IBM Data Science Professional Certificate

Table of Contents

[1 What is Data Science 2](#_Toc11795446)

[2 Open Source tools for Data Science 2](#_Toc11795447)

[3 Data Science Methodology 2](#_Toc11795448)

[3.1 Data Science Methodology in a Nutshell 2](#_Toc11795449)

[3.2 Data Science Methodology Steps 2](#_Toc11795450)

[4 Python for Data Science 5](#_Toc11795451)

[4.1 Data Definition 5](#_Toc11795452)

[4.2 Python Data Types 6](#_Toc11795453)

[4.3 Expressions and Variables 7](#_Toc11795454)

[4.4 String 7](#_Toc11795455)

[4.5 List and Tuples 8](#_Toc11795456)

[4.6 Dictionaries {} - keys: immutable, values: immutable, mutable 9](#_Toc11795457)

[4.7 Sets {} - mutable 9](#_Toc11795458)

[4.8 Conditions and Branching 10](#_Toc11795459)

[4.9 Loops 10](#_Toc11795460)

[4.10 Functions 10](#_Toc11795461)

[4.11 Objects and Classes 11](#_Toc11795462)

[4.12 Methods 12](#_Toc11795463)

[4.13 Read, Write and Append Files 12](#_Toc11795464)

[4.14 Pandas 13](#_Toc11795465)

[4.15 NumPy – One Dimensional 13](#_Toc11795466)

[4.16 NumPy – Two Dimensional 14](#_Toc11795467)

[5 Databases and SQL for Data Science 15](#_Toc11795468)

[5.1 Databases Terminology 15](#_Toc11795469)

[5.2 Database Instances 15](#_Toc11795470)

[5.3 Information Model vs Data Model 16](#_Toc11795471)

[5.4 Relational Model Constraints 16](#_Toc11795472)

[5.5 SQL Statements 16](#_Toc11795473)

[5.6 Using String Patterns, Ranges, Sort, Grouping 17](#_Toc11795474)

[5.7 Sub-Queries and Nested Selects 18](#_Toc11795475)

[5.8 Working with Multiple Tables 18](#_Toc11795476)

[5.9 Access Databases using Python 18](#_Toc11795477)

[5.10 JOIN Overview 19](#_Toc11795478)

[6 Data Analysis with Python 20](#_Toc11795479)

[6.1 Understanding the Data 20](#_Toc11795480)

[6.2 Python Packages for Data Science 20](#_Toc11795481)

[6.3 Importing and Exporting Data 21](#_Toc11795482)

[6.4 Analyzing Data 21](#_Toc11795483)

[6.5 Pre-processing Data 21](#_Toc11795484)

[6.6 Exploratory Data Analysis (EDA) 23](#_Toc11795485)

[6.7 Model Development 24](#_Toc11795486)

[6.8 Model Evaluation using Visualization 25](#_Toc11795487)

[6.9 Polynomial Regression and Pipelines 26](#_Toc11795488)

[6.10 Model Evaluation and Refinement 27](#_Toc11795489)

[7 Data Visualization with Python 30](#_Toc11795490)

[7.1 Matplotlib - Introduction 30](#_Toc11795491)

[7.2 Matplotlib - Architecture Layers 30](#_Toc11795492)

[7.3 Matplotlib - Types of Plots 31](#_Toc11795493)

[8 Machine Learning with Python 32](#_Toc11795494)

[8.1 Introduction to Machine Learning 32](#_Toc11795495)

[8.2 Regression Models 33](#_Toc11795496)

[8.3 Classification 38](#_Toc11795497)

[8.4 Clustering 43](#_Toc11795498)

[8.5 Recommender Systems 45](#_Toc11795499)

[9 Applied Data Science Capstone 47](#_Toc11795500)

# What is Data Science

Talking talking talking.

# Open Source tools for Data Science

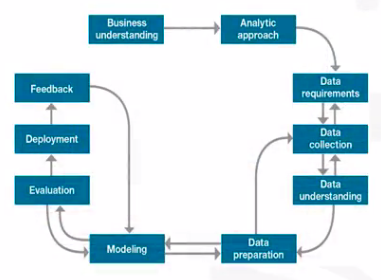
IBM Watson – 50 hrs free (lite). IBMid: <https://cognitiveclass.ai/courses/> and https://labs.cognitiveclass.ai/

OTHER OPTIONS: 1) anaconda, use “Jupyter Labs”, or anaconda prompt, type “Jupyter Notebook“

# Data Science Methodology

## Data Science Methodology in a Nutshell

by John Rollins – IBM,

*From problem to approach:*

1. What is the problem that you are trying to solve?
2. How can you use data to answer the question?

*Working with the data:*

1. What data do you need to answer the question?
2. Where is the data coming from (identify all sources) and how will you get it?
3. Is the data that you collected representative of the problem to be solved?
4. What additional work is required to manipulate and work with the data?

*Deriving the answer:*

1. In what way can the data be visualized to get to the answer that is required?
2. Does the model used really answer the initial question or does it need to be adjusted?
3. Can you put the model into practice?
4. Can you get constructive feedback into answering the question?

## Data Science Methodology Steps

### Business Understanding

Data science methodology begins with spending the time to seek clarification, to attain what can be referred to as a business understanding. Having this understanding is placed at the beginning of the methodology because getting clarity around the problem to be solved, allows you to determine which data will be used to answer the core question.

Once the goal is clarified, the next piece of the puzzle is to figure out the objectives that are in support of the goal.

### Analytics Approach

Selecting the right analytic approach depends on the question being asked. The approach involves seeking clarification from the person who is asking the question, so as to be able to pick the most appropriate path or approach.

*Descriptive*

* Current status
* Relationships, clusters

*Diagnostic (Statistical Analysis)*

* What happened?
* Why is it happening?
* counts

*Predictive (Forecasting)*

* What if these trends continue?
* What will happen next?
* Probability, classification (yes/no answer)

*Prescriptive*

* How do we solve it?

Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed. Machine Learning can be used to identify relationships and trends in data that might otherwise not be accessible or identified. Machine Learning is a field of study that gives computers the ability to learn without being explicitly programmed.

Machine Learning can be used to identify relationships and trends in data that might otherwise not be accessible or identified. In the case where the question is to learn about human behavior, then an appropriate response would be to use Clustering Association approaches.

### Data Requirements

Prior to undertaking the data collection and data preparation stages of the methodology, it's vital to define the data requirements. This includes identifying the necessary data content, formats and sources for initial data collection.

Thinking ahead and anticipating subsequent methodology stages is important.

### Data Collection

After the initial data collection is performed, an assessment by the data scientist takes place to determine whether or not they have what they need. In this phase the data requirements are revised and decisions are made as to whether or not the collection requires more or less data. Once the data ingredients are collected, then in the data collection stage, the data scientist will have a good understanding of what they will be working with.

When collecting data, it is alright to defer decisions about unavailable data, and attempt to acquire it at a later stage.

Techniques such as descriptive statistics and visualization can be applied to the data set, to assess the content, quality, and initial insights about the data. Gaps in data will be identified and plans to either fill or make substitutions will have to be made.

### Data Understanding

Data understanding encompasses all activities related to constructing the data set. Essentially, the data understanding section of the data science methodology answers the question: Is the data that you collected representative of the problem to be solved?

