20 Minute Data Science Crash Course for 2020

The ultimate resource to help you get a job at top tech companies

Terence Shin Follow
Mar 30 · 20 min read ★

Image by Amanda Fawcett

This article serves as an extensive crash-course on what I believe are some of the most fundamental and instrumental concepts that you need to know to be a Data Scientist. I have broken this down into various sections so that you can go through this bit by bit.

Okay, this does not cover **everything** related to data science (that would be impossible) and no, this should **not** be the only resource that you use to develop your knowledge and skills…

HOWEVER, if you know nothing then this will help you develop a good understanding of the basics of data science. And if you have some understanding of data science, this serves as a compact crash course to be used as a refresher, hone your knowledge, and/or identify gaps in your knowledge.

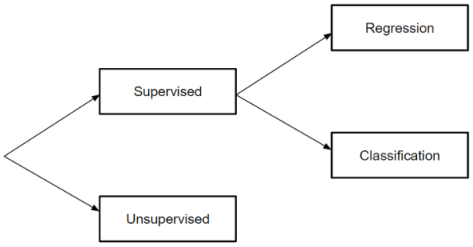
As always, I hope you find this helpful and wish you the best of luck in your data science endeavors!

. . .

Table of Content

. . .

1. MACHINE LEARNING MODELS



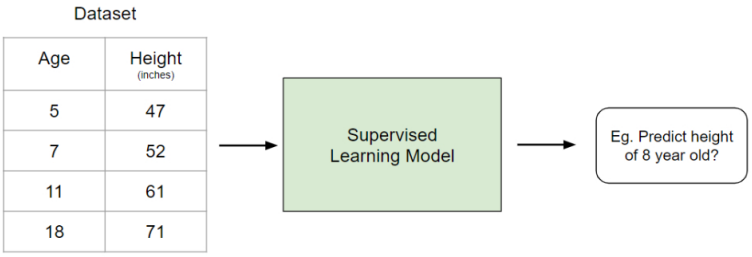
Fundamental Segmentation of Machine Learning Models

All machine learning models are categorized as either **supervised** or **unsupervised**. If the model is a supervised model, it's then sub-categorized as either a **regression** or **classification** model. We'll go over what these terms mean and the corresponding models that fall into each category below.

Supervised Learning

Supervised learning involves learning a function that maps an input to an output based on example input-output pairs.

For example, if I had a dataset with two variables, age (input) and height (output), I could implement a supervised learning model to predict the height of a person based on their age.



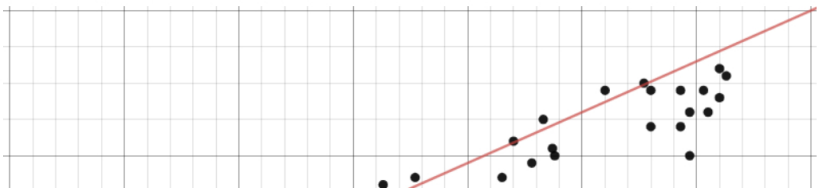
Example of Supervised Learning

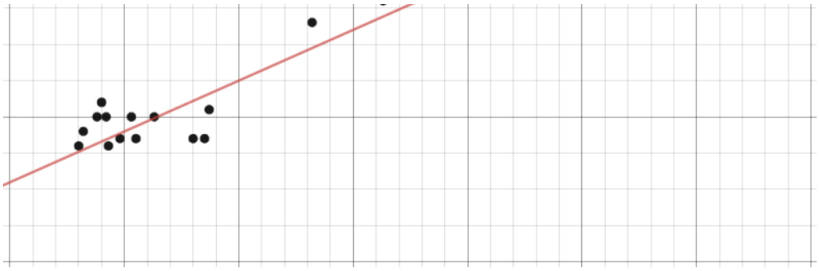
To re-iterate, within supervised learning, there are two sub-categories: regression and classification.

Regression

In **regression** models, the output is continuous. Below are some of the most common types of regression models.

Linear Regression





Example of Linear Regression

The idea of linear regression is simply finding a line that best fits the data. Extensions of linear regression include multiple linear regression (eg. finding a plane of best fit) and polynomial regression (eg. finding a curve of best fit). You can learn more about linear regression in my previous article.

Decision Tree

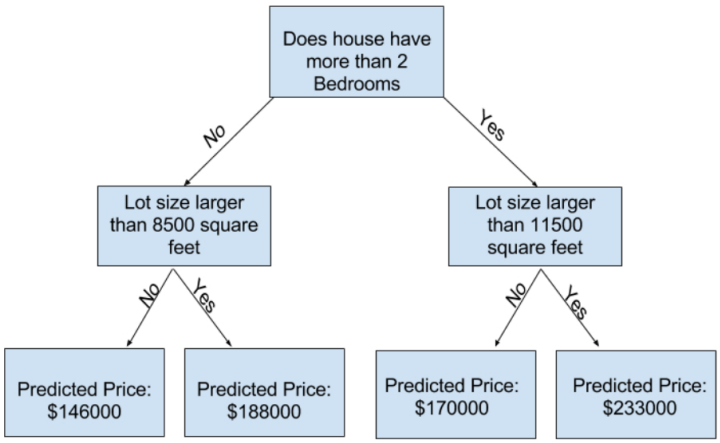
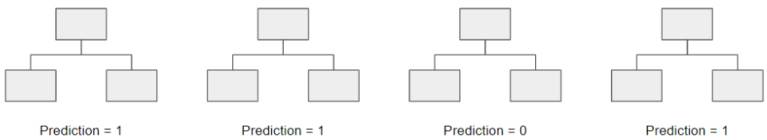


