
# Advanced Classification - Part 1

*One should look for what is and not what he thinks should be. (Albert Einstein)*

# Welcome!



DATASOCIETY: © 2022

# Welcome: Icebreaker!

In the chat, discuss the following questions:

**1. Why are you taking this course and what would you like to learn?**

**2. How do you use classification for your job? Think about specific tasks and goals.**

**3. What do you know about random forest algorithm and how it is used?**



DATASOCIETY: © 2022

# Best practices for virtual classes

1. Find a quiet place, free of as many distractions as possible. Headphones are recommended.
2. Stay on mute unless you are speaking.
3. Remove or silence alerts from cell phones, e-mail pop-ups, etc.
4. Participate in activities and ask questions. This will remain interactive!
5. Give your honest feedback so we can troubleshoot problems and improve the course.


DATASOCIETY: © 2022

# What level of proficiency do I need?

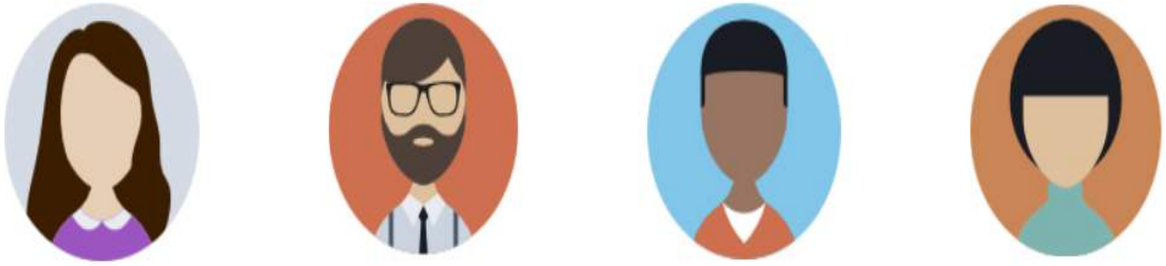
- To use programming as a tool in your professional toolkit, you don't need to be a computer scientist or have a similar level of knowledge as one
- The level of proficiency will depend on
  - **the problems** you are trying to solve on daily basis
  - **the subject matter area** you are in
  - **the level of sophistication of your solutions** that you would like to implement
- Most of the time, people who are subject matter experts who also use various programming tools and languages are known as *data analysts* or *data scientists*

***What are the problems you are trying to solve? What is your area of expertise? What level of complexity would you like your programmatic solution to have?***


DATASOCIETY: © 2022

# A data scientist can

1. **Pose** the right question
2. **Wrangle** the data (gather, clean, and sample data to get a suitable data set)
3. **Manage** the data for easy access by the organization
4. **Explore** the data to generate a hypothesis
5. **Make predictions** using statistical methods such as regression and classification
6. **Communicate** the results using visualizations, presentations, and products



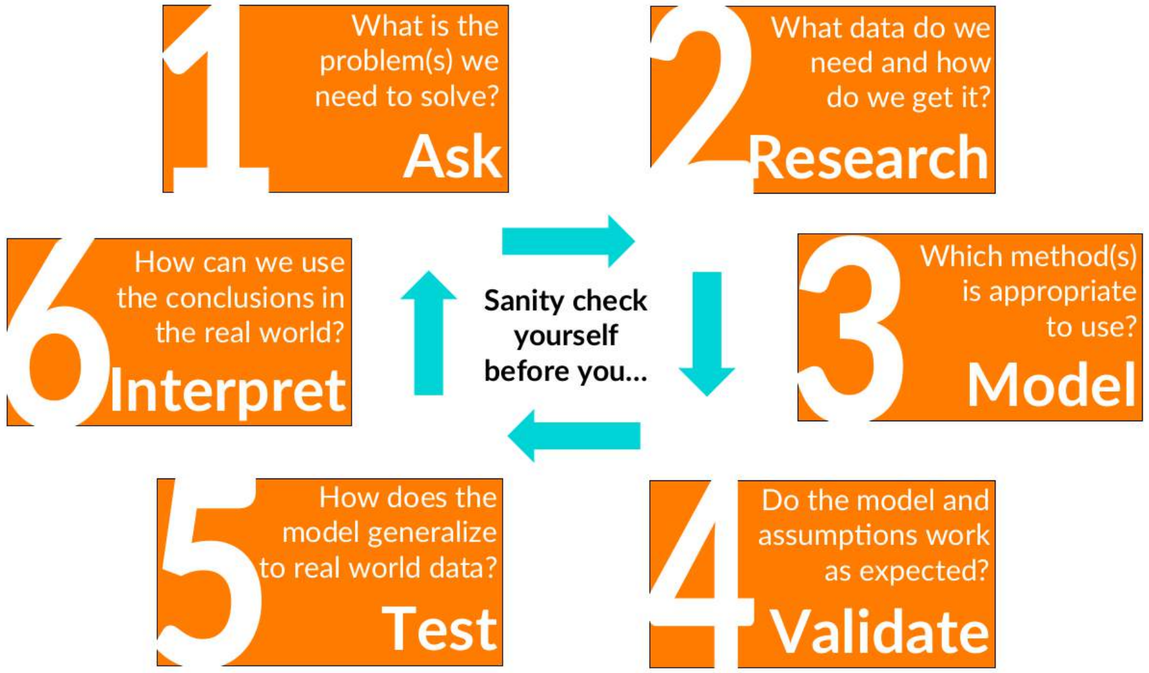
# A data scientist needs to be able to

Use programming languages and tools to

1. **Wrangle** the data (gather, clean, and sample data to get a suitable dataset)
2. **Manage** the data for easy access by the organization
3. **Explore** the data to generate a hypothesis
4. **Make predictions** using statistical methods such as regression and classification

