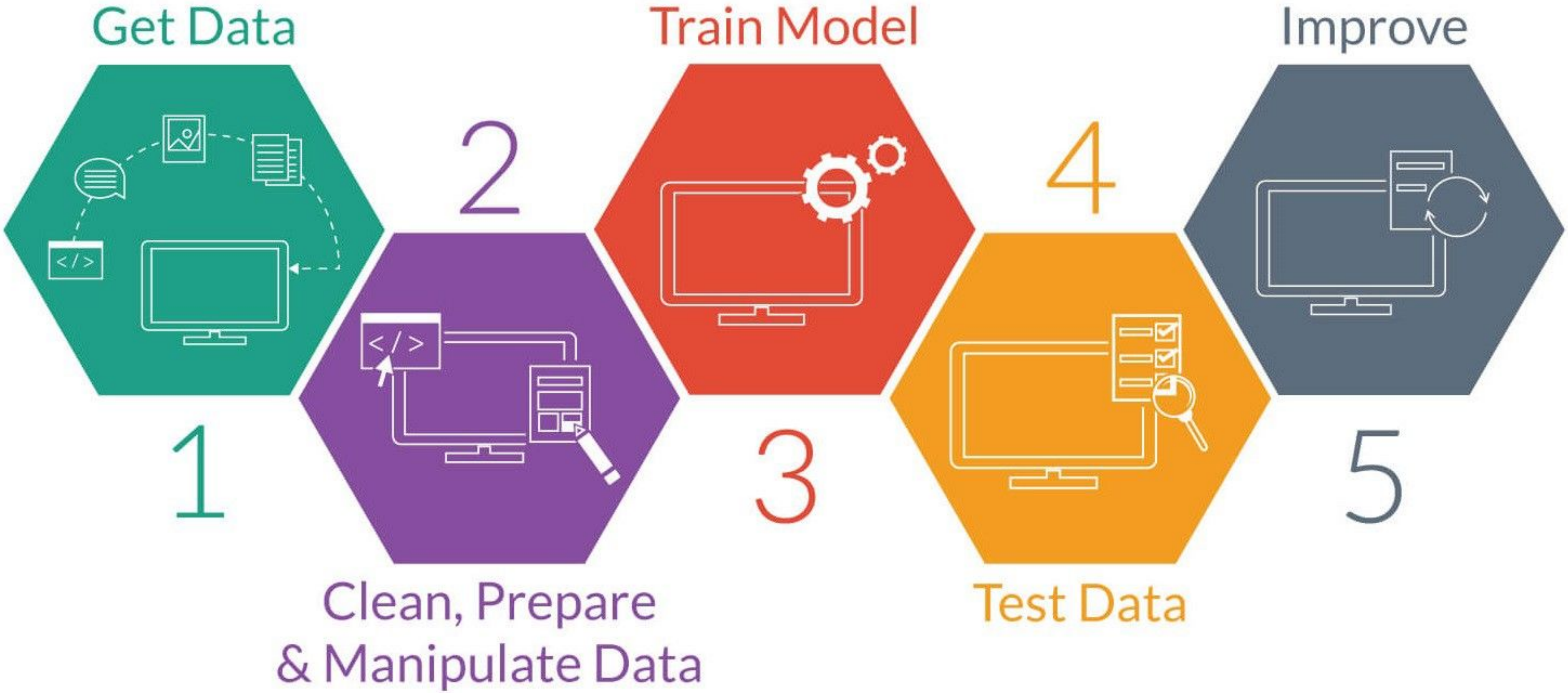
sklearn.tree.DecisionTreeClassifier

class `sklearn.tree.`**`DecisionTreeClassifier`**(**, criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort='deprecated', ccp_alpha=0.0*) [source]

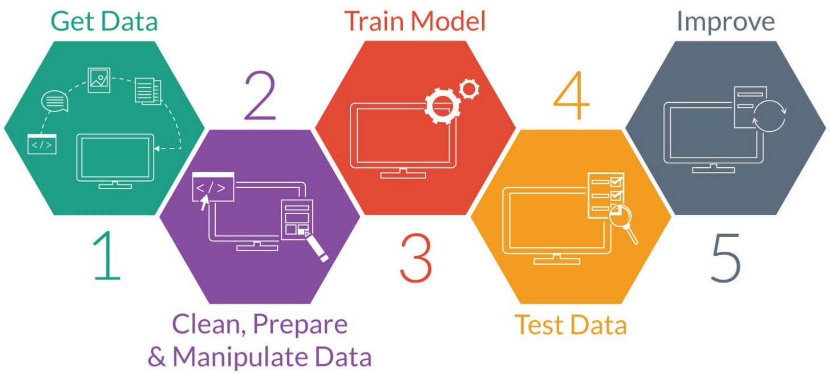
sklearn.tree.DecisionTreeRegressor

class `sklearn.tree.`**`DecisionTreeRegressor`**(**, criterion='mse', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, presort='deprecated', ccp_alpha=0.0*) [source]