Statistical analysis, plots, correlations, missing values, creating/renaming values to determine data quality.

### Data Preparation

In a sense, data preparation is similar to washing freshly picked vegetables in so far as unwanted elements, such as dirt or imperfections, are removed. Together with data collection and data understanding, data preparation is the most time-consuming phase of a data science project, typically taking seventy percent and even up to even ninety percent of the overall project time. Automating some of the data collection and preparation processes in the database, can reduce this time to as little as 50 percent. This time savings translates into increased time for data scientists to focus on creating models.

Data preparation/cleansing: invalid, missing, duplicate and formatting

*Feature engineering* (part of data preparation) is the process of using domain knowledge of the data to create features that make the machine learning algorithms work. A feature is a characteristic that might help when solving a problem. Features within the data are important to predictive models and will influence the results you want to achieve. Feature engineering is critical when machine learning tools are being applied to analyze the data.

While this phase may take a while to do, if done right the results will support the project. If this is skipped over, then the outcome will not be up to par and may have you back at the drawing board. It is vital to take your time in this area, and use the tools available to automate common steps to accelerate data preparation.

### Modeling

Data Modelling focuses on developing models that are either *descriptive* or *predictive*. An example of a *descriptive* model might examine things like: if a person did this, then they're likely to prefer that. A *predictive* model tries to yield yes/no, or stop/go type outcomes. These models are based on the analytic approach that was taken, either statistically driven or machine learning driven.

The data scientist will use a *training* set for *predictive* modelling. A training set is a set of historical data in which the outcomes are already known. The training set acts like a gauge to determine if the model needs to be calibrated. In this stage, the data scientist will play around with different algorithms to ensure that the variables in play are actually required.

The success of data compilation, preparation and modelling, depends on the understanding of the problem at hand, and the appropriate analytical approach being taken. The data supports the answering of the question, and like the quality of the ingredients in cooking, sets the stage for the outcome. Constant refinement, adjustments and tweaking are necessary within each step to ensure the outcome is one that is solid.

In John Rollins' descriptive Data Science Methodology, the framework is geared to do 3 things:

1. First, understand the question at hand.
2. Second, select an analytic approach or method to solve the problem, and
3. third, obtain, understand, prepare, and model the data.

The end goal is to move the data scientist to a point where a data model can be built to answer the question.

In this stage of the methodology, model evaluation, deployment, and feedback loops ensure that the answer is near and relevant. This relevance is critical to the data science field overall, as it is a fairly new field of study, and we are interested in the possibilities it has to offer.

**Type 1 – false positive YES accuracy sensitivity**

**Type 2 – false negative NO accuracy specificity**

### Evaluating the Model

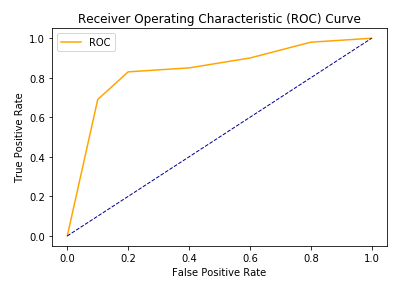
A model evaluation goes hand-in-hand with model building as such, the modeling and evaluation stages are done iteratively. Model evaluation is performed during model development and before the model is deployed. Evaluation allows the quality of the model to be assessed but it's also an opportunity to see if it meets the initial request.

Model evaluation can have two main phases.

* The first is the *diagnostic measures* phase, which is used to ensure the model is working as intended. If the model is a *predictive* model, a decision tree can be used to evaluate if the answer the model can output, is aligned to the initial design. It can be used to see where there are areas that require adjustments. If the model is a *descriptive* model, one in which relationships are being assessed, then a testing set with known outcomes can be applied, and the model can be refined as needed.
* The second phase of evaluation that may be used is *statistical significance* testing. This type of evaluation can be applied to the model to ensure that the data is being properly handled and interpreted within the model. This is designed to avoid unnecessary second guessing when the answer is revealed.

**ROC curve**

The optimal model is the one giving the *maximum separation* between the blue ROC ((*Receiver Operating Characteristic*) curve relative to the red base line. ROC plot is True Positive (Sensitivity, y-axis) vs False Positive Rate (1-Specificity, x-axis).



Source: <https://stackabuse.com/understanding-roc-curves-with-python/>

### Deployment

While a data science model will provide an answer, the key to making the answer relevant and useful to address the initial question, involves getting the stakeholders familiar with the tool produced. Once the model is evaluated and the data scientist is confident it will work, it is deployed and put to the ultimate test. Depending on the purpose of the model, it may be rolled out to a limited group of users or in a test environment, to build up confidence in applying the outcome for use across the board.

### Feedback

Once in play, feedback from the users will help to refine the model and assess it for performance and impact. The value of the model will be dependent on successfully incorporating feedback and making adjustments for as long as the solution is required.

# Python for Data Science

## Data Definition

**Attribute: column name**

**Feature: column data**

**Observation: row data**

**Target: what we want to predict**

**Dataframe: data under analysis, in rows and columns. Each column is a series.**

**CATEGORICAL:**

**Nominal variables that have two or more categories, but which do not have an intrinsic order. Embarked-port name**

**Dichotomous nominal variable with only two categories. Sex**

**Ordinal variables that have two or more categories just like nominal variables. Only the categories can also be ordered or ranked. Passenger Class**

**NUMERIC:**

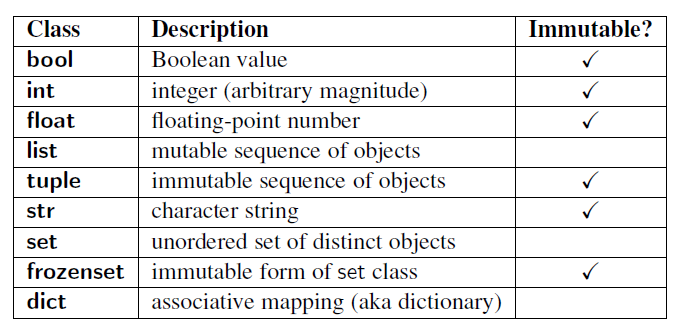
**Discrete: PassengerID, SiblingSpouseCount, Survived**

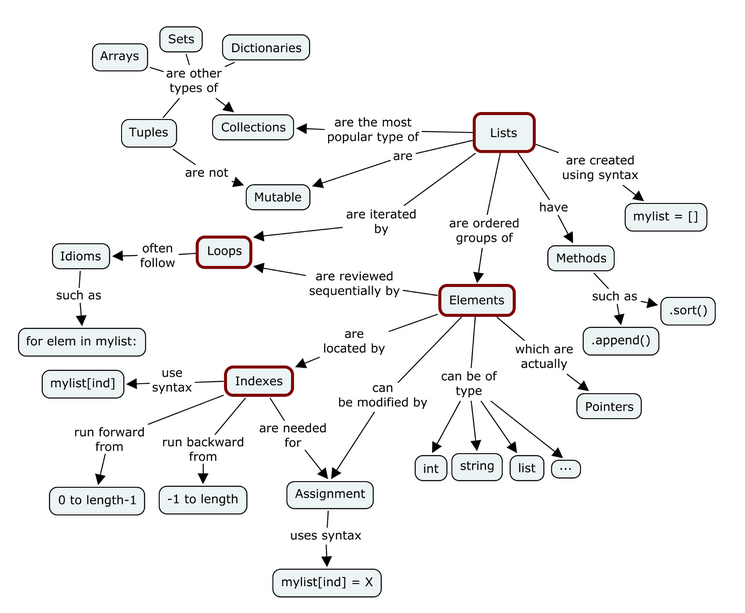
**Continuous: Age, Fare - Interval and Ratio**