Image taken from Kaggle

Decision trees are a popular model, used in operations research, strategic planning, and machine learning. Each square above is called a **node**, and the more nodes you have, the more accurate your decision tree will be (generally). The last nodes of the decision tree, where a decision is made, are called the **leaves** of the tree. Decision trees are intuitive and easy to build but fall short when it comes to accuracy.

Random Forest

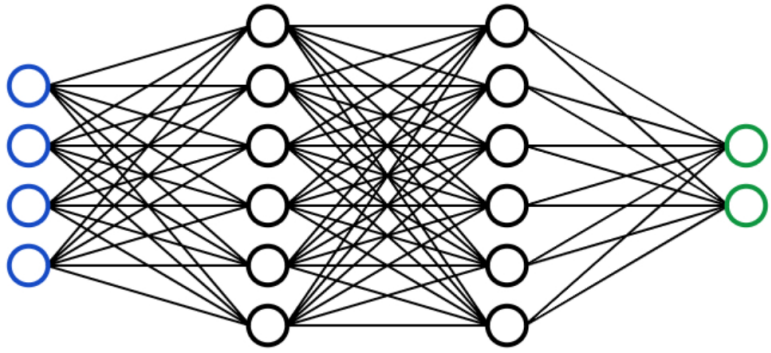
Random forests are an ensemble learning technique that builds off of decision trees. Random forests involve creating multiple decision trees using bootstrapped datasets of the original data and randomly selecting a subset of variables at each step of the decision tree. The model then selects the mode of all of the predictions of each decision tree. What's the point of this? By relying on a "majority wins" model, it reduces the risk of error from an individual tree.



For example, if we created one decision tree, the third one, it would predict 0. But if we relied on the mode of all 4 decision trees, the predicted value would be 1. This is the power of random forests.

StatQuest does an amazing job walking through this in greater detail. See here.

Neural Network



Visual Representation of a Neural Network

A **neural network** is a multi-layered model inspired by the human brain. Like the neurons in our brain, the circles above represent a node. The blue circles represent the **input layer**, the black circles represent the **hidden layers**, and the green circles represent the **output layer.** Each node in the hidden layers represents a function that the inputs go through, ultimately leading to an output in the green circles.

Neural networks are actually very complex and very mathematical, so I won't get into the details of it but…

Tony Yiu's article gives an intuitive explanation of the process behind neural networks (see here).

If you want to take it a step further and understand the math behind neural networks, check out this free online book here.

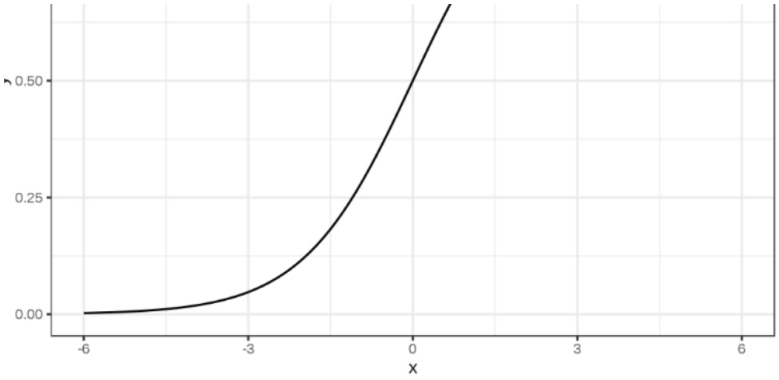
If you're a visual/audio learner, 3Blue1Brown has an amazing series on neural networks and deep learning on YouTube here.

Classification

In classification models, the output is discrete. Below are some of the most common types of classification models.

Logistic Regression

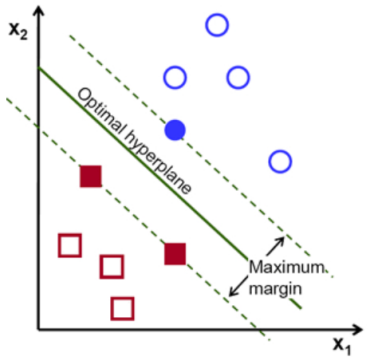
Logistic regression is similar to linear regression but is used to model the probability of a finite number of outcomes, typically two. There are a number of reasons why logistic regression is used over linear regression when modeling probabilities of outcomes (see here). In essence, a logistic equation is created in such a way that the output values can only be between 0 and 1 (see below).



Support Vector Machine

A **Support Vector Machine** is a supervised classification technique that can actually get pretty complicated but is pretty intuitive at the most fundamental level.