A general ML workflow



A general ML workflow

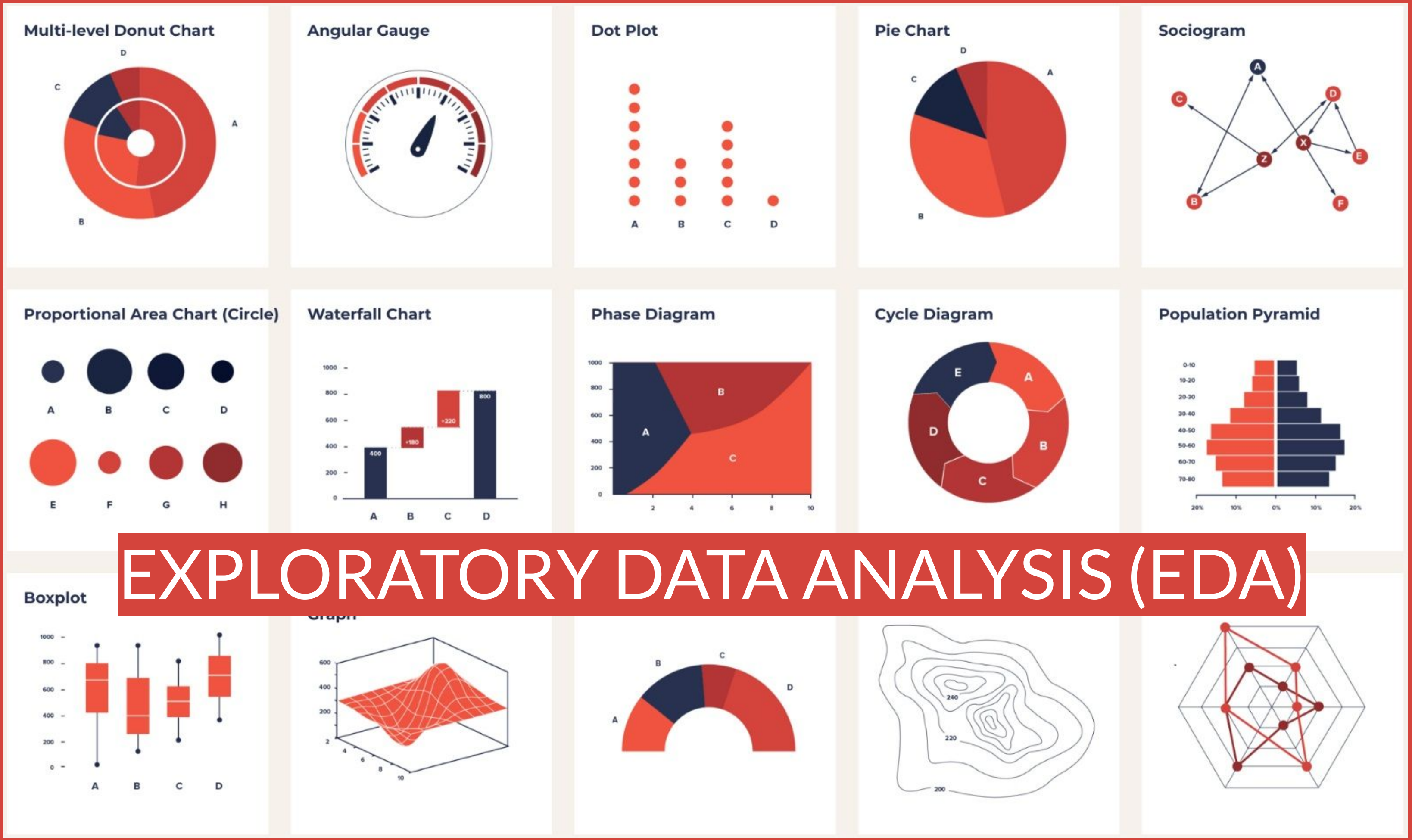


1. Load Libraries
2. Get data, including EDA
3. Clean, prepare and manipulate data (feature engineering)
4. Modeling (train and test)
5. Algorithm Tuning
6. Finalizing the Model