**TEXT: Ticket Number (alphanumeric), Passenger Name**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****DATA-    DATA CAN BE CLASSIFIED INTO TWO DIFFERENT TYPES.**** | | | | |
|  | | | | |
| ****CATEGORICAL** represents characteristics, and are values or observations that can be sorted into groups or categories.**  **Bar charts and pie graphs are used to graph categorical data.** | |  | ****NUMERICAL** values or observations that can be measured, and placed in ascending or descending order.**  **Scatter plots and line graphs are used to graph numerical data.** | |
|  | |  |  | |
| ****NOMINAL** Nominal values represent discrete units and are used to label variables, that have no quantitative value. Just think of them as “labels”. You can count but not order or measure nominal data.**  **Examples: Sex, and eye color.** | ****ORDINAL** Values or observations can be ranked (put in order) or have a rating scale attached. You can count and order, but not measure, ordinal data.**  **Example: house numbers and swimming level.** |  | ****DISCRETE** Values or observations that are counted as distinct and separate and can only take particular values. They cannot be measured.**  **Examples: number of threads in a sheet, number of stars given for an energy rating.** | ****CONTINUOUS** Continuous Data represents measurements and therefore their values can’t be counted but they can be measured. Values or observations may take on any value within a finite or infinite interval.**  **Examples: height, time and temperature.** |

Everything in Python is an object, and all objects in Python can be either **mutable** or **immutable**. A **mutable** object can be changed after it is created, and an **immutable** object can’t.





Source: <https://medium.com/@meghamohan/mutable-and-immutable-side-of-python-c2145cf72747>

## Python Data Types

Each type is an *object*:

**Int -9,0,54,123**

**Float -2.3,0.0,5.36,698**

**String ‘a’,’Michael J’,’124.2’**

**Tuple (‘hard’,43,’apple’)**

**List [1,34,’abc’]**

**Dictionary {‘dog’:1,’cat’:2}**

**Set {‘rock’,’mango’,1,-98.45,’123’}**

**Boolean True, False**

* **type(11) int**
* **type(True) bool**

*type casting:*

**float(2):2.0**

**int(1.1):1**

**int(‘1’):1**

**int(‘A’) - gives an error**

**str(1):”1”**

**str(4.5):’4.5’**

**int(True):1**

**int(False):0**

### **bool(1):True**

**bool(0):False**

## Expressions and Variables

**+ - \* /**

**11//2 = 5 - integer division**

**my\_variable = 1**

**x = 4 - x is 4**

**x = x/2 - x is 2.0**

## String

The r means that the string is to be treated as a raw string, which means all escape codes will be ignored

**print(r"hashajsh \n kjhashajksh") - hashajsh \n kjhashajksh**

### Ordered sequence:

**Name = ‘Michael Jackson’**

**Name[4] “h”**

**Name[-1] “n”, last**

**Name[-7] “J”**

**Name[0:4] “Mich”**

**Name[8:12] “Jack”**

**Name[::2] “McalJcsn”, every 2nd value**

**Name[0:5:2] “Mca”, every 2nd value till index 4**

**Len(Name) 15**

**Stat=Name+”is best” “Michael Jackson is best”**

**3\*”Michael Jackson” “Micheal JacksonMicheal JacksonMicheal Jackson**

### Strings are Immutable:

**Name[0]=”J” error, cannot do this**

### String and Sequence Methods

B=Name.upper() 'MICHAEL JACKSON'

B = Name.replace(‘Michael’,’Janet’) 'Janet Jackson'

Name.find(‘el’) 5, start of index

Name.find(‘z’) -1, not found

## List and Tuples

(aka compound data types)

### Tuples () - immutable

Tuples, tuples are an ordered sequence, are expressed as comma separated elements within parentheses. These are values inside the parentheses. Tuples are *immutable*, which means we can't change them.

**tuple1 = (‘disco’, 10, 1.2)**

**tuple1[1] 10**

**tuple1[-1] 1.2**

**tuple2 = tuple1 + ('hard rock', 10)**

**tuple2 ('disco', 10, 1.2, 'hard rock', 10)**

**tuple2[1:4] (10, 1.2, ‘hard rock’)**

**ratings = (10,9,6,5,10,8)**

**ratings1 = ratings**

**ratings[1] = 100 error**

**ratings = (2,3,4) tuple can be replaced by another tuple**

*nesting***:**

**NT = (1,2, ('pop', 'rock'), 3,4,("disco",(1,2)))**

**NT[2] ('pop', 'rock')**

**NT[2][1] ‘rock’**

**NT[2][1][3] ‘k’**

**('pop', 'rock')[1] “rock”**

**NT[5] ('disco', (1, 2))**

**NT[5][1] (1, 2)**

**NT[5][1][0] 1**

### Lists [] - mutable

A list is represented with square brackets. In many respects, lists are like tuples. One key difference is they are *mutable*. Lists can contain strings, floats, integers. We can nest other lists. We also nest tuples and other data structures.

**L = ['M J', 10.1, 1983, [1,2], ('A',1)]**

**len(L) 5**

**L[3][0] 1**

**L.extend(['pop',102]) ['M J', 10.1, 1983, [1, 2], ('A', 1), 'pop', 102]**

**len(L) 7**

**L1 = L + ['zip', 101] ['M J', 10.1, 1983, [1, 2], ('A', 1), 'pop', 102, 'zip', 101]**

**len(L1) 9**

**append** adds only ONE element to the list

**L.append([‘zag’,103]) ['M J', 10.1, 1983, [1, 2], ('A', 1), 'pop', 102, ['zag', 103]]**

**len(L) 8**

**L[1] = ‘hard’ ['M J', 'hard', 1983, [1, 2], ('A', 1), 'pop', 102, ['zag', 103]]**

**L1 (stays same) ['M J', 10.1, 1983, [1, 2], ('A', 1), 'pop', 102, 'zip', 101]**

**split**

**del(L[1]) ['M J', 1983, [1, 2], ('A', 1), 'pop', 102, ['zag', 103]]**

**“hello world”.split() [‘hello’,’world’]**

**'A,B,C,D'.split(',') ['A', 'B', 'C', 'D']**

**aliasing**

When we set one variable B equal to A, both A and B are referencing the same list. Multiple names referring to the same object is known as aliasing. We know from the list slide that the first element in B is set as hard rock. If we change the first element in A to banana, we get a side effect. The value of B would change as a consequence.

**A = ['hard rock', 10, 1.2]**

**B = A**

**A[0] = 'banana' ['banana', 10, 1.2]**

**B ['banana', 10, 1.2]**

**clone**

**B = A[:]**

**A[0] = 'mango'**

**A ['mango', 10, 1.2]**

**B ['banana', 10, 1.2]**

## Dictionaries {} - keys: immutable, values: immutable, mutable

Dictionaries are a type of collection in Python. A dictionary has keys and values. The key is analogous to the index, they are like addresses but they don't have to be integers. They are usually characters. The values are similar to the element in a list and contain information.