*Stemming from the list above, the programming skills should cover knowing a programming language (or two, or three, or ...) to a degree that allows you to perform these operations!*

# Data science control cycle (DSCC)




- **There is a protocol or standard for working with data that most data scientists follow**
- The cycle involves everything from asking the right questions and being knowledgeable about the data you're studying, to optimizing your model's performance
- Which part of the cycle do you think takes up the most time?


DATASOCIETY: © 2022

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Introduce random forest and discuss use cases | |
| Summarize the concepts associated with random forest and bagging | |
| Load dataset and implement random forest | |

*For each module, we'll start out with our objectives for the session, so you know what to expect!* - Let's get started!


DATASOCIETY: © 2022

# Loading packages

- Let’s load the packages we will be using
- These packages are used for classification using random forests, boosting and other tools

```
import os
import pickle
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from pathlib import Path
from textwrap import wrap
from sklearn.model_selection import train_test_split
from sklearn import metrics

# Random forest and boosting packages
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

# Random forest

- What is Random forest?
  - Ensemble method used for **classification and regression tasks**
  - Supervised learning algorithm which builds **multiple decision trees** and aggregates the result
  - Uses a technique called Bootstrap Aggregation, commonly known as **Bagging**
  - Limits overfitting and bias error

# Random forest: use cases

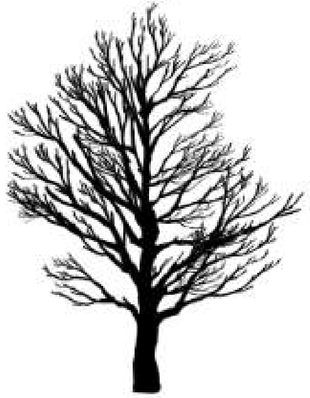
- The random forest algorithm is used in a **multitude of industries** such as banking, medicine, e-commerce, etc.
- Some examples are:
  - Fraud detection
  - Identifying a disease by analyzing patient's medical history
  - Predicting the behavior of the stock market
  - Understanding whether a customer will buy a product or not


DATASOCIETY: © 2022

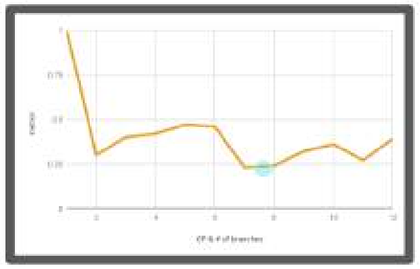
# Decision trees process

- You probably already know how a decision tree works!
- Decision trees **predict** the target value of an item by **mapping observations about the item**
- Below is a brief overview of decision trees as they are related to random forests

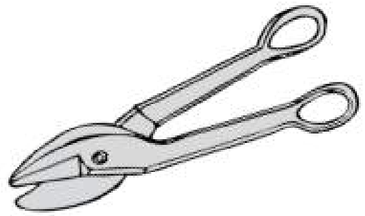
**Step 1:**
Grow tree on training data



**Step 2:**
Examine Model output



**Step 3:**
Prune Tree



**Step 4:**
Check performance on test data