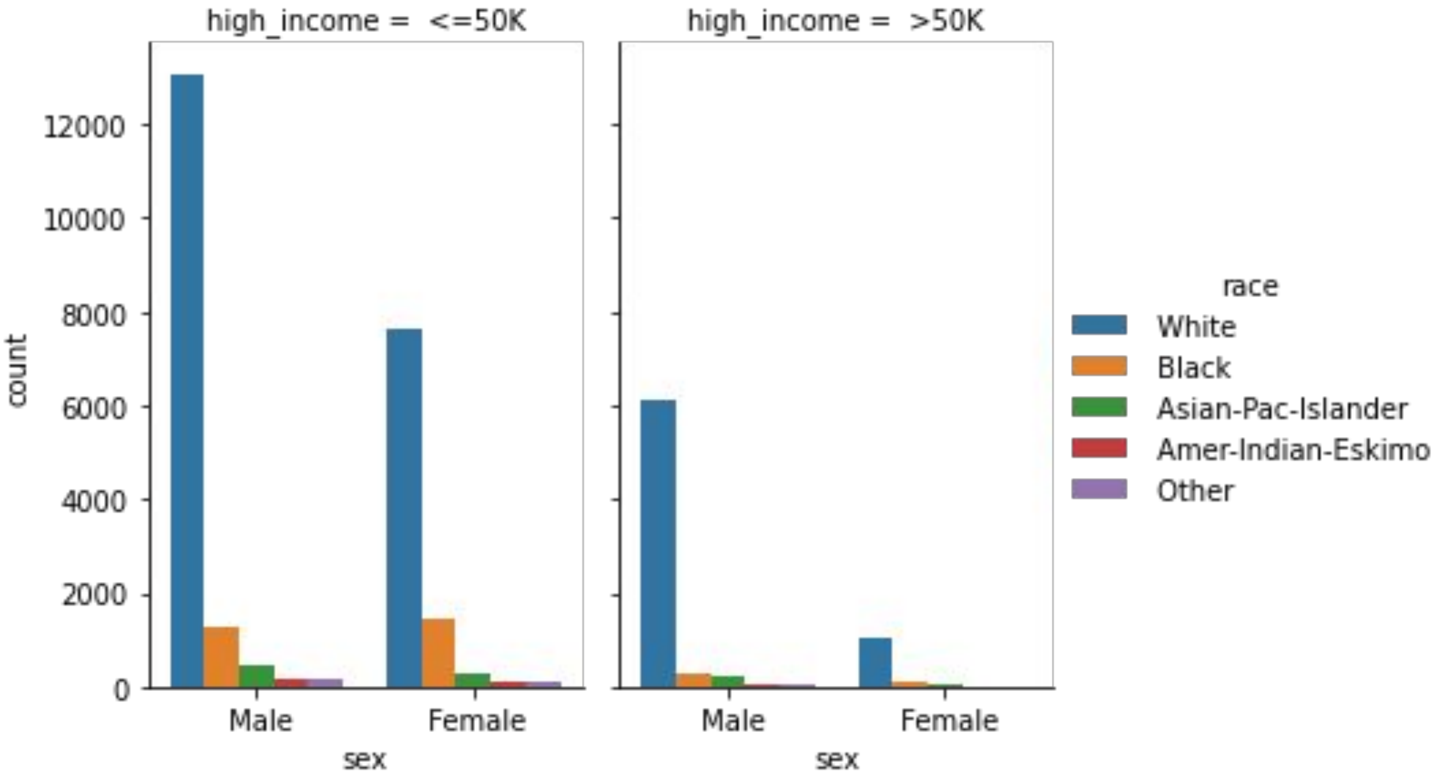
Load Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.neighbors import LocalOutlierFactor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import OneHotEncoder
from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.model_selection import StratifiedKFold