To create a dictionary,

* use curly brackets
* keys are the first elements - immutable and unique
* each key is followed by a value separated by a colon
* values can be immutable, mutable, and duplicates
* each key and value pair is separated by a comma

**A = {'a':3,'b':46,'c':'monkey','d':[1,5,3]}**

**A['a'] 3**

**A['c'] 'monkey'**

**A[‘d’] = “added value’ # add new key ‘d’**

**A {'a': 3, 'b': 46, 'c': 'monkey', 'd': 'added value'}**

**del(A['c']) # delete key ‘c’**

**A {'a': 3, 'b': 46, 'd': 'added value'}**

**'a' in A True # works with only keys**

**A.keys() dict\_keys(['a', 'b', 'd'])**

**A.values() dict\_values([3, 46, 'added value'])**

## Sets {} - mutable

Set are unordered, unlike lists and tuples. This means sets do not record element position. Sets only have unique elements. This means there is only ONE of a particular element in a set.

You notice there are duplicate items. When the actual set is created, duplicate items will not be present

**set1 = {'rock', 'mango', 'rock', 'soul', 'rock', 'disco'}**

**set1 {'disco', 'mango', 'rock', 'soul'} # note duplicate ‘rock’ is not present**

*type-casting:*

You can convert a list to a set by using the function set, this is called type casting

**album\_list = ['M J', 'Thriller', 'Thriller','1982']**

**album\_list ['M J', 'Thriller', 'Thriller', '1982']**

**album\_set = set(album\_list)**

**album\_set {'1982', 'M J', 'Thriller'}**

*add/remove:*

**set1 {'disco', 'mango', 'rock', 'soul'}**

**set1.add(‘pear’) {'disco', 'mango', 'pear', 'rock', 'soul'}**

**set1.add(‘pear’) {'disco', 'mango', 'pear', 'rock', 'soul'} # no duplicates in set**

**set1.remove('rock') {'disco', 'mango', 'pear', 'soul'}**

**'pear' in set1 True**

**'rock' in set1 False**

*intersection, union, issubset – Venn diagrams:*

**setA = {'red', 'white', 'blue', 'green'}**

**setB = {'yellow','gray','green','red'}**

**setA & setB {'green', 'red'} # intersection operation**

**setA.union(setB) {'blue', 'gray', 'green', 'red', 'white', 'yellow'} # union**

**(setA & setB).issubset(setA) True**

## Conditions and Branching

### Comparison Operator

**==, !=, >, >=, <, <= boolean: True or False**

**Case sensitive**

### Branching

### Logic Operator

**Boolean: NOT, OR, AND**

**a = 11**

**if (a > 10) or (a < 30):**

**print(a)**

## Loops

**range(M,N,step) [M,..N-1,step] # will not output values in Python 3**

**for i in range(10,23,2):**

**print(i) 10 12 14 16 18 20 22**

### Enumerate for Lists

**squares = ['red','yellow','green']**

**for i, j in enumerate(squares):**

**print(i,j)**

**0 red**

**1 yellow**

**2 green**

## Functions

Piece of code you can reuse.

**len, sum, sorted, sort, print**

**def add1(a):**

**“””**

**Add 1**

**“””**

**b = a + 1**

**return(b)**

**add1(5)**

**output: 6**

**help(add1) “Add 1”**

*sorted* – function that returns a new list, it does not change the list

*sort* – function that changes the list, no new list is created

*reverse* - opposite of sort

Function with no return statement will return “**None**”

Function with integer and string will not error, but return a string and continue.

Python does not allow an empty code block. Special key word “**pass**” can be used instead.

### Value Added Parameters (\*)

Allows us to input a variable number of elements

**def ArtistNames(\*names):**

**for name in names:**

**print(name)**

**ArtistNames("Pink", "U2", "Clash") Pink, U2, Clash**

**ArtistNames('Shaggy') Shaggy**

### Global and Local Variables

Variables inside of a function are *local variables*. If a variable is inside a function and not defined, it will check the *global variable* for the value. Key word **global** can be used inside of function to explicitly define it as global and is assigned for use outside the function (i.e. global)

## Objects and Classes

Each type is an *object*:

**Int -9,0,54,123**

**Float -2.3,0.0,5.36,698**

**String ‘a’,’Michael J’,’124.2’**

**Tuple (‘hard’,43,’apple’)**

**List [1,34,’abc’]**

**Dictionary {‘dog’:1,’cat’:2}**

**Set {‘rock’,’mango’,1,-98.45,’123’}**

**Boolean True, False**

### Objects

Every object has:

* A type
* An internal data representation (a blueprint)
* A set of procedures (*methods*) for interacting with the object
* An object is an *instance* of a particular type

Every time we create an integer, we are creating an *instance* of type integer, or we are creating an *integer object*. Similarly, every time we create a list, we are creating an *instance* of type list, or we are creating a *list object*.

### Methods

A *method* changes or interacts with an object. *A* *class* or *type's methods* are *functions* that every instance of that class or type provides.

**sort, add, remove, append, print**

*Sorting* is an example of a method that interacts with the data in the object. The *method* sort will change the data within the object. We call the method by adding a period at the end of the object's name, and the methods name we would like to call with parentheses.

### Class

*Class* has data attributes and methods. You can create objects or instances of that Class.

* **\_\_init\_\_** is a special method or constructor used to initialize data attributes. Function init is a constructor. It's a special function that tells Python you are making a new class. There are other special functions in Python to make more complex classes
* **self** parameter refers to the newly created instance of the class.
* **radius** and **color** parameters are used to initialize the radius and color data attributes of the class instance.

****# Class: Circle****

****# Data Attributes: raduis,color****

**class Circle(object):**

**# data attributes used to initialize each instance of class**

**def \_\_init\_\_(self,radius,color):**

**self.radius = radius;**

**self.color = color;**

**"""**

**C1=Circle(10,"red")**

**C1.color Out: 'red'**

**C1.radius Out: 10**

**"""**

****# Class: Rectangle****

****# Data Attributes: height,width,color****

**class Rectangle(object):**

**def \_\_init\_\_(self,height,width,color):**

**self.height = height;**

**self.width = width;**

**self.color = color;**

**"""**

**C2 = Rectangle(3,2,"green")**

**C2.height Out: 3**

**C2.width Out: 2**

**C2.color Out: 'green'**

**C2.color = 'red' # update color**

**C2.color Out:'red'**

**"""**

## Methods

Methods are functions that interact and change the data attributes, changing or using the data attributes of the object.