Let's assume that there are two classes of data. A support vector machine will find a **hyperplane** or a boundary between the two classes of data that maximizes the margin between the two classes (see below). There are many planes that can separate the two classes, but only one plane can maximize the margin or distance between the classes.



If you want to get into greater detail, Savan wrote a great article on Support Vector Machines here.

Naive Bayes

Naive Bayes is another popular classifier used in Data Science. The idea behind it is driven by Bayes Theorem:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

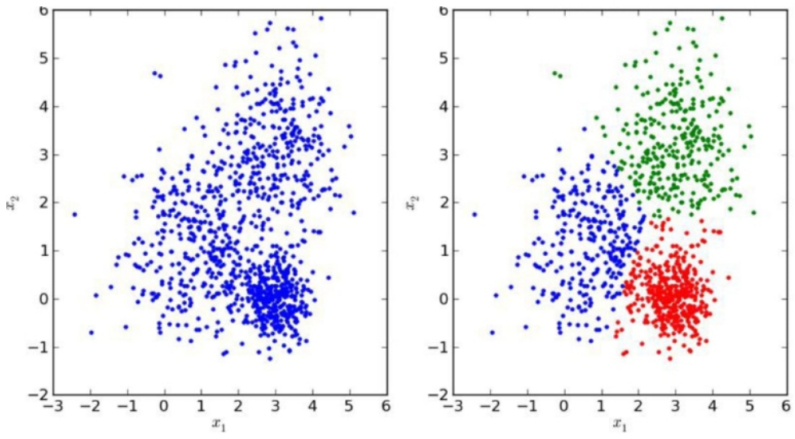
While there are a number of unrealistic assumptions made in regards to Naive Bayes (hence why it's called 'Naive'), it has proven to perform quite most of the time and it is also relatively fast to build.

See here if you want to learn more about them.

Decision Tree, Random Forest, Neural Network

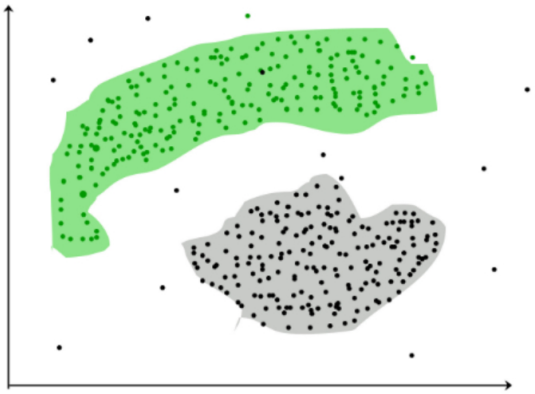
These models follow the same logic as previously explained. The only difference is that that output is discrete rather than continuous.

Unsupervised Learning



Unlike supervised learning, **unsupervised learning** is used to draw inferences and find patterns from input data without references to labeled outcomes. Two main methods used in unsupervised learning include clustering and dimensionality reduction.

Clustering



Taken from GeeksforGeeks

Clustering is an unsupervised technique that involves the grouping, or **clustering**, of data points. It's frequently used for customer segmentation, fraud detection, and document classification.

Common clustering techniques include **k-means** clustering, **hierarchical** clustering, **mean shift** clustering, and **density-based** clustering. While each technique has a different method in finding clusters, they all aim to achieve the same thing.

Dimensionality Reduction

Dimensionality reduction is the process of reducing the number of random variables under consideration by obtaining a set of principal variables. In simpler terms, its the process of reducing the dimension of your feature set (in even simpler terms, reducing the number of features). Most dimensionality reduction techniques can be categorized as either **feature elimination** or **feature extraction.**

A popular method of dimensionality reduction is called **principal**

A popular method of dimensionality reduction is called **principal component analysis.**

Principal Component Analysis (PCA)

In the simplest sense, **PCA** involves project higher dimensional data (eg. 3 dimensions) to a smaller space (eg. 2 dimensions). This results in a lower dimension of data, (2 dimensions instead of 3 dimensions) while keeping all original variables in the model.

There is quite a bit of math involved with this. If you want to learn more about it…

Check out this awesome article on PCA here.

If you'd rather watch a video, StatQuest explains PCA in 5 minutes here.

. . .

2. STATISTICS

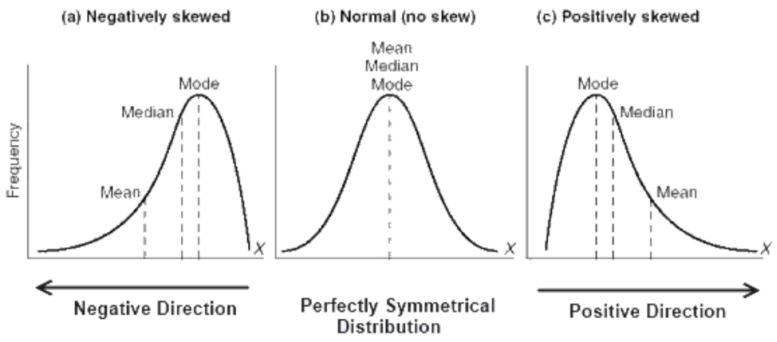
Data Types

Numerical: data expressed with digits; is measurable. It can either be **discrete** or **continuous**.