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.metrics import ConfusionMatrixDisplay
from sklearn.tree import plot_tree
import joblib
```

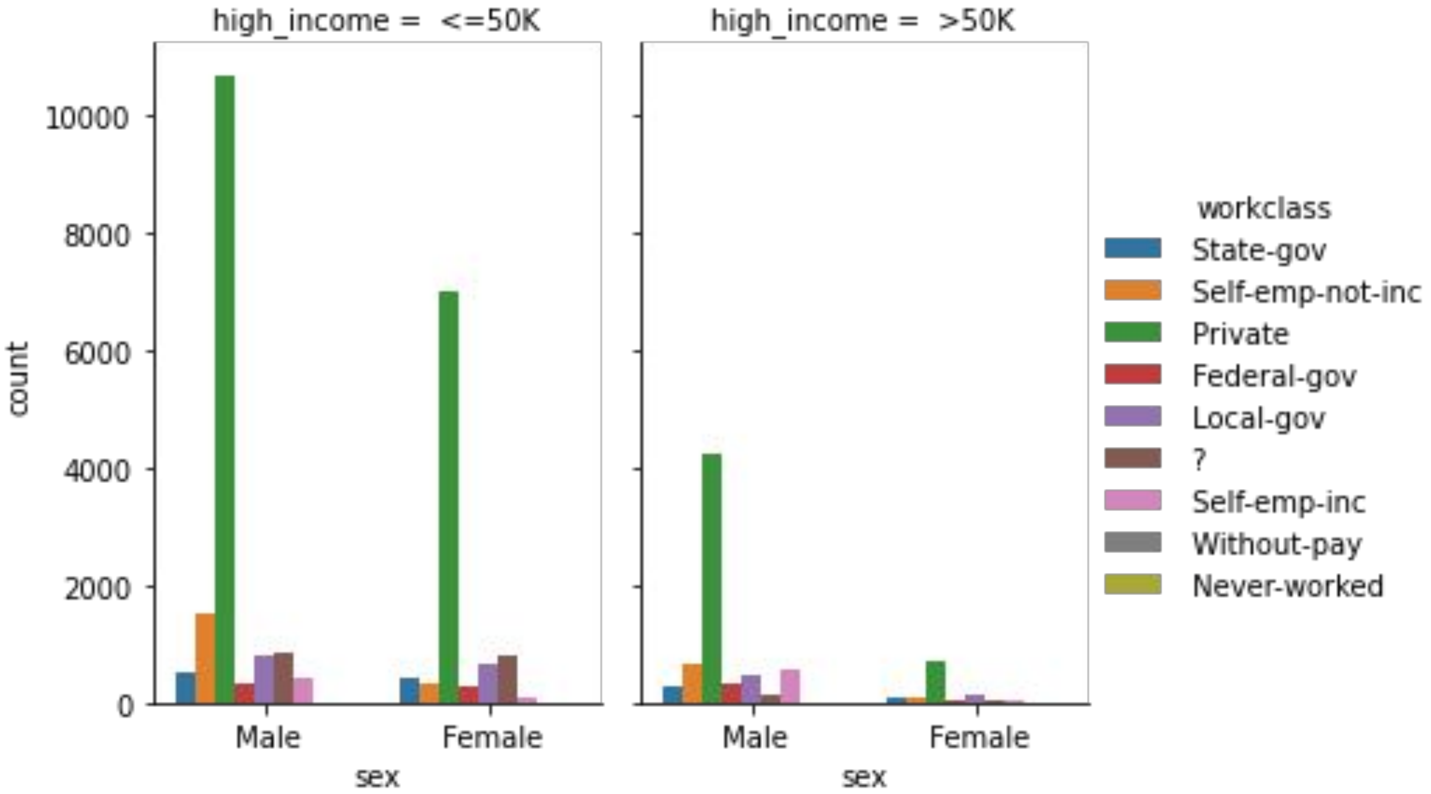
EXPLORATORY DATA ANALYSIS (EDA)

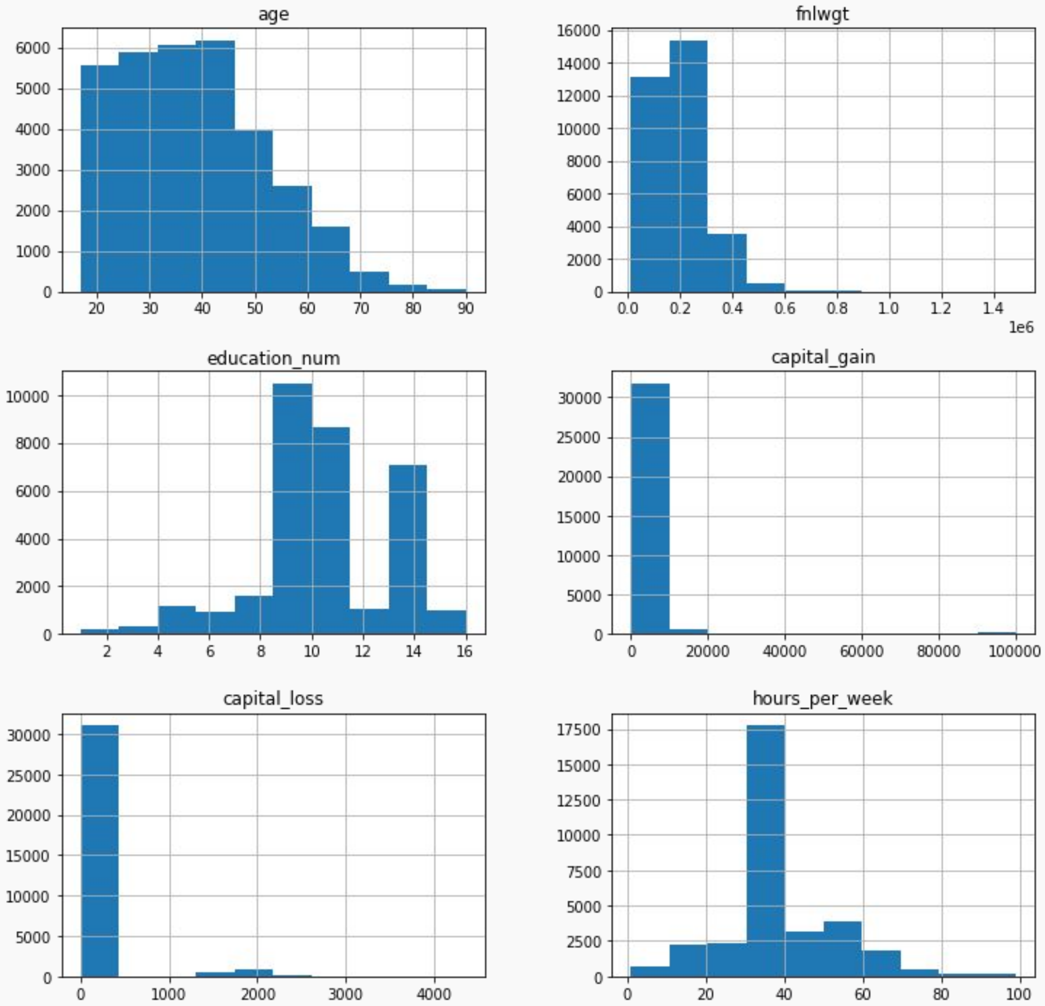