**class Circle(object):**

**# data attributes used to initialize each instance of class**

**def \_\_init\_\_(self,radius,color):**

**self.radius = radius;**

**self.color = color;**

**def add\_radius(self,r): # method**

**self.radius=self.radius+r**

**C1 = Circle(2,'red')**

**C1.radius 2**

**C1.add\_radius(8)**

**C1.radius 10**

**dir(Circle) attributes of object**

## Read, Write and Append Files

*Modes:*

r reading

w writing

a appending

### Read Files

**File1 = open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleR.txt","r")**

**File1.name "C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleR.txt"**

**File1.mode “r”**

**File1.close**

**# with open - automatically closes the file object**

**with open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/suicides.txt","r") as File1:**

**file\_stuff=File1.read()**

**#file\_stuff=File1.readlines()**

**print(file\_stuff)**

**print(File1.closed)**

**print(file\_stuff)**

### Write Files

**File2 = open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleW.txt","w")**

**File2.write('this is line A\n')**

**File2.write('this is line B\n')**

**Lines = ['this is line A\n','this is line B\n','this is line C\n']**

**with open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleW.txt","w") as File2:**

**for line in Lines:**

**File2.write(line)**

### Append Files

**with open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleW.txt","a") as File2:**

**File2.write('this is appended line D')**

**File2.close()**

### Copying Files

**with open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleR.txt","r") as readFile:**

**with open("C:/Users/ACER/Desktop/JAVA/IBM\_Certificate/exampleC.txt","w") as writeFile:**

**for line in readFile:**

**writeFile.write(line)**

## Pandas

import pandas as pd

df = pd.read\_csv('C:/Users/ACER/Desktop/JAVA/Kaggle/data/suicide-rates.csv')

df.head()

We can create a data frame out of a dictionary. We then cast the dictionary to a data frame using the function data frame.

# dictionary

**DIC = {'album': ['Thriller','Bat','Dark Side'], 'released': ['1982','1999','2005'], 'length': ['00:42:19','00:34:32','00:66:23']}**

**df2 = pd.DataFrame(DIC)**

**x = df[['length']] # create dataframe ‘x’ with ‘length’ column**

### ‘ix’ function (loc and iloc in newer versions)

**df2.ix[1,2] access row 1, column 2**

**df2.ix[0,'album']**

**z = df2.ix[0:2,0:3] slice dataframe**

**z = df2.ix[0:2,'album':'released']**

**df2.loc[0,'album'] label based indexing**

**df2.loc[2,'length'] positional indexing**

### List Unique Values

**df[‘year’].unique()**

**df['year'] >= 2012**

**df[df['year'] >= 2012]**

## NumPy – One Dimensional

Library for scientific computing. Much faster processing.

**import numpy as np**

### Basics and Array Creation

Numpy array is similar to a list, is fixed in size and contains the same datatype elements.

**a = np.array([0,1,2,3,4])**

**type(a) # numpy.ndarray**

**a.dtype # dtype('int32')**

**a.size # 5**

**a.ndim # 1**

**a.shape # (5,)**

### Indexing and Slicing

**c = np.array([20,1,2,3, 4]) array([20, 1, 2, 3, 4])**

**c[0] = 100 array([100, 1, 2, 3, 4])**

**c[1:4] array([1, 2, 3])**

### Basic Operations

*without numpy:*

**u = [1,0]**

**v = [0,1]**

**z = []**

**for n,m in zip(u,v):**

**z.append(n+m) [1,1]**

*with numpy:*

**u = np.array([1,0])**

**v = np.array([0,1])**

**z = u + v array([1, 1])**

*dot function:*

**np.dot(u,v) (u1 \* v1) + (u2 \* v2) = 0**

**a = np.array([[1,2], [3,4]])**

**b = np.array([[11,12],[13,14]])**

**np.dot(a,b) [[37 40]**

**[85 92]]**

**Calculations: [[1\*11+2\*13, 1\*12+2\*14],[3\*11+4\*13, 3\*12+4\*14]]**

*broadcasting (adding scalar value):*

**u + 1 array([2, 1])**

### Universal Functions

**a = np.array([1,2,3,4])**

**a.mean() 2.5**

**a.max() 4**

**x = np.array([0,np.pi/2,np.pi]) array([0., 1.57079633, 3.14159265])**

**np.sin(x) array([0.0000000e+00, 1.0000000e+00, 1.2246468e-16])**

**np.linspace(-3,3,num=5) array([-3. , -1.5, 0. , 1.5, 3. ])**

**x = np.linspace(0, 6\*np.pi, 100)**

**y = np.sin(x)**

**import matplotlib.pyplot as plt**

**# %matplotlib inline # for jupyter**

**plt.plot(x,y) plots x vs y**

## NumPy – Two Dimensional

**a = [[11,12,13],[21,22,23],[31,32,33]] # list**

**A = np.array(a)**

**A.ndim 2, number of lists in array (look at square brackets)**

**A.shape (3, 3), rows,columns**

**A.size 9, number of elements**

**A[1][2] 23**

**A[0,2] 13**

**A[0,0:2] array([11, 12])**

**A[0:2,0:3] array([[11, 12, 13]**

**[21, 22, 23]])**

**b = [[41,42,43],[51,52,53],[61,62,63]] #**

**B = np.array(b)**

**A + B array([[52, 54, 56],**

**[72, 74, 76],**

**[92, 94, 96]])**

**A\*2 array([[22, 24, 26],**

**[42, 44, 46],**

**[62, 64, 66]])**

**B/A array([[3.72727273, 3.5 , 3.30769231],**

**[2.42857143, 2.36363636, 2.30434783],**

**[1.96774194, 1.9375 , 1.90909091]])**

**A\*B array([[ 451, 504, 559],**

**[1071, 1144, 1219],**

**[1891, 1984, 2079]])**

*dot product*

num of columns in first matrix must match num of rows in second matric

**a = np.array([[1,2], [3,4]])**

**b = np.array([[11,12],[13,14]])**

**np.dot(a,b) [[37 40]**

**[85 92]]**

**Calculations: [[1\*11+2\*13, 1\*12+2\*14],[3\*11+4\*13, 3\*12+4\*14]]**

# Databases and SQL for Data Science

## Databases Terminology

**SQL** Language used for relational databases to query data

**Data** Collection of facts (words, numbers). Data is one of the most critical assets of any business, so it needs to be secure and it needs to be stored and accessed quickly

**Database** repository of data. Provides functionality for adding, modifying and querying the data. Data stored in tabular form is a *relational database*.

**DBMS** Set of software tools for the data in the database is called a database management system or DBMS. The terms database, database server, database system, data server and database management systems are often used interchangeably.

**RDBMS** Set of software tools that controls data stored in tabular form, such as access, organization and storage. For relational databases, it's called a relational database management system or RDBMS.

* *Primary key* of a relational table uniquely identifies each row in a table.
* *Foreign key* is a set of columns referring to a primary key of another table.
* *Referencing* establishes the data integrity between two relations.
* *Parent table* is a table containing a primary key that is related to at least one foreign key.
* *Child table* or *dependent table* is a table containing one or more foreign keys.

## Database Instances

* Database *services* are logical abstractions for managing workloads in a database.
* An *instance* of the Cloud database operates as a service that handles all application requests to work with the data and any of the databases managed by that instance.
* Database *service instance* is the target of the connection request from applications

Database service instances provide ease of use, access from anywhere, scalability and disaster recovery.

## Information Model vs Data Model

An *information model* is an abstract formal representation of entities that includes their properties, relationships, and the operations that can be performed on them. The entities being modeled can be from the real world, such as a library. Information models and data models are different and serve different purposes. An information model is at the conceptual level and defines relationships between objects. There are several types of information models. The most familiar is the hierarchical, typically used to show organization charts

*Data models* are defined in a more concrete level, are specific and include details. A data model is the blueprint of any database system. The relational model is the most used data model for databases, because this model allows for data independence. Data is stored in simple data structure tables. This provides logical data independence, physical data independence, and physical storage independence. An entity-relationship data model or ER data model, is an alternative to a relational data model.