Categorical: qualitative data classified into categories. It can be **nominal** (not ordered) or **ordinal** (ordered data).

Measures of Central Tendency

Mean: the average of a dataset.
Median: the middle of an ordered dataset; less susceptible to outliers.
Mode: the most common value in a dataset; only relevant for discrete data.



Measures of Variability

Range: the difference between the highest and lowest value in a dataset.

Variance (σ2): measures how spread out a set of data is relative to the mean.

$$\sigma^2 = \frac{\sum_{i=1}^{n} (x_i - \bar{x})^2}{n}$$

$$\sigma^2 = \frac{\sum_{i=1} \quad}{n}$$

Standard Deviation (σ): another measurement of how spread out numbers are in a data set; it is the square root of variance.

Z-score: determines the number of standard deviations a data point is from the mean.

$$z = \frac{x_i - \mu}{\sigma}$$

R-Squared: a statistical measure of fit that indicates how much variation of a dependent variable is explained by the independent variable(s); only useful for simple linear regression.

$$R^2 = 1 - \frac{\text{Explained Variation}}{\text{Total Variation}}$$

Adjusted R-squared: a modified version of r-squared that has been adjusted for the number of predictors in the model; it increases if the new term improves the model more than would be expected by chance and vice versa.

Measurements of Relationships between Variables

Covariance: Measures the variance between two (or more) variables. *If it's positive then they tend to move in the same direction, if it's negative then they tend to move in opposite directions, and if they're zero, they have no relation to each other.*

$$\sigma_{XY} = \frac{\sum_{i=1}^{n} (X_i - \mu_X)(Y_i - \mu_Y)}{n}$$

Denominator becomes (n-1) for samples

Correlation: Measures the strength of a relationship between two variables and ranges from -1 to 1; the normalized version of covariance. Generally, a correlation of +/- 0.7 represents a strong relationship between two variables. On the flip side, correlations between -0.3 and 0.3 indicate that there is little to no relationship between variables.

$$Correlation = \frac{Cov\,(x, y)}{\sigma x * \sigma y}$$

Probability Distribution Functions

Probability Density Function (PDF): a function for continuous data where the value at any point can be interpreted as providing a *relative* likelihood that the value of the random variable would equal that sample. (Wiki)

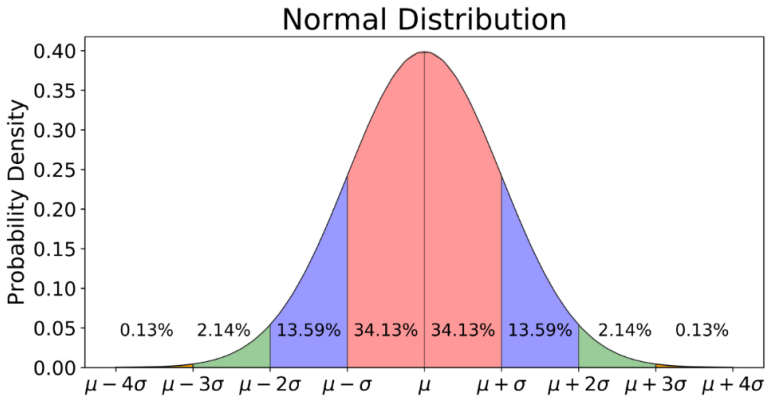
Probability Mass Function (PMF): a function for discrete data which gives the probability of a given value occurring.

Cumulative Density Function (CDF): a function that tells us the probability that a random variable is less than a certain value; the integral of the PDF.

Continuous Data Distributions

Uniform Distribution: a probability distribution where all outcomes are equally likely.

Normal/Gaussian Distribution: commonly referred to as the bell curve and is related to the **central limit theorem**; has a mean of 0 and a standard deviation of 1.



T-Distribution: a probability distribution used to estimate population parameters when the sample size is small and/r when the population variance is unknown (*see more here*).

Chi-Square Distribution: distribution of the chi-square statistic (*see here*).

Discrete Data Distributions

Poisson Distribution: probability distribution that expresses the probability of a given number of events occurring within a fixed time period.

Binomial Distribution: a probability distribution of the number of successes in a sequence of n independent experiences each with its own Boolean-valued outcome (p, 1-p).

Moments

Moments describe different aspects of the nature and shape of a distribution. The first moment is the **mean,** the second moment is the **variance,** the third moment is the **skewness,** and the fourth moment is the **kurtosis**.

Accuracy

True positive: detects the condition when the condition is present.

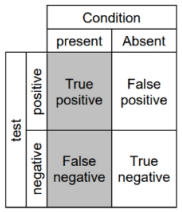
True negative: does not detect the condition when the condition is not present.

False-positive: detects the condition when the condition is absent.

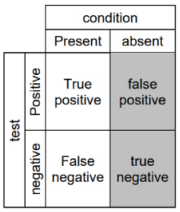
False-negative: does not detect the condition when the condition is present.