| | Act + | Act - | |
|---|---|---|---|
| Pred + | | | |
| Pred - | | | |
| | | | |


DATASOCIETY: © 2022

# Decision trees

- Decision trees are **great** when used for:
  - Classification and regression
  - Handling numerical and categorical data
  - Handling data with missing values
  - Handling data with nonlinear relationships between parameters
- Decision trees are **not very good** at:
  - Generalization: they are known for overfitting
  - Robustness: small variations in data can result in a different tree
  - Mitigating bias: if some classes dominate, trees may be unbalanced and biased


DATASOCIETY: © 2022

# Why should we use random forest?

- Reduction in overfitting
- Higher predictive accuracy
- Efficient with large datasets
- Why should we use **decision trees** instead?
- Intuitive and easily interpretable results
- Less computationally expensive algorithm


DATASOCIETY: © 2022

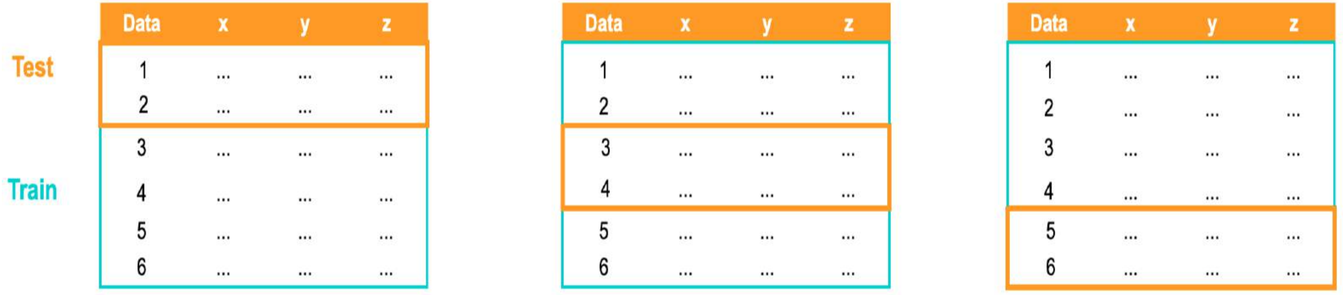
# Module completion checklist

| Objective | Complete |
| --- | --- |
| Introduce random forest and discuss use cases | ✔ |
| Summarize the concepts associated with random forest and bagging | |
| Load dataset and implement random forest | |


DATASOCIETY: © 2022

# Why is random forest popular?

- It uses many decision trees on different subsections of the dataset and averages out the results to improve the predictive accuracy and control overfitting
- **“Bagging”** is an ensemble method that adopts the bootstrap sampling technique, which creates new datasets by using random sampling with replacement
- The Out of Bag error rate for the forest of trees is used as a metric to assess the algorithms’ performance
- It uses a built-in form of multi-fold cross-validation method





DATASOCIETY: © 2022

# Bagging observations


- **Bootstrap aggregation** is the process that makes up bagging
- **Bootstrap sampling technique** creates new datasets by random sampling with replacement
- **Bagging** within **CART** lets you choose how many trees, i.e., how many bootstrapped sampled training sets to create

# Random forest: bagging

*Bagging of observations in itself is pretty cool!*


*But that's not all it does...*


DATASOCIETY: © 2022

# Bagging is not enough

- With using just bagging, random forests can have a lot of structural similarities with decision trees and, in turn, have a high **bias** (a known drawback of tree algorithms!)

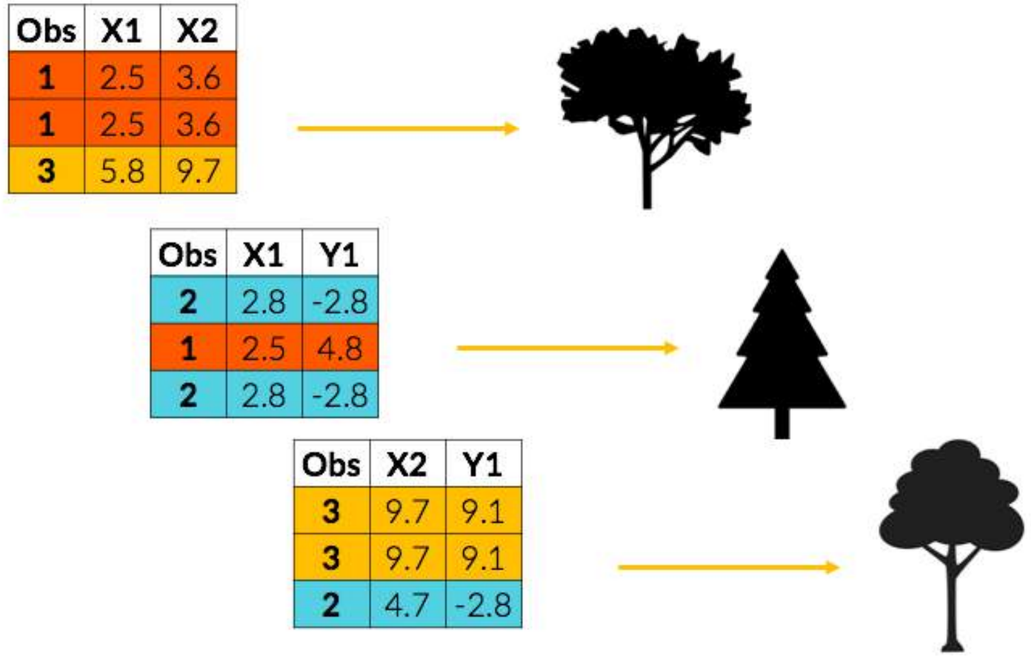
| Obs | X1 | X2 | Y1 | Y2 |
|---|---|---|---|---|
| 1 | 2.5 | 3.6 | 4.8 | 3.7 |
| 2 | 2.8 | 4.7 | -2.8 | 7.1 |
| 3 | 5.8 | 9.7 | 9.1 | 13 |

| Obs | X1 | X2 | Y1 | Y2 |
|---|---|---|---|---|
| 2 | 2.8 | 4.7 | -2.8 | 7.1 |