An *entity-relationship model* (ER model) proposes thinking of a database as a collection of entities, rather than being used as a model on its own. The ER model is used as a tool to design relational databases. In the ER model, entities are objects that exist independently of any other entities in the database. The building blocks of an ER diagram are entities and attributes. Entities have attributes, which are the data elements that characterize the entity. Attributes are connected to exactly one entity.

### Types of Relationships

* One-to-one relationships
* One-to-many relationships
* Many-to-many relationships

### Mapping Entities to Tables

The entity book becomes a *table* in the database, and the attributes become the *columns* in a table.

### Relational Model Concepts

First proposed in 1970, based on mathematical model and terms. Building blocks: Relation & Sets

* **Relational database** is a **set** of relations. A relation is also the mathematical term for a table. A relation is made up of two parts, relational schema and relational instance.
  + *relational schema* specifies the name of a relation and the attributes.
  + *relational instance* is a table made up of the attributes or columns and the tuples or rows.
* **Set** is unordered collection of distinct elements. It is a collection of items of the same type. Sets have no order and no duplicates.
* **Table** is a combination of rows and columns. The *columns* are the attributes or fields. The *rows* are tuples.
* **Degree** refers to the number of attributes or *columns* in a relation.
* **Cardinality** refers to the number of tuples or *rows* in a relation.

## Relational Model Constraints

Constraints help implement the business rules. The following six constraints are defined in a relational database model:

1. *entity integrity constraint* - the primary key cannot have an unknown value
2. *referential integrity constraint* - defines relationships between tables and ensures that these relationships remain valid. The validity of the data is enforced using a combination of primary keys and foreign keys.
3. *semantic integrity constraint* - refers to the correctness of the meaning of the data.
4. *domain constraint* - specifies the permissible values for a given attribute.
5. *null constraint* - specifies that attribute values cannot be null.
6. *check constraint* - enforces domain integrity by limiting the values that are accepted by an attribute.

## SQL Statements

* STATEMENTS
  + Create, insert, select, update, delete
* DDF (Data Definition Language) statements:
  + Define, change or drop data
* DML (Data Manipulation Language) statements:
  + Read and modify data
* *Predicates* are used in the search condition of the WHERE clause. A predicate is conditioned evaluates to true, false or unknown.
* *Result Set* or a *result table* is the output we get from executing this query.

### CREATE TABLE

|  |  |  |
| --- | --- | --- |
| **create table TABLENAME (**  **COLUMN1 datatype,**  **COLUMN2 datatype,**  **COLUMN3 datatype);** | **create table TEST (**  **ID integer,**  **NAME varchar(30));** | **create table COUNTRY (**  **ID int,**  **CCODE char(2),**  **NAME varchar(2));** |
|  |  |  |
| **create table COUNTRY (**  **ID int NOT NULL,**  **CCODE char(2),**  **NAME varchar(60),**  **PRIMARY KEY (ID));** | **CREATE TABLE Persons (**  **ID int NOT NULL PRIMARY KEY,**  **LastName varchar(255) NOT NULL,**  **FirstName varchar(255),**  **Age int);** | ****DROP table:****  ****drop** table COUNTRY;**  **create table COUNTRY (**  **ID integer PRIMARY KEY NOT NULL,**  **CCODE char(2),**  **NAME varchar(60));** |

### SELECT Statement

**SELECT \* FROM [tablename] - DML**

**SELECT book\_id, title FROM book WHERE book\_id=’B1’ - operators: = > < >= <= <>**

**select COLUMN1, COLUMN2, ... from TABLE1 ;**

**select ID, NAME from COUNTRY ;**

**select \* from COUNTRY ;**

**select \* from COUNTRY where ID < 5 ;**

**select \* from COUNTRY where CCODE = 'CA';**

**COUNT()** is a built in database function that retrieves the number of rows that match the query criteria.

**select COUNT(\*) from tablename**

**select COUNT(COUNTRY) from MEDALS where COUNTRY='CANADA'**

**DISTINCT** is used to remove duplicate values from a result set.

**select DISTINCT columnname from tablename**

**select DISTINCT COUNTRY from MEDALS where MEDALTYPE = 'GOLD'**

**LIMIT** is used for restricting the number of rows retrieved from the database.

**select \* from tablename LIMIT 10**

**select \* from MEDALS where YEAR = 2018 LIMIT 5**

### INSERT Statement

**INSERT INTO [tablename] (---) VALUES (‘---‘),(‘---‘)**

### UPDATE and DELETE Statements

**UPDATE [tablename] SET [[ColumnName]=[Value]] <WHERE [Condition]>**

When using the UPDATE statement, if you do not specify the WHERE clause, all the rows in the table are updated.

**DELETE FROM [tablename] <WHERE [Condition]>**

When using the DELETE statement, if you do not specify the WHERE clause, all the rows in the table will be removed.

## Using String Patterns, Ranges, Sort, Grouping

**--run Customer.sql script to create & populate Customer table**

**select \* from customer;**

**select \* from customer where firstname LIKE 'S%'; -- string values**

**select \* from customer where homephone between '(800%' and '(820%'; -- ranges**

**select \* from customer where state in ('FL','ND'); -- set of values**

**select \* from customer ORDER BY firstname; -- sort**

**select \* from customer ORDER BY 2; -- sort by column 2**

**select \* from customer ORDER BY firstname DESC; -- sort, descending**

**select distinct(state) from customer; -- unique**

**select state, count(state) from customer group by state; -- count, group by**

**select state, count(state) as Count from customer group by state; -- coln name = count**

**select state, count(state) as Count from customer**

**group by state having count(state) >=2 ; -- having works with group by**

## Sub-Queries and Nested Selects

Sub-queries or sub-selects are like regular queries but placed within parenthesis and nested inside another query. This allows you to form more powerful queries than would have been otherwise possible.

**Option 1: will error:**

**select \* from employees where salary > avg(salary)**

**Option 2: will pass:**

**select \* from employees where salary < (select AVG(salary) from employees);**

**select EMP\_ID, F\_NAME, L\_NAME, DEP\_ID from employees**

**where DEP\_ID IN**

**(select DEP\_ID from employees where DEP\_ID >**

**(select MIN(DEP\_ID) from employees));**

**select EMP\_ID, SALARY, (select AVG(SALARY) from employees)**

**AS AVG\_SALARY from employees;**

**select \* from (select EMP\_ID, F\_NAME, L\_NAME, DEP\_ID from employees);**

## Working with Multiple Tables

### Sub-Queries

**-- sub-queries**

**select \* from employees where DEP\_ID IN (select DEPT\_ID\_DEP from departments);**

**select \* from employees where DEP\_ID IN (select DEPT\_ID\_DEP from departments where LOC\_ID = 'L0002');**

**select DEPT\_ID\_DEP, DEP\_NAME from departments where DEPT\_ID\_DEP IN**

**(select DEP\_ID from employees where SALARY > 70000 );**

### Implicit Join

**-- implicit join**

**select \* from employees, departments;**

**select \* from employees, departments where employees.DEP\_ID = departments.DEPT\_ID\_DEP;**

**select E.EMP\_ID, D.DEPT\_ID\_DEP from employees E, departments D where E.DEP\_ID = D.DEPT\_ID\_DEP;**

**select E.EMP\_ID, D.DEP\_NAME from employees E, departments D where E.DEP\_ID = D.DEPT\_ID\_DEP;**

## Access Databases using Python

Python uses database API (DB API) to access databases. The two main concepts in the Python DB-API are connection objects and query objects:

* *Connection objects* connect to a database and manage transactions.
* *Cursor objects* are used to run queries.
* **cursor()** method returns a new cursor object using the connection.
* **commit()** method is used to commit any pending transaction to the database.
* **rollback()** method causes the database to roll back to the start of any pending transaction.
* **close()** method is used to close a database connection.

|  |  |
| --- | --- |
| Application or Database | SQL API |
| MySQL | MySQL C API |
| PostgreSQL | psycopg2 |
| IBM DB2 | ibm\_db |
| SQL Server | dblib API |
| DB access for MS OS | ODBC |
| Oracle | OCI |
| Java | JDBC |

APIs used by popular SQL-based DBMS systems.