```
g = sns.catplot(x="sex",
                hue="race",
                col="high_income",
                data=income, kind="count",
                height=4, aspect=.7)
```



```
g = sns.catplot(x="sex",
                hue="workclass",
                col="high_income",
                data=income, kind="count",
                height=4, aspect=.7)
```

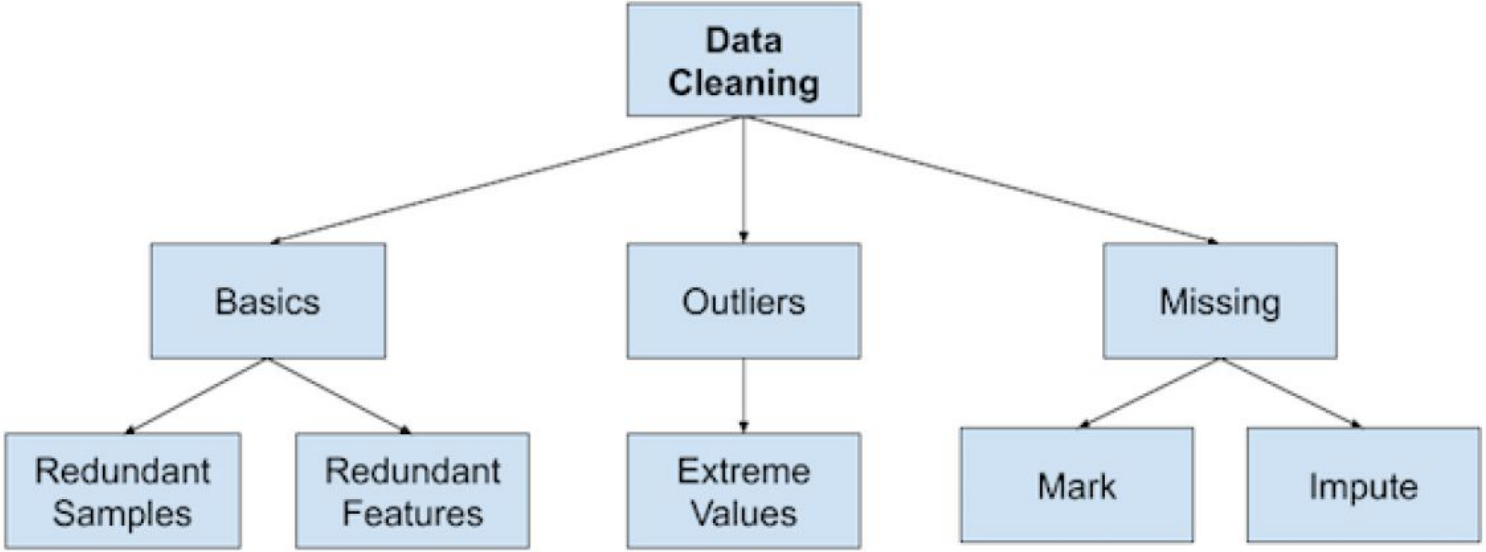