| 1 | 2.5 | 3.6 | 4.8 | 3.7 |
| 3 | 5.8 | 9.7 | 9.1 | 13 |

| Obs | X1 | X2 | Y1 | Y2 |
|---|---|---|---|---|
| 3 | 5.8 | 9.7 | 9.1 | 13 |
| 1 | 2.5 | 3.6 | 4.8 | 3.7 |
| 2 | 2.8 | 4.7 | -2.8 | 7.1 |

# Sample predictors as well!

- The true power of **random forests** vs **CART** is the limitation of predictors
- For each tree, both samples of observations and **random samples of features** are used instead of using the entire set of features every time




- The resulting model becomes **unbiased** due to a good tree variety, where no variable dominates!

# Building the forest

The two main parameters we need to set to build a **random forest** are:

1. Number of trees
2. Number of features per tree

We can stick with these rules:

1. **N of trees** - the more the better, but a good rule of thumb is $n \approx 100$, where $n = number\ of\ trees$
2. **N of features per tree** - the rule of thumb here is $m = \sqrt{p}$, where $p = number\ of\ predictors$

# Random forest methodology

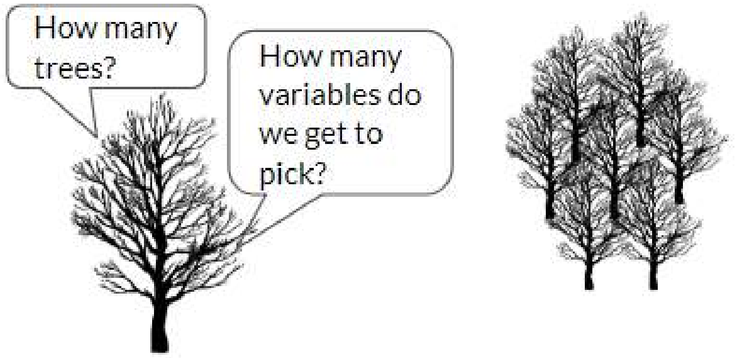
**Step 1:**
Set *n* & *m* for forest

**Step 2:**
Build forest on training data

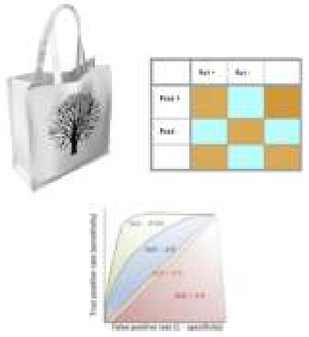
**Step 3:**
Check performance

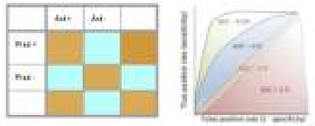
**Step 4:**
Apply to test data & evaluate

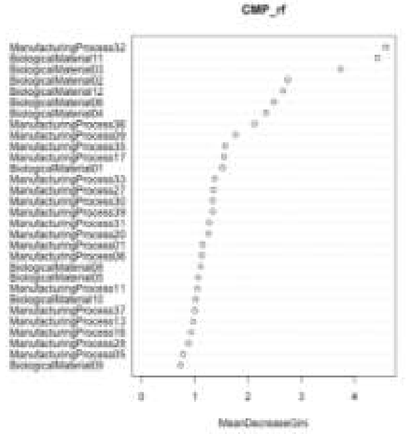
**Step 5:**
Use variable importance as needed





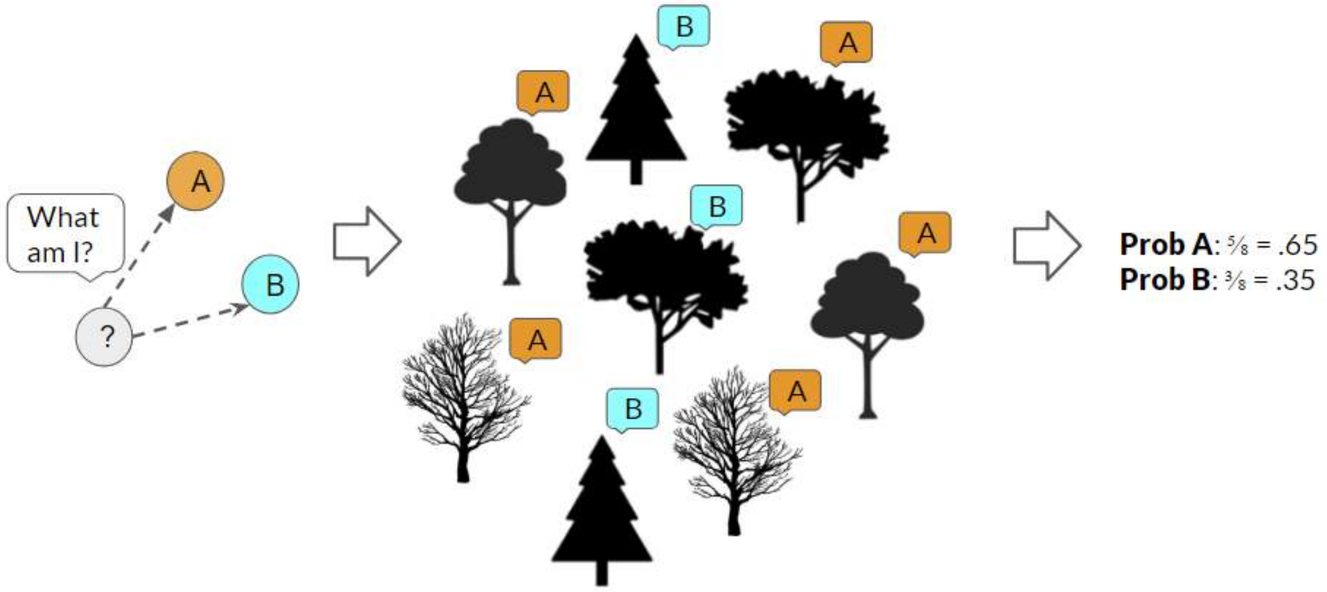






DATASOCIETY: © 2022

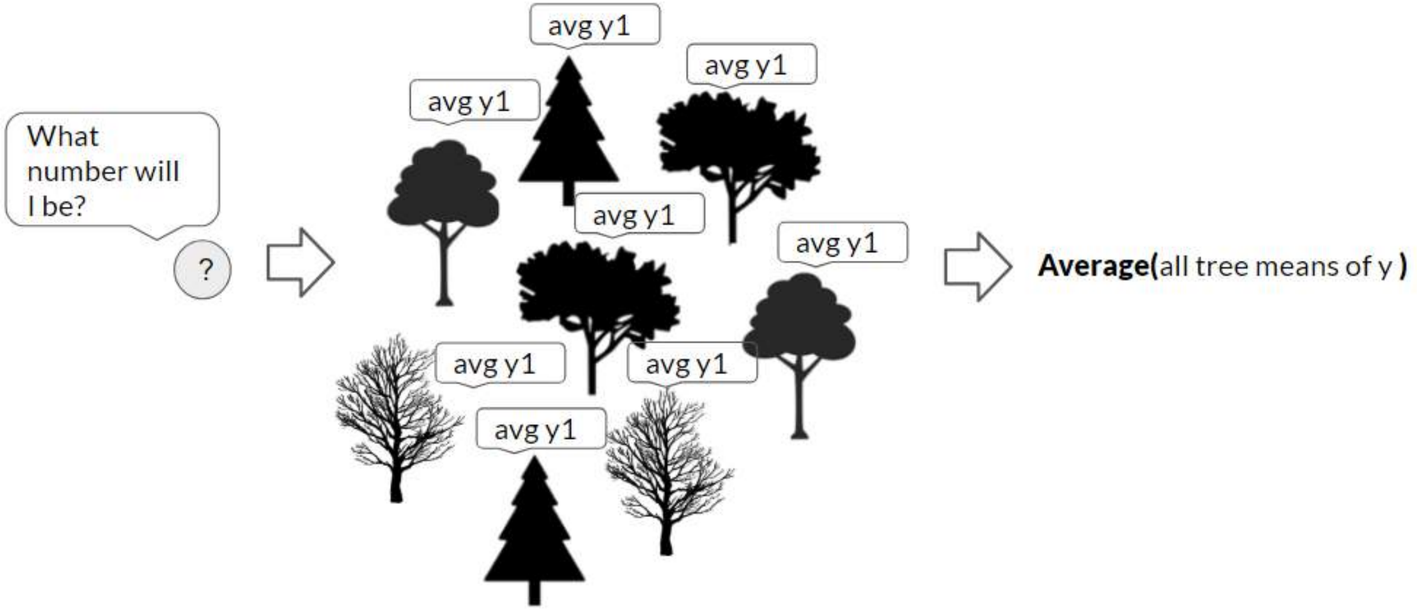
# Random forest classification





DATASOCIETY: © 2022

# Handling regression with random forest




*For this module, we will focus on classification*


DATASOCIETY: © 2022

# Knowledge Check 1

# Module completion checklist

| Objective | Complete |
| --- | --- |
| Introduce random forest and discuss use cases | ✔ |
| Summarize the concepts associated with random forest and bagging | ✔ |
| Load dataset and implement random forest | |


DATASOCIETY: © 2022

# Datasets for today

- **We will be using two datasets in class today, one for in-class practice and the other for self guided-exercise work**

- **A dataset in class to learn the concepts**
  - Costa Rica household poverty data by the Inter-American Development Bank

- **A dataset for our in-class exercises**
  - Bank marketing dataset
    - Related to direct marketing campaigns of a Portuguese banking institution
    - The target variable is whether the user subscribed to the bank term deposit (‘yes’) or not (‘no’)

# Costa Rican poverty: case study

- We will be diving into a case study from the **Inter-American Development Bank (IDB)**