Sensitivity: also known as **recall**; measures the ability of a test to detect the condition when the condition is present; sensitivity = TP/(TP+FN)

Specificity: measures the ability of a test to correctly exclude the condition when the condition is absent; specificity = TN/(TN+FP)



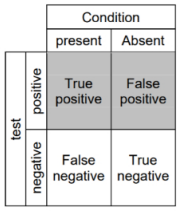
Sensitivity



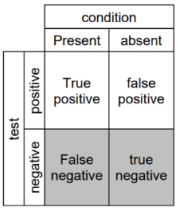
Specificity

Predictive value positive: also known as **precision**; the proportion of positives that correspond to the presence of the condition; PVP = TP/(TP+FP)

Predictive value negative: the proportion of negatives that correspond to the absence of the condition; PVN = TN/(TN+FN)



Predictive value positive



Predictive value negative

Hypothesis Testing and Statistical Significance

Check out my article 'Hypothesis Testing Explained as Simply as Possible' for a deeper explanation here.

Null Hypothesis: the hypothesis that sample observations result purely from chance.

Alternative Hypothesis: the hypothesis that sample observations are influenced by some non-random cause.

P-value: the probability of obtaining the observed results of a test, assuming that the null hypothesis is correct; a smaller p-value means that there is stronger evidence in favor of the alternative hypothesis.

Alpha: the significance level; the probability of rejecting the null hypothesis when it is true — also known as **Type 1 error.**

Beta: type 2 error; failing to reject the null hypothesis that is false.

Steps to Hypothesis testing:
1. State the null and alternative hypothesis
2. Determine the test size; is it a one or two-tailed test?
3. Compute the test statistic and the probability value
4. Analyze the results and either reject or do not reject the null hypothesis (*if the p-value is greater than the alpha, do not reject the null!)*

. . .

3. PROBABILITY

Probability is the likelihood of an event occurring.

Conditional Probability [P(A|B)] is the likelihood of an event occurring, based on the occurrence of a previous event.

Independent events are events whose outcome does not influence the probability of the outcome of another event; $P(A|B) = P(A)$.

Mutually Exclusive events are events that cannot occur simultaneously; $P(A|B) = 0$.

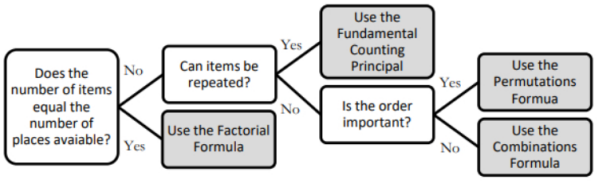
Bayes' Theorem: a mathematical formula for determining conditional probability. *"The probability of A given B is equal to the probability of B given A times the probability of A over the probability of B".*

$$P(A|B) = \frac{P(A \bigcap B)}{P(B)} = \frac{P(A) \cdot P(B|A)}{P(B)}$$

Eight rules of probability

- Rule #1: For any event A, **0 ≤ P(A) ≤ 1**; *in other words, the probability of an event can range from 0 to 1.*
- Rule #2: The sum of the probabilities of all possible outcomes always equals 1.
- Rule #3: **P(not A) = 1 — P(A)**; *This rule explains the relationship between the probability of an event and its complement event. A complement event is one that includes all possible outcomes that aren't in A.*
- Rule #4: If A and B are disjoint events (mutually exclusive), then **P(A or B) = P(A) + P(B)**; *this is called the addition rule for disjoint events*
- Rule #5: **P(A or B) = P(A) + P(B) — P(A and B)**; *this is called the general addition rule.*
- Rule #6: If A and B are two independent events, then **P(A and B) = P(A) * P(B)**; *this is called the multiplication rule for independent events.*
- Rule #7: The conditional probability of event B given event A is **P(B|A) = P(A and B) / P(A)**
- Rule #8: For any two events A and B, **P(A and B) = P(A) * P(B|A)**; *this is called the general multiplication rule*

Counting Methods



Factorial Formula: n! = n x (n -1) x (n — 2) x … x 2 x 1
Use when the number of items is equal to the number of places available.
Eg. Find the total number of ways 5 people can sit in 5 empty seats.
= 5 x 4 x 3 x 2 x 1 = 120

Fundamental Counting Principle (multiplication)
This method should be used when repetitions are allowed and the number of ways to fill an open place is not affected by previous fills.
Eg. There are 3 types of breakfasts, 4 types of lunches, and 5 types of desserts. The total number of combinations is = 5 x 4 x 3 = 60

Permutations: P(n,r)= n! / (n−r)!
This method is used when replacements are not allowed and order of item ranking matters.

Eg. A code has 4 digits in a particular order and the digits range from 0 to 9. How many permutations are there if one digit can only be used once?
P(n,r) = 10!/(10–4)! = (10x9x8x7x6x5x4x3x2x1)/(6x5x4x3x2x1) = 5040

Combinations Formula: C(n,r)=(n!)/[(n−r)!r!]
This is used when replacements are not allowed and the order in which items are ranked does not mater.
Eg. To win the lottery, you must select the 5 correct numbers in any order from 1 to 52. What is the number of possible combinations?
C(n,r) = 52! / (52–5)!5! = 2,598,960

. . .

4. PANDAS

Pandas is a software library in Python used for data manipulation and analysis. It is universal in the data science world and is essential to know! Below is a guide to learning basic Pandas functionality.