```
fig, ax = plt.subplots(1,1,figsize=(12,12))
income.hist(ax=ax)
plt.show()
```

CLEAN, PREPARE AND MANIPULATE DATA

Overview of Data Cleaning

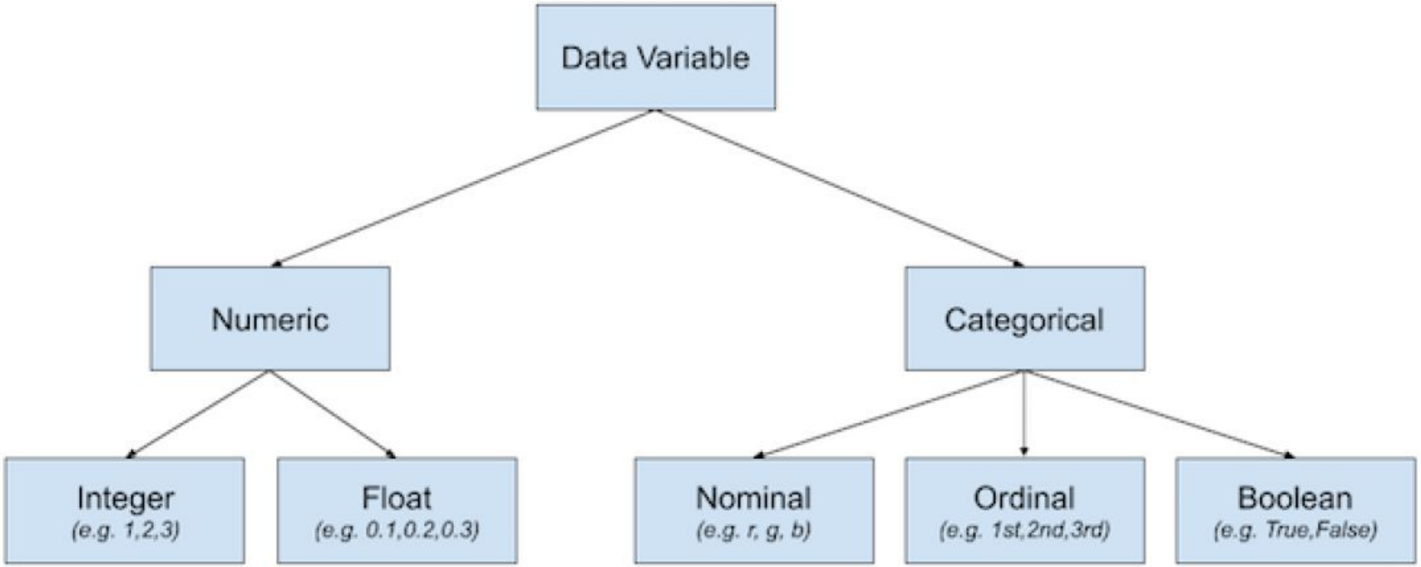


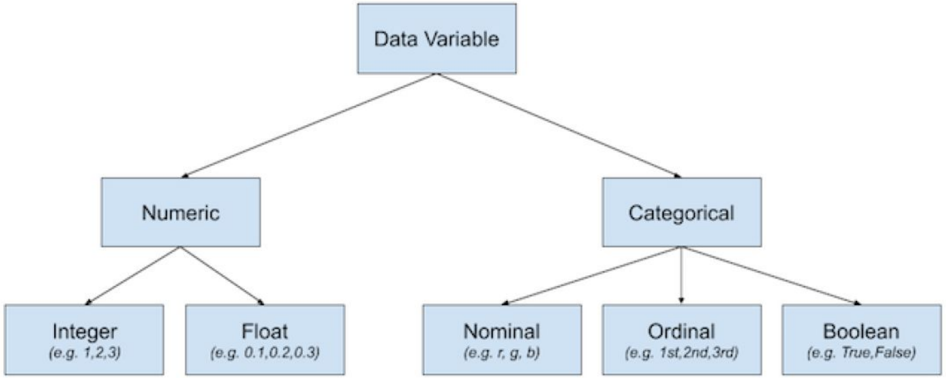

```
df.drop_duplicates()
df.duplicated()
```

```
from sklearn.neighbors import LocalOutlierFactor
from sklearn.ensemble import IsolationForest
```

```
from sklearn.impute import SimpleImputer
from sklearn.impute import KNNImputer
```

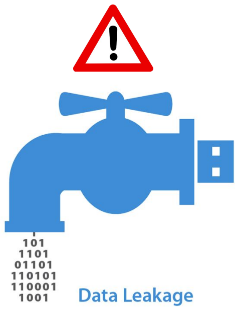
Overview of Data Variable Types

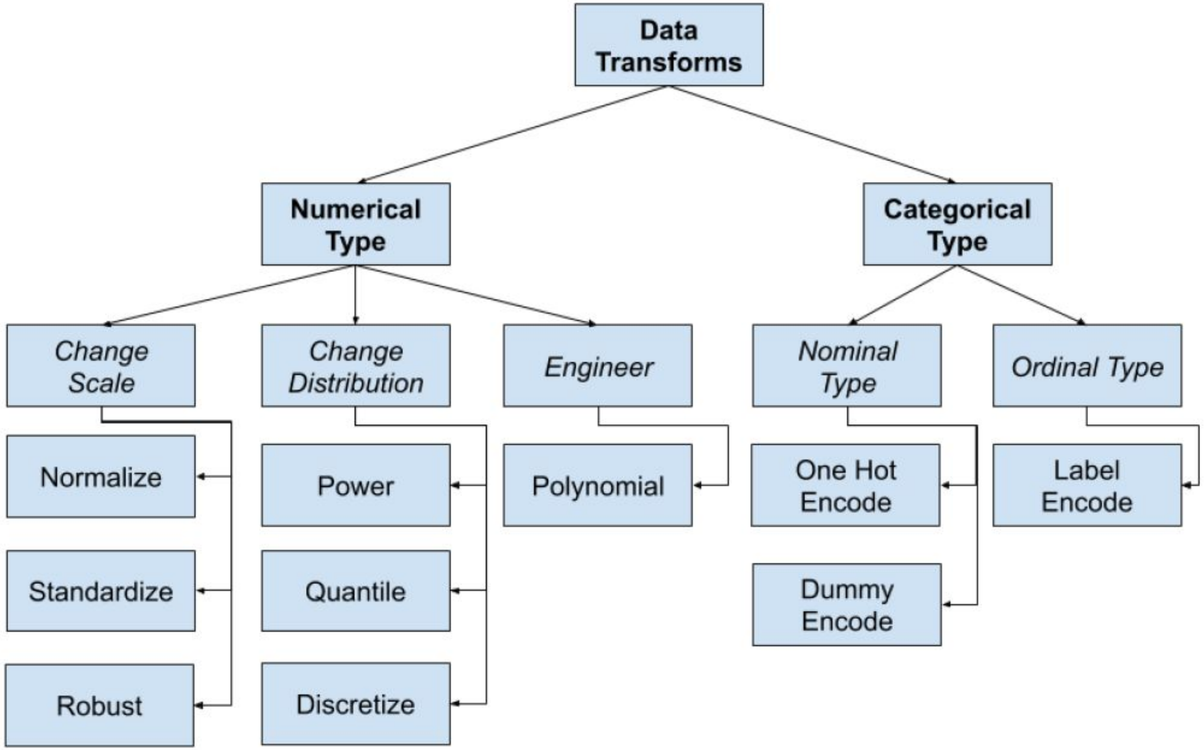




	age	workclass	fnlwgt	education	education_num	marital_status	occupation	relationship	race	sex	high_income
0	39	State-gov	77516	Bachelors	13	Never-married	Adm-clerical	Not-in-family	White	Male	<=50K
1	50	Self-emp-not-inc	83311	Bachelors	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	<=50K
2	38	Private	215646	HS-grad	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	<=50K
3	53	Private	234721	11th	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	<=50K
4	28	Private	338409	Bachelors	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	<=50K

Overview of Data Transforms





Wrong way!!!!