- The **IDB** conducted a competition amongst data scientists on ***Kaggle.com***
- Many countries face this same problem of inaccurately assessing social needs
- The following case study on Costa Rican poverty levels is a good example of how we can use data science within social sciences


DATASOCIETY: © 2022

# Costa Rican poverty: backstory

## Costa Rican poverty level prediction

As stated by the **IDB**:

- Social programs have a hard time making sure the right people are given enough aid
- It's especially tricky when a program focuses on the poorest segment of the population
- The world's poorest typically can't provide the necessary income and expense records to prove that they qualify

# Costa Rican poverty: backstory

- **Proxy Means Test (or PMT)** is an algorithm used to verify income qualification
- PMT uses a model that considers a family's observable household attributes (e.g. building materials) or assets to classify them and predict their level of need
- While this is an improvement, accuracy remains a problem as the region's population grows and poverty declines

# Costa Rican poverty: proposed solution

- To improve on PMT, the IDB built a competition for Kaggle participants to use methods beyond traditional econometrics

- The given dataset contains Costa Rican household characteristics with a target of four categories:
  - extreme poverty
  - moderate poverty
  - vulnerable households
  - non-vulnerable households

# Costa Rican poverty: proposed solution

The goal is to develop an algorithm to predict these poverty levels, that can eventually be used not only on the Costa Rican population, but on other countries facing the same problem

**We will work with the Costa Rican dataset and see what we can develop. We will:**

- Clean the dataset
- Wrangle the data and create classification models to predict poverty level

# Predicting poverty

**Ultimate goal**

- Understand the patterns and groups within the dataset
- Predict poverty levels of Costa Rican households and build a model that is reproducible for other countries

# Data cleaning steps

- **Today, we will first clean the Costa Rican dataset**
- The steps to get to this cleaned dataset are:
  - Remove household ID and individual ID
  - Remove variables with over 50% NAs
  - Transform target variable to binary
  - Remove highly correlated variables

# Directory settings

- In order to maximize the efficiency of your workflow, you should encode your directory structure into `variables`
- We will use the `pathlib` library
- Let the `main_dir` be the variable corresponding to your `course` folder
- Let `data_dir` be the variable corresponding to your `data` folder