### Writing code using DB\_API

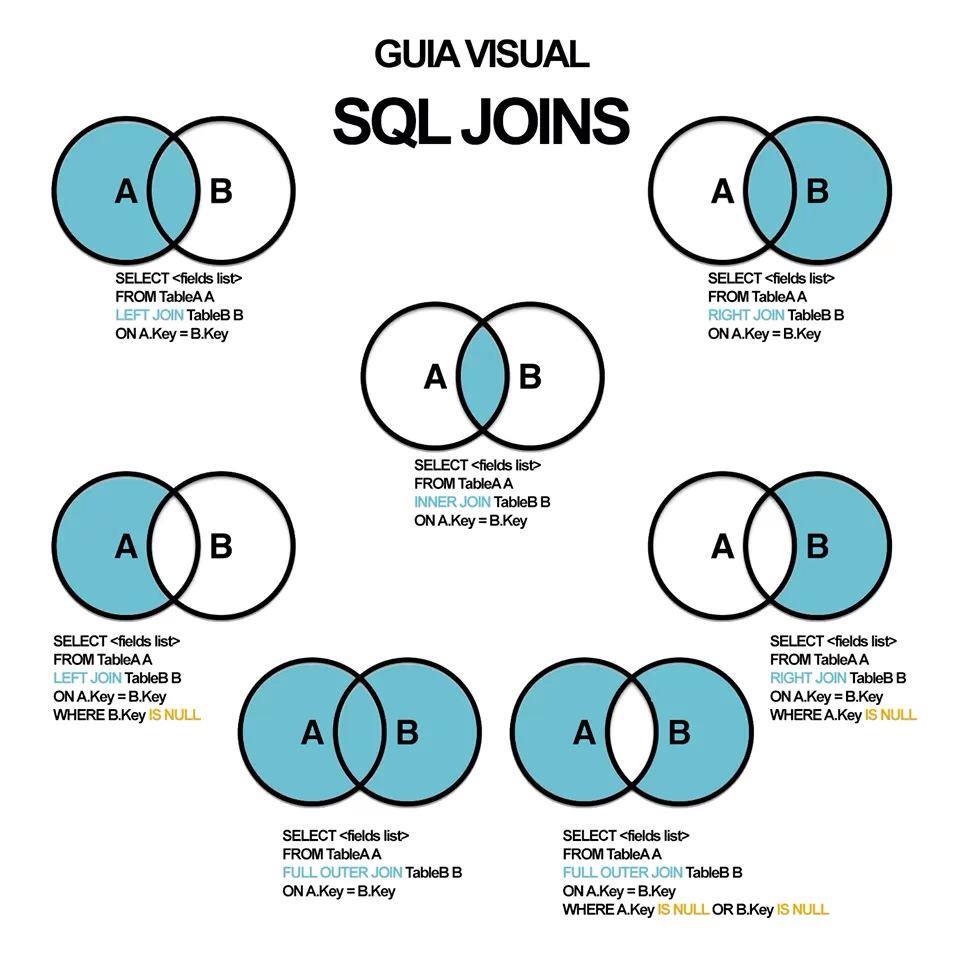
1. **from dbmodule import connect # import DB API**
2. **connection = connect('db\_name','username','password') # connection object**
3. **cursor = connection.cursor() # cursor object**
4. **cursor.execute('select \* from mytable') # queries**
5. **results = cursor.fetchall()**
6. **cursor.close() # close connection**
7. **connection.close() # close connection**

## JOIN Overview

Working with multiple tables.

Primary Key uniquely identifies each row in a table.

Foreign Key refers to a primary key of another table.



### INNER JOIN Operator

Returns only the rows that match in the tables.

**select B.borrower\_id, B.lastname, B.country, L.borrower\_id, L.loan\_date from borrower B**

**INNER JOIN loan L ON B.borrower\_id = L.borrower\_id**

**select B.lastname, L.copy\_id, C.status from borrower B**

**INNER JOIN loan L ON L.borrower\_id = B.borrower\_id**

**INNER JOIN copy C ON C.copy \_id = L.copy\_id**

### OUTER JOIN Operator

*LEFT JOIN*: ALL rows from the LEFT table and any MATCHING rows from the RIGHT table. The first table in the “**FROM**” clause is the left table. Any values not in the right table will display as NULL.

**select B.borrower\_id, B.lastname, B.country, L.borrower\_id, L.loan\_date from borrower B**

****LEFT JOIN** loan L ON L.borrower\_id = B.borrower\_id**

*RIGHT JOIN*: ALL rows from the RIGHT table and any MATCHING rows from the LEFT table. The first table in the “**FROM**” clause is the left table.

**select B.borrower\_id, B.lastname, B.country, L.borrower\_id, L.loan\_date from borrower B**

****RIGHT JOIN** loan L ON L.borrower\_id = B.borrower\_id**

*FULL JOIN*: ALL rows from the LEFT and ALL rows from the RIGHT table. Any values not in the right table will display as NULL.

**select B.borrower\_id, B.lastname, B.country, L.borrower\_id, L.loan\_date from borrower B**

****FULL JOIN** loan L ON L.borrower\_id = B.borrower\_id**

# Data Analysis with Python

Data analysis and data science, helps us unlock the information and insights from raw data to answer our questions. Data analysis plays an important role by helping us to discover useful information from the data (data does not mean information.), answer questions, and even predict the future or the unknown.

## Understanding the Data

### Data Frame Data

**Dataframe: data under analysis, in rows and columns. Each column is a series.**

**Attribute: column name**

**Feature: column data**

**Observation: row data**

**Target: what we want to predict**

### Categories of Data

**CATEGORICAL:**

**Nominal variables that have two or more categories, but which do not have an intrinsic order. Embarked-port name**

**Dichotomous nominal variable with only two categories. Sex**

**Ordinal variables that have two or more categories just like nominal variables. Only the categories can also be ordered or ranked. Passenger Class**

**NUMERIC:**

**Discrete: PassengerID, SiblingSpouseCount, Survived**

**Continuous: Age, Fare**

**TEXT: Ticket Number (alphanumeric), Passenger Name**

## Python Packages for Data Science

Python library is a collection of functions and methods that allow you to perform lots of actions without writing any code. The libraries usually contain built in modules providing different functionalities which you can use directly.