```
# data
x = income.select_dtypes("int64")
y = np.where(data["high_income"] == ' <=50K',0,1)

# standardize the dataset
scaler = MinMaxScaler()
x = scaler.fit_transform(x)

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=1)


# fit the model
model = DecisionTreeClassifier()
model.fit(x_train,y_train)

# evaluate the model
yhat = model.predict(x_test)

accuracy_score(y_test,yhat)
0.7723410789231242
```

Right way!!!!

```
# data
x = income.select_dtypes("int64")
y = np.where(data["high_income"] == ' <=50K',0,1)

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=1)

# standardize the train dataset
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)

# scale the test dataset
x_test = scaler.transform(x_test)

# fit the model
model = DecisionTreeClassifier()
model.fit(x_train,y_train)

# evaluate the model
yhat = model.predict(x_test)

accuracy_score(y_test,yhat)
0.7734670897737742
```

Outlier removal without data leakage

```
# data
x = data.select_dtypes("int64")
y = np.where(data["high_income"] == ' <=50K',0,1)

# split into train and test sets
x_train, x_test, y_train, y_test = train_test_split(x,
                                                    y,
                                                    test_size=0.3,
                                                    random_state=1)

# identify outlier in the training dataset
lof = LocalOutlierFactor()
outlier = lof.fit_predict(x_train)
mask = outlier != -1
# select all rows that are not outliers
x_train, y_train = x_train[mask], y_train[mask]

# standardize the train dataset
scaler = MinMaxScaler()
x_train = scaler.fit_transform(x_train)

# scale the test dataset
x_test = scaler.transform(x_test)

# fit the model
model = DecisionTreeClassifier()
model.fit(x_train,y_train)

# evaluate the model
yhat = model.predict(x_test)

accuracy_score(y_test,yhat)
0.7632306274951377
```

Train and Test

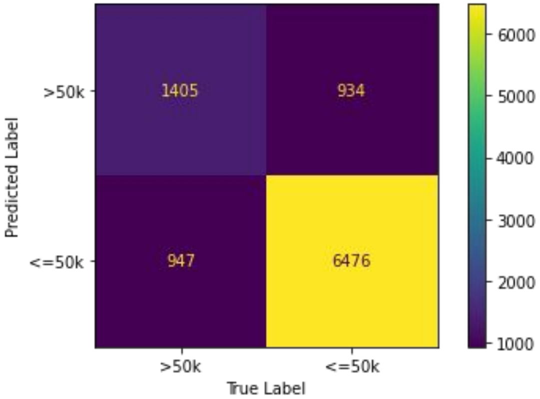
```
# split-out train/validation and test dataset
x_train, x_test, y_train, y_test =
train_test_split(income.drop(labels="high_income",axis=1),
                                                     income["high_income"],
                                                     test_size=0.30,
                                                     random_state=0,
                                                     shuffle=True,
                                                     stratify=income["high_income"])
```

Hyperparameter Tuning

```python
# create a pipeline
pipe = Pipeline([("classifier", DecisionTreeClassifier())])

# create a dictionary with the hyperparameters
search_space = [{"classifier":[DecisionTreeClassifier()],
                 "classifier__criterion": ["gini","entropy"]},
                {"classifier": [KNeighborsClassifier()],
                 "classifier__n_neighbors": [5,7]}]

# create grid search
kfold = StratifiedKFold(n_splits=10,random_state=0,shuffle=True)

# see other scoring
# https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter
grid = GridSearchCV(estimator=pipe,
                    param_grid=search_space,
                    cv=kfold,
                    scoring="accuracy",
                    n_jobs=-1)