```
# Set 'main_dir' to location of the project folder
home_dir = Path(".").resolve()
main_dir = home_dir.parent.parent
print(main_dir)
```

```
data_dir = str(main_dir) + "/data"
print(data_dir)
```

# Load the dataset

- Let’s load the entire dataset

```
household_poverty = pd.read_csv(data_dir + "/costa_rica_poverty.csv")
print(household_poverty.head())
```

```
  household_id         ind_id  rooms  ...  age  Target  monthly_rent
0     21eb7fcc1  ID_279628684      3  ...   43       4      190000.0
1     0e5d7a658  ID_f29eb3ddd      4  ...   67       4      135000.0
2     2c7317ea8  ID_68de51c94      8  ...   92       4           NaN
3     2b58d945f  ID_d671db89c      5  ...   17       4      180000.0
4     2b58d945f  ID_d56d6f5f5      5  ...   37       4      180000.0

[5 rows x 84 columns]
```

- The entire dataset consists of `9557` observations and `84` variables

# Converting the target variable

- Let's convert poverty to a target variable with two levels, which will help to balance it out
- The four original levels would also increase the complexity of the visualizations and the code
- For this reason, we will convert levels 1, 2 and 3 to `vulnerable` and 4 to `non_vulnerable`
- The levels translate to 1, 2 and 3 as being **vulnerable** households
- Level 4 is **non-vulnerable**

```
household_poverty['Target'] = np.where(household_poverty['Target'] <= 3,
'vulnerable','non_vulnerable')
```

```
print(household_poverty['Target'].head())
```

```
0    non_vulnerable
1    non_vulnerable
2    non_vulnerable
3    non_vulnerable
4    non_vulnerable
Name: Target, dtype: object
```

# Data prep: target

- The next step of our data cleanup is to ensure the target variable is binary and has a label
- Let's look at the `dtype` of `Target`

```
print(household_poverty.Target.dtypes)
```

```
object
```

- We want to convert this to `bool` so that it is a binary class

```
household_poverty["Target"] = np.where(household_poverty["Target"] == "non_vulnerable", True,
False)
# Check class again.
print(household_poverty.Target.dtypes)
```

```
bool
```

# Data prep: check NAs

- Let's now check for missing data or otherwise known as NAs in the dataset

```
household_poverty.isnull().sum()
```

```
household_id          0
ind_id                0
rooms                 0
tablet                0
males_under_12        0
                    ...
urban_zone            0
rural_zone            0
age                   0
Target                0
monthly_rent       6860
Length: 84, dtype: int64
```


DATASOCIETY: © 2022

# Subsetting the dataset

- We don't want to use `monthly_rent` as a variable right now because there are so many NAs
- We're also removing `ID` variables
- Save as `costa_tree`

```
costa_tree = household_poverty.drop(['household_id', 'ind_id', 'monthly_rent'], axis = 1)
print(costa_tree.head())
```

```
   rooms  tablet  males_under_12  ...  rural_zone  age  Target
0      3       0               0  ...           0   43    True
1      4       1               0  ...           0   67    True
2      8       0               0  ...           0   92    True
3      5       1               0  ...           0   17    True
4      5       1               0  ...           0   37    True

[5 rows x 81 columns]
```

# Pickle library

- We are going to pause for a moment to learn about the library `pickle`
- When we have objects we want to carry over and do not want to rerun code, we can `pickle` these objects
- In other words, `pickle` will help us save objects from one script/ session and pull them up in new scripts
- How do we do that? We use a function in Python called `pickle`
- It is similar to **flattening** a file
  - **Pickle/saving:** a Python object is converted into a byte stream
  - **Unpickle/loading**: the inverse operation where a byte stream is converted back into an object

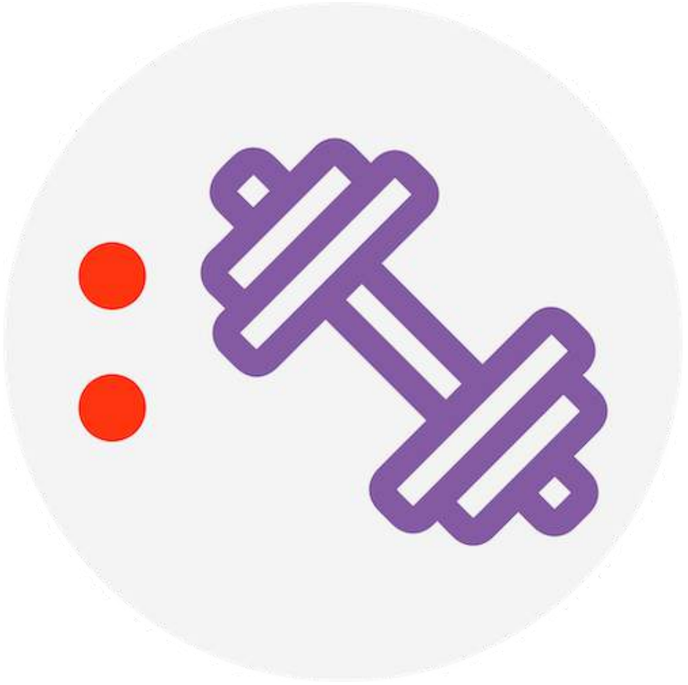
# Pickle the dataset

- Let's pickle the cleaned `Costa Rican` dataset so that we can use it later