Setup

Import Pandas library

```
import pandas as pd
```

Creating and Reading Data

Create a DataFrame

A DataFrame is simply a table made up of multiple arrays. In the example below, the code would create a table with two columns, ABC and DEF.

```
pd.DataFrame({'ABC':[1,2,3],'DEF':[4,5,6]},index=[1,2,3])
```

Create a Series

A Series is a sequence of values, also known as a list. From a visual perspective, imagine it being one column of a table.

```
pd.Series([1,2,3],index=[], name ='ABC')
```

Read a CSV file into a data frame

The most common way of getting our data. This converts a CSV file into a DataFrame.

```
# example
df = pd.read_csv("filename.csv", index_col=0)
```

Convert a DataFrame into a CSV file

Vice versa, if you want to convert a DataFrame into a CSV, you can use the code below:

```
# example
df.to_csv("filename.csv", index_col=0)
```

Determine the shape of a DataFrame

This tells you how large a DataFrame is and is in the format (rows, columns).

```
df.shape()
```

See the first 5 rows of a DataFrame

If you want to get a visual idea of what a DataFrame looks like, **.head()** returns the first 5 rows of the given DataFrame.

```
df.head()
```

See the data type of one or more columns

```
# For one column
df.variable.dtype

# For all columns
df.dtypes
```

Convert a column to another data type

This is useful if you want to convert integers to floats (or vice versa).

```
df.variable.astype()
```

Manipulating DataFrames

Selecting a Series from a Dataframe

```
# a) Method 1
df.property_name

# b) Method 2
df['property_name']
```

Indexing a Series

```
# if you want to get the first value in a series
df['property_name'][0]
```

Index-based Selection

Index-based selection retrieves data based on its numerical position in the DataFrame. It follows a rows-first, columns-second format. Iloc's indexing scheme is such that the **first number is inclusive** and the **last number is exclusive**.

```
df.iloc[]
```

Label-based Selection

Label-based selection is another way to index a DataFrame, but it retrieves data based on the actual data values rather than the numerical position. Loc's indexing scheme is such that the **both the first and last values are inclusive.**

```
df.loc[]
```

Set index using existing columns

Because label-based selection relies on the index of the DataFrame, you can use **.set_index()** to assign a column to the index.

```
df.set_index("variable")
```

Conditional Label-based Selection

We can filter out a DataFrame using Label-based selection too.

```
# a) Single Condition
df.loc[df.property_name == 'ABC']

# b) Multiple conditions using AND
df.loc[df.property_name == 'ABC' & df.property_name == 'DEF']

# c) Multiple conditions using OR
df.loc[df.property_name == 'ABC' | df.property_name == 'DEF']
```

Select where value is in a list of values

We can use **isin()** to filter a DataFrame as well. If you know SQL, it's similar to the WHERE ___ IN() statement.

```
df.loc[df.property_name.isin(['ABC','DEF'])
```

Select where values are null / aren't null

The first line of code will filter a DataFrame to only show rows where the property name is null.
Vice versa, the second line of code with the filter it so that the property name is not null.

```
df.loc[df.property_name.isnull()]
```

```
df.loc[df.property_name.notnull()]
```

Adding a new column

```
df['new_column'] = 'ABC'
```

Renaming a column

You'll often want to rename a column to something easier to refer to. Using the code below, the column ABC would be renamed to DEF.

```
df.rename(columns={'ABC': 'DEF'})
```

Summary Functions

.describe()

This gives a high-level summary of a DataFrame or a variable. It is type-sensitive, meaning that its output will be different for numerical variables compared to string variables.

```
df.describe()
df.variable.describe()
```

.mean()

This returns the average of a variable.

```
df.variable.mean()
```

.unique()

This returns all of the unique values of a variable.

```
df.variable.unique()
```

.value_counts()

This shows a list of unique values and also the frequency of occurrence in the DataFrame.

```
df.variable.value_counts()
```

Mapping Functions

.map()

Mapping is used to transform an initial set of values to another set of values through a function. For example, we could use mapping to convert the

values of a column from meters to centimeters or we could normalize the values.

.map() is used to transform a Series.

```
df.numerical_variable.map()
```

.apply()

.apply() is similar to .map(), except that it transforms the entire DataFrame.

```
df.numerical_variable.apply()
```

Grouping and Sorting

.groupby()

Get the count for each value of a variable (*same as value_counts*)

```
df.groupby('variable').variable.count()
```

Get the min value for each value of a variable

```
df.groupby('variable').variable.min()
```

Get a summary (length, min, max) for each value of a variable

```
df.groupby(['variable']).variable.agg([len, min, max])
```

Multi-indexing

```
df.groupby(['variable_one', 'variable_two'])
```

Sorting a DataFrame

Sorting by one variable

```
df.sort_values(by='variable', ascending=False)
```

Sorting by multiple variables

```
df.sort_values(by=['variable_one', 'variable_two'])
```

Sorting by index

```
df.sort_index()
```

Handling Missing Data

Handling missing data is one of the most important steps in EDA. Below are a number of ways to handle missing data.