# fit grid search
best_model = grid.fit(x_train,y_train)
```

Performance Evaluation

```
0.800610 (0.006070) with: {'classifier': DecisionTreeClassifier(criterion='entropy'), 'classifier__criterion': 'gini'}
0.806520 (0.009790) with: {'classifier': DecisionTreeClassifier(criterion='entropy'), 'classifier__criterion': 'entropy'}
0.781004 (0.006412) with: {'classifier': KNeighborsClassifier(), 'classifier__n_neighbors': 5}
0.790619 (0.006123) with: {'classifier': KNeighborsClassifier(), 'classifier__n_neighbors': 7}
```



	precision	recall	f1-score	support
0	0.87	0.87	0.87	7410
1	0.60	0.60	0.60	2352
accuracy			0.81	9762
macro avg	0.74	0.74	0.74	9762
weighted avg	0.81	0.81	0.81	9762

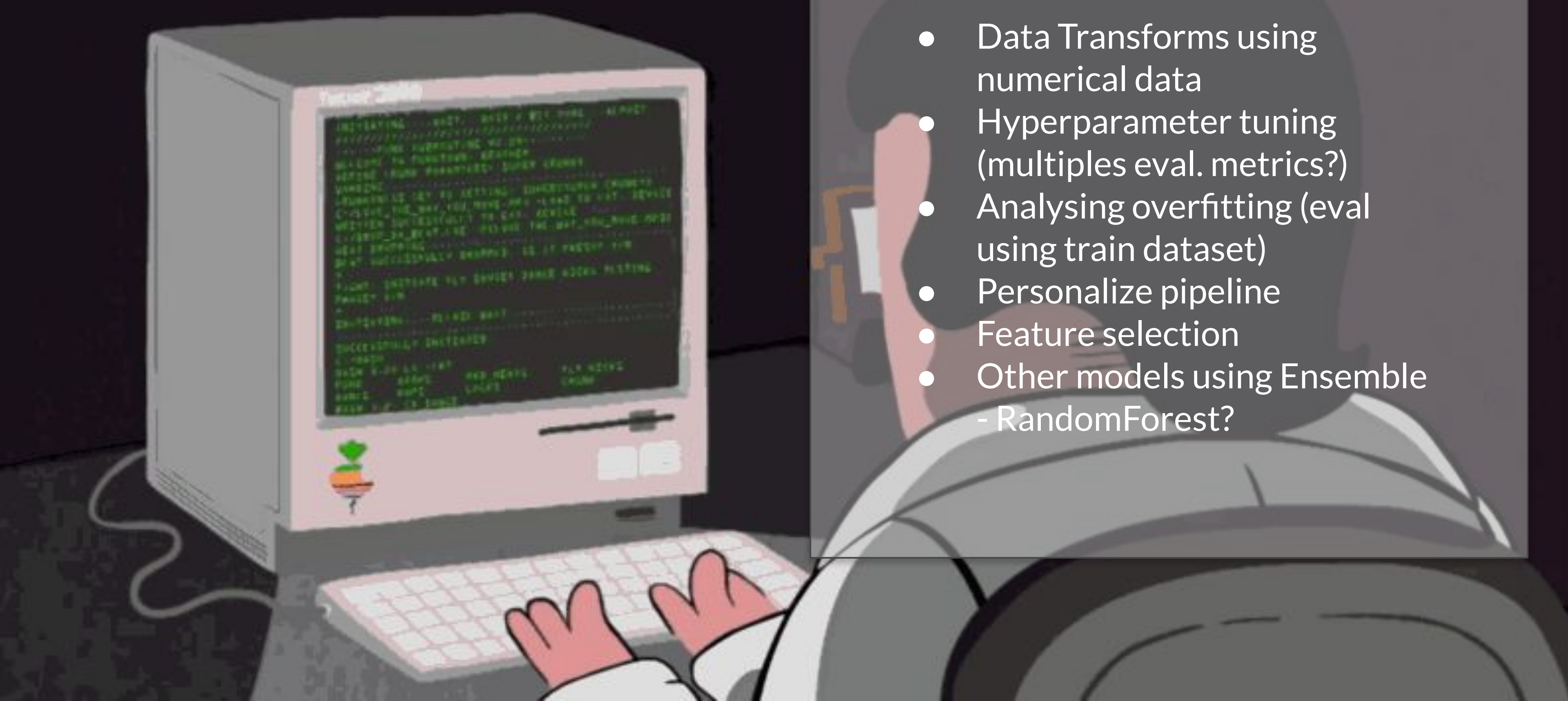
Knowing when to use decision trees

The main advantages of using decision trees is that they're:

- Easy to interpret
- Relatively fast to fit and make predictions
- Able to handle multiple types of data
- Able to pick up nonlinearities in data, and usually fairly accurate

The main disadvantage of using decision trees is their **tendency to overfit.**

Lesson #04 - Decision Trees.ipynb



- Data Transforms using numerical data
- Hyperparameter tuning (multiples eval. metrics?)
- Analysing overfitting (eval using train dataset)
- Personalize pipeline
- Feature selection
- Other models using Ensemble - RandomForest?