```
pickle.dump(costa_tree, open(data_dir + "/costa_clean.sav", "wb" ))
```


DATASOCIETY: © 2022

# Exercise 1



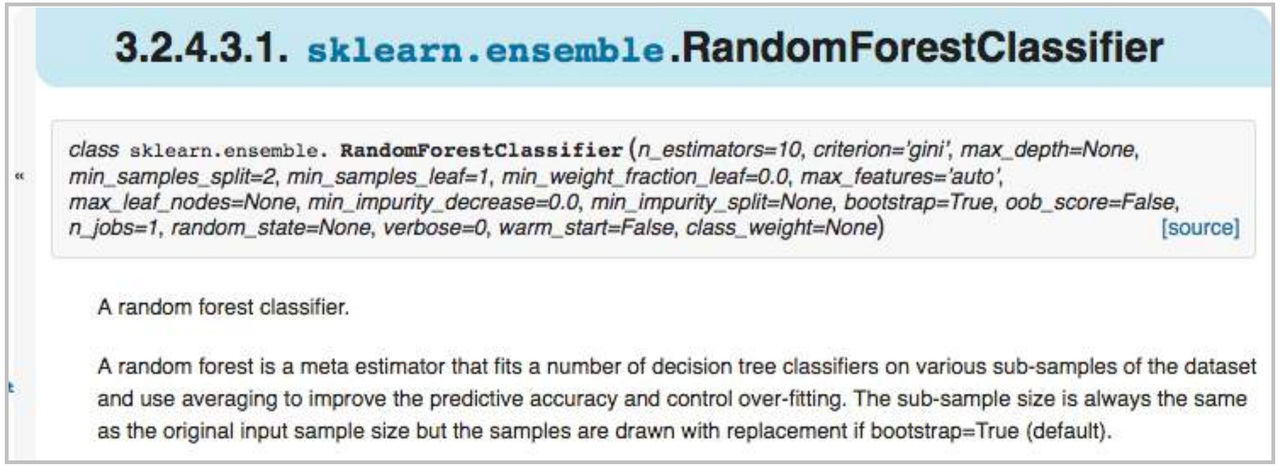
# Scikit-learn

- We will be using the well known Python library `scikit-learn` today
- Scikit-learn is used for many machine learning algorithms
- Here is a quick overview of some of the popular methods scikit-learn touches

| ML methods | Purpose |
|---|---|
| Clustering | Unsupervised learning methods such as k-means |
| Classification and regression | Supervised learning methods like generalized linear models, logistic regression, support vector machines, and decision trees |
| Cross validation | Estimating the performance of supervised models |
| Dimensionality reduction | Feature selection and feature extraction methods |
| Ensemble methods | Combining predictions of multiple supervised models |
| Parameter tuning | Adjusting model parameters to get the most out of models |
| Manifold learning | Summarizing and depicting complex multi-dimensional data |

# Scikit-learn: random forest

- We will be using the `RandomForestClassifier` library from `scikit-learn`



### 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier

*class* sklearn.ensemble. **RandomForestClassifier** (*n_estimators=10, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None*) [source]

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

- The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement as long as `bootstrap = True` (default)
- For all the parameters of the `tree` package, visit ***scikit-learn's documentation***

# Split into training and test sets

```
# Select the predictors and target.
X = costa_tree.drop(['Target'], axis = 1)
y = np.array(costa_tree['Target'])