**Python** **Scientific Computing Libraries**:

* *Pandas* has many functions for data importing, manipulation and analysis, data structures and operations for manipulating numerical tables and timeseries. Primary instrument of Pandas is the two dimensional table *data frame*.
* *NumPy* library uses arrays for its inputs and outputs and is a math library to work with N-dimensional arrays
* *SciPy* is a collection of numerical algorithms and domain specific toolboxes, including signal processing, optimization, statistics and scientific and high performance computation

**Python** **Visualization Libraries**:

* *Matplotlib* is a very popular plotting package that provides highly customizable 2D/3D graphs and plots.
* *Seaborn* is based on Matplotlib and can be used to generate various plots such as heat maps, time series violin plots.

**Python** **Algorithmic Libraries**:

* *Scikit-learn* library contains tools for statistical modeling, including regression, classification, clustering and so on. This library is built on NumPy, SciPy and Matplotib. Scikit Learn implements machine learning pipelines including pre-processing of data, feature selection, feature extraction, train test splitting, defining the algorithms, fitting models, tuning parameters, prediction, evaluation and exporting the model.
* *Statsmodels* allows users to explore data, estimate statistical models and perform statistical tests.

## Importing and Exporting Data

Format (csv, json, xlsx, hdf) and file-path.

**import pandas as pd**

**url = “https:// … “**

**df = pd.read\_csv(url, header = None) # no headers (=None)**

**headers = [“h1”, “h2”, …] # adding headers**

**df.columns = headers # add headers to dataframe**

**path = “c:\Windows\filename.csv” # assign path and filename**

**df.to\_csv(path) # save to path\filename**

| **Data Format** | **Read** | **Save** |
| --- | --- | --- |
| **csv** | **pd.read\_csv()** | **df.to\_csv()** |
| **json** | **pd.read\_json()** | **df.to\_json()** |
| **excel** | **pd.read\_excel()** | **df.to\_excel()** |
| **hdf** | **pd.read\_hdf()** | **df.to\_hdf()** |
| **sql** | **pd.read\_sql()** | **df.to\_sql()** |
| **...** | **...** | **...** |

## Analyzing Data

Pandas **object** is **string** type (numeric or strings)

Data types should be checked for potential info, type mismatch and compatibility with python methods.

**df.dtypes # display data-types**

**df.describe() # statistical summary (skips non-numeric columns)**

**df.describe(include=’all’) # statistical summary of all columns**

**# non-numeric columns will have unique, top, frequency, NaN**

**unique is number of distinct objects in column**

**top is most frequently occurring object**

**freq is number of times top object appears in column**

**NaN which stands for “not a number”**

**df.info() # display the top 30 and bottom 30 rows of the data frame**

## Pre-processing Data

*Data preprocessing* is the process of converting or mapping data from one raw form into another format to make it ready for further analysis. Data preprocessing is often called *data cleaning* or *data wrangling*

### Missing Values

When no data value is stored for feature for a particular observation, we say this feature has a missing value. Usually missing value in data set appears as “?”, “N/A”, “NaN” or just a blank cell. Missing data can be dealt with as follows:

* Check with data collection source
* Drop the missing value
  + Drop the variable
  + Drop the single data entries
* Replace the missing data
  + Replace with average value (mean)
  + Replace it by frequency (mode)
  + Replace based on other functions
* Leave it as missing data

*Dropping missing values:*

**df.dropna()**

**df.dropna(axis=0) # 0 for rows, =1 for columns)**

**df.dropna(subset=[‘price’], axis=0, inplace=True) # inplace; also update dataframe**

**df = df.dropna(subset=[‘price’], axis=0) # same as above code**

**df.dropna(subset=[‘price’], axis=0 # does not update dataframe**

**# simply drop whole row with NaN in "price" column**

**df.dropna(subset=["price"], axis=0, inplace=True)**

**# reset index, because we droped two rows**

**df.reset\_index(drop=True, inplace=True)**

*Replace missing values:*

**df.replace(<missing\_value>,<new\_value>)**

**df[‘normalized-losses’].replace(np.nan, df[‘normalized-losses’].mean())**

### Data Formatting/Standardization

*Data formatting* or *standardization* is the process of transforming data into a common format which allows the researcher to make meaningful comparisons. As a part of dataset cleaning, data formatting ensures the data is consistent and easily understandable, i.e all NY, New York, N.Y. changed to New York, or mpg to kpg

***Convert all “city-mpg” to “city-L/100km”:***

**df[‘city-mpg’] = 235/df[‘city-mpg]**

**df.rename(columns={‘city-mpg’:’city-L/100km’}, inplace=True)**

***Change data type to correct format:***

**df[‘price’] = df[‘price’].astype(‘int’) # cast to as int**

**df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float") # cast to as float**

### Data Normalization

Data normalization is the process of making feature values with different ranges uniform in order to perform statistical analysis and modeling. For example, correlation will be significantly biased toward income with respect to age in a non-normalized dataset. To avoid this, we can *normalize* these two variables into values with a uniform range, i.e from zero to one, from -3 to +3, etc.

|  |  |  |
| --- | --- | --- |
| **Simple Feature Scaling** |  | **df[‘length’] = df[‘length’]/ df[‘length’].max()**  *range: 0 to 1* |
| **Min-Max** |  | **df[‘length’] = (df[‘length’] - df[‘length’].min()) / (df[‘length’].max() - df[‘length’].min())** |
| **Z-Score** | µ - mean  σ – standard deviation | **df[‘length’] = (df[‘length’] - df[‘length’].mean()) / df[‘length’].std()**  *range: -3 to +3* |

### Binning

*Binning* is a process of transforming continuous numerical variables into discrete categorical 'bins', for grouped analysis. For example, you can bin “age” into [0 to 5], [6 to 10], [11 to 15] and so on. Sometimes, binning can improve accuracy of the predictive models. In addition, sometimes we use data binning to group a set of numerical values into a smaller number of bins to have a better understanding of the data distribution.

Binning:

**bins = np.linspace(min(df[‘price’], max(df[‘price’]), 4) # create 4 bins**

**group\_names = [“Low”, “Medium”, “High”]**

**df[‘price binned’] = pd.cut(df[‘price’], bins, labels = group\_names, include\_lowest=True)**

### Turn Categorical Variables into Quantitative Variables

Most statistical models cannot take in objects or strings as input and for model training only take the numbers as inputs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Car | Fuel | … | gas | diesel |
| A | gas | … | 1 | 0 |
| B | diesel | … | 0 | 1 |
| C | gas | … | 1 | 0 |
| D | gas | … | 1 | 0 |

**One-hot Encoding:**

* Add dummy variables for each unique category
* Assign 0 or 1 in each category
* pd.get\_dummies(df[‘fuel’])

## Exploratory Data Analysis (EDA)

EDA is an approach to analyze data in:

* order to summarize main characteristics of the data
* gain better understanding of the data set
* uncover relationships between different variables
* extract important variables for the problem we're trying to solve

The main question we are trying to answer in this module is,

* what are the characteristics that had the most impact on the target

### Descriptive Statistics

Descriptive statistical analysis helps to describe basic features of a dataset and obtains a short summary about the sample and measures of the data.

**describe() # statistical summary**

**value\_counts() # summarizes categorical data**

**boxplot # distribution of data**

**scatterplot # relationship between two variables**