Drop rows with null values

If you have a DataFrame with a large number of rows and you can afford to remove rows with null values completely, then .dropna() is a useful tool.

```
df.dropna()
```

Drop columns with null values

This is similar to above except that it drops any **columns** with null values rather than the rows.

```
df.dropna(axis=1)
```

Fill missing values

If you would rather fill missing values instead of removing the row or column completely, you can use the code below:

```
df.variable.fillna("n/a")
```

Replace values

Let's say there's a DataFrame where someone already filled missing values with "n/a", but you want the missing values to be filled with "unknown". Then you can use the following code below:

```
df.variable.replace("n/a", "unknown")
```

Combining Data

.concat()

This is useful when you want to combine two DataFrames that have the same columns. For example, if we wanted to combine January sales and February sales together to analyze longer-term trends, you could use the following code:

```
Jan_sales = pd.read_csv("jan_sales.csv")
Feb_sales = pd.read_csv("feb_sales.csv")

pd.concat([Jan_sales, Feb_sales])
```

.join()

If you want to combine two columns that have a common index (e.g. customer_id), then you can use .join().

Use the parameter, **on**, to determine what column to join on.

To determine if it's a left, right, inner, or outer join, you use the parameter, **how**.

```
# example
table_1.join(table_2, on='customer_id', how='left')
```

If you don't know about SQL joins, read here. It's essentially the same idea.

. . .

5. SQL & QUERYING

Definitions

A **row**, also called a **record**, is a collection of attributes (variables) that represent a single entity. For example, one row may represent one hospital patient and may have attributes/variables like age, weight, height, etc…

ID	Name	Age	Weight_lbs	Height_cm	
1	Bob	22	155	175	<-- example of a row
2	Adrian	25	140	160	
3	Jessica	21	105	145	
4	Michelle	26	120	155	

A **table** is a collection of rows with the same attributes (with the same variables). What helps me the most is to think of a table as an Excel table.

table name: patient_info

ID	Name	Age	Weight_lbs	Height_cm
1	Bob	22	155	175
2	Adrian	25	140	160
3	Jessica	21	105	145
4	Michelle	26	120	155

example of a table

A **query** is a request for data from a database table or combination of tables. Using the table above, I would write a **query** if I wanted to find all patients that were older than 23 years old.

How to Write A SQL

Since this is a tutorial for beginners, I'm going to show you how to write a query if you wanted to extract data from **one** table.

There are **five** components to a basic query:

1. SELECT (mandatory)
2. FROM (mandatory)

2. FROM (mandatory)

3. WHERE (optional)

4. GROUP BY (optional)

5. ORDER BY (optional)

The structure is as follows:

```
SELECT
   [column_name_1],
   [column_name_2],
   [column_name_n]
FROM
   [table_name]
WHERE
   [condition 1]
GROUP BY
   [column_name]
ORDER BY
   [column_name]
```

Let's bring back my example as a reference:

table name: patient_info

ID	Name	Age	Weight_lbs	Height_cm
1	Bob	22	155	175
2	Adrian	25	140	160
3	Jessica	21	105	145
4	Michelle	26	120	155

1. SELECT (Mandatory)

SELECT determines which columns you want to pull from a given table. For example, if I wanted to pull Name then my code would look like:

```
SELECT Name
```

A neat trick is if you want to pull **all** columns, you can use an asterisk — see below:

```
SELECT *
```

2. FROM (Mandatory)

table name: patient_info

ID	Name	Age	Weight_lbs	Height_cm
1	Bob	22	155	175
2	Adrian	25	140	160
3	Jessica	21	105	145
4	Michelle	26	120	155

table name: incident

ID	Date	Reason	Urgency	Location_id
1	12/1/2019	Fever	low	0002
2	12/12/2019	Wrist injury	low	0673
3	12/24/2019	Ankle injury	medium	0231
4	12/30/2019	Chest pain	high	0923

FROM determines which table you want to pull the information from. For example, if you wanted to pull the Name of the patient, you would want to pull the data FROM the table called patient_info (see above). The code would look something like this:

```
SELECT
   Name
FROM
   patient_info
```

```
    patient_info
```

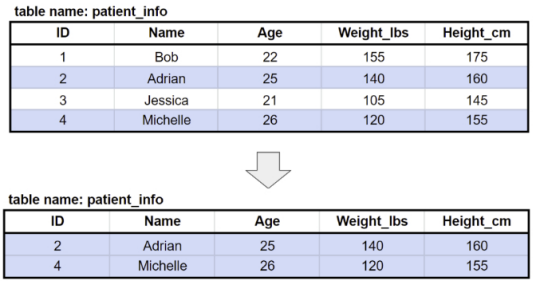
And there's your first functional query! Let's go through the 3 additional optional steps.

3. WHERE (optional)

What if you wanted to select the Names of patients who are older than 23? This is when WHERE comes in. **WHERE is a statement used to filter your table**, the same way you would use the filter tool in Excel!

The code to get the Names of patients who are older than 23 is to the left. A visual representation is shown to the right:

```
SELECT
   Name
FROM
   patient_info
WHERE
   Age > 23
```



If you want the Names of patients that satisfy **two** clauses, you can use **AND**. *Eg. Find the Names of patients who are older than 23 and weigh more than 130 lbs.*

```
SELECT
   Name
FROM
   patient_info
WHERE
   Age > 23
   AND
   Weight_lbs > 130
```

If you want the Names of patients that satisfy **one of two** clauses, you can use **OR**. *Eg. Find the Names of patients who are younger than 22 or older than 23.*

```
SELECT
   Name
FROM
   patient_info
WHERE
   Age < 22
   OR
   Age > 23
```

4. GROUP BY (optional)

GROUP BY does what it says — **it groups rows that have the same values into summary rows**. It is typically used with aggregate functions like COUNT, MIN, MAX, SUM, AVG.

Let's use the example below:

table name: incident

ID	Date	Reason	Urgency	Location_id

1	12/1/2019	Fever	low	0002
2	12/12/2019	Wrist injury	low	0673
3	12/24/2019	Ankle injury	medium	0231
4	12/30/2019	Chest pain	high	0923
2	12/31/2019	Wrist injury follow up	low	0673

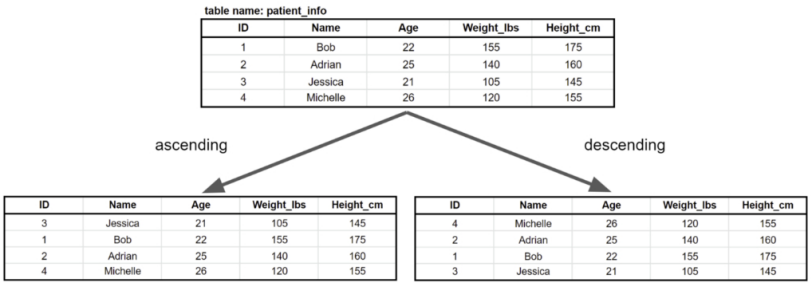
If we wanted to get the number of hospital visits for each patient, we could use the code below and get the following result:

```
SELECT
   ID,
   COUNT(ID)
FROM
   incident
GROUP BY
   ID
```

ID	Count
1	1
2	2
3	1
4	1

5. ORDER BY (optional)

ORDER BY allows you to sort your results based on a particular attribute or a number of attributes in ascending or descending order. Let's show an example.




```
SELECT
   *
FROM
   patient_info
ORDER BY
   Age asc
```

'ORDER BY Age asc' means that your result set will order the rows by age in ascending order (see the left table in the image above). If you want to order

it in descending order (right table in the image above), you would replace **asc** with **desc.**

Now that you've learned the basic structure, the next step is to learn about SQL Joins, which you can read about here.

. . .

6. BONUS CONTENT

If you got to the end of this, congrats! I hope this inspires you to continue your data science journey. The truth is that there's so much more to learn about each topic that I wrote about, but luckily there are thousands of resources out there that you can use!

Below are some additional resources and tutorials that you can use to continue your learnings:

- **A Guide to Build Your First Machine Learning Model and Start Your Data Science Career:** Refer to this if you've never created a machine learning model and don't know where to start.
- **An Extensive Step by Step Guide to Exploratory Data Analysis**: Exploring your data is essential for every dataset that you work with. Go through this article to learn what EDA is and how to conduct it.
- **How to Evaluate Your Machine Learning Models with Python Code!**: Creating your machine learning model is one thing. Creating a **good** machine learning model is another. This article teaches you how to evaluate whether you've built a good machine learning model or not.
- **OVER 100 Data Scientist Interview Questions and Answers!**: Once you've built a strong data science portfolio and you feel ready to search for a job, use this resource to help you prepare for your job search.

Thanks for Reading!

If you like my work and want to support me…

1. The BEST way to support me is by following me on **Medium** here.
2. Be one of the FIRST to follow me on **Twitter** here. *I'll be posting lots of updates and interesting stuff here!*
3. Also, be one of the FIRST to subscribe to my new **YouTube channel** here!
4. Follow me on **LinkedIn** here.
5. Sign up on my **email list** here.
6. Check out my website, **terenceshin.com**.

Data Science | Artificial Intelligence | Technology | Programming | Education

394 claps

WRITTEN BY

Terence Shin

Follow

MSc Analytics | Data Scientist | Entrepreneur | Content Creator | Follow me on Twitter: @terence_shin | Send any inquiries to terenceshin1997@gmail.com

Follow

Towards Data Science

A Medium publication sharing concepts, ideas, and codes.

Follow

See responses (4)

More From Medium

Bye-bye Python. Hello Julia!

Rhea Moutafis in Towards Data Science

Machine Learning Engineers Will Not Exist In 10 Years.

Luke Posey in Towards Data Science

If I had to start learning data science again, how would I do it?

Santiago Víquez Segura in Towards Data Science

Discover Medium

Welcome to a place where words matter. On Medium, smart voices and original ideas take center stage - with no ads in sight. Watch

Make Medium yours

Follow all the topics you care about, and we'll deliver the best stories for you to your homepage and inbox. Explore

Become a member

Get unlimited access to the best stories on Medium — and support writers while you're at it. Just $5/month. Upgrade

Medium

About Help Legal