# Set the seed to 1.
np.random.seed(1)

# Split into the training and test sets.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
```


DATASOCIETY: © 2022

# RandomForestClassifier

- We introduced the package `RandomForestClassifier` earlier today
- We are now going to use it to build a random forest on our clean data
- First, let's look at the **methods** available once the model is built

**Methods**

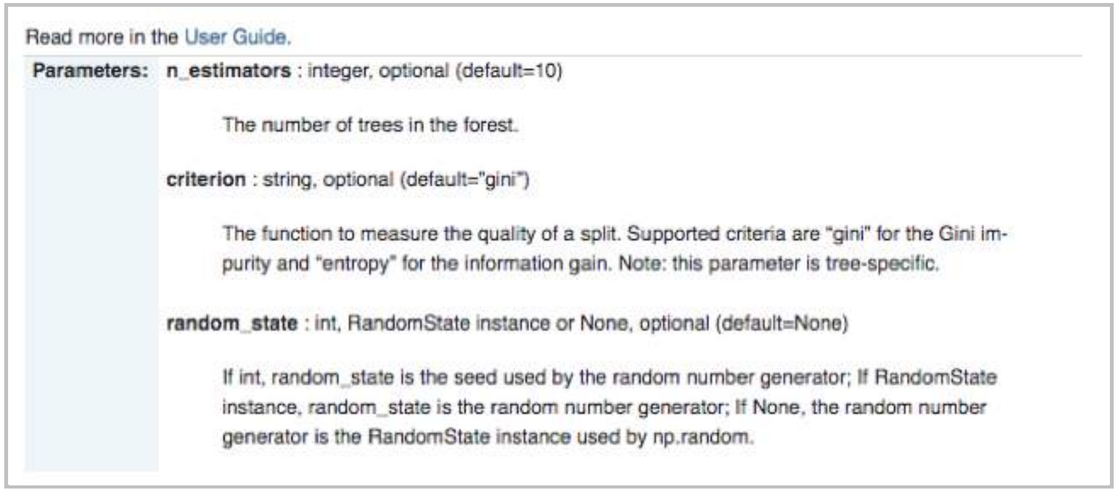
| | |
|---|---|
| `apply` (X) | Apply trees in the forest to X, return leaf indices. |
| `decision_path` (X) | Return the decision path in the forest |
| `fit` (X, y[, sample_weight]) | Build a forest of trees from the training set (X, y). |
| `get_params` ([deep]) | Get parameters for this estimator. |
| `predict` (X) | Predict class for X. |
| `predict_log_proba` (X) | Predict class log-probabilities for X. |
| `predict_proba` (X) | Predict class probabilities for X. |
| `score` (X, y[, sample_weight]) | Returns the mean accuracy on the given test data and labels. |
| `set_params` (**params) | Set the parameters of this estimator. |

- We are going to:
  - **Build** the random forest model
  - **Fit** the model to the training data
  - **Predict** on the test data using our trained model

# Building our model

- Let’s build our random forest model and use all default parameters for now, as our baseline model

```
forest = RandomForestClassifier(criterion = 'gini',
                                n_estimators = 100,
                                random_state = 1)
```



Read more in the User Guide.

**Parameters:**

**n_estimators** : integer, optional (default=10)

The number of trees in the forest.

**criterion** : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

**random_state** : int, RandomState instance or None, optional (default=None)

If int, random_state is the seed used by the random number generator; If RandomState instance, random_state is the random number generator; If None, the random number generator is the RandomState instance used by np.random.

# Fitting our model

- **Fit** the model to the training data

```
# Fit the saved model to your training data.
forest.fit(X_train, y_train)
```

```
RandomForestClassifier(random_state=1)
```


DATASOCIETY: © 2022

# Predicting with our data

- We will predict on the test data using our trained model
- The result is a **vector of the predictions**

```
# Predict on test data.
y_predict_forest = forest.predict(X_test)

# Look at the first few predictions.
print(y_predict_forest[0:5,])
```

```
[ True  True  True  True  True]
```

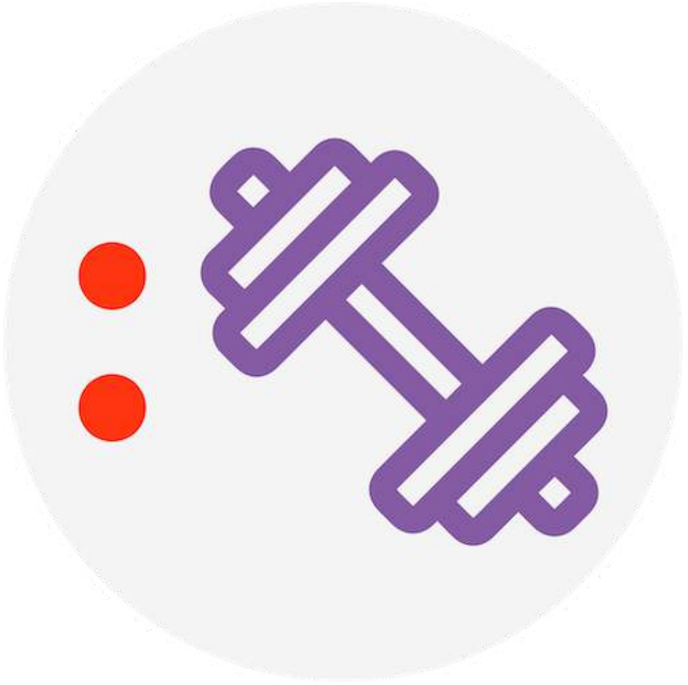
# Module completion checklist

| Objective | Complete |
| --- | --- |
| Introduce random forest and discuss use cases | ✔ |
| Summarize the concepts associated with random forest and bagging | ✔ |
| Load dataset and implement random forest | ✔ |


DATASOCIETY: © 2022

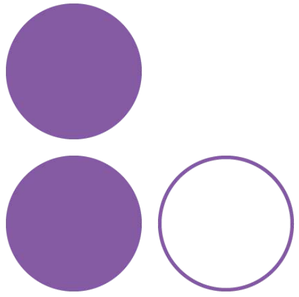
# Knowledge Check 2

# Exercise 2




DATASOCIETY: © 2022

# Congratulations on completing this module!




DATASOCIETY: © 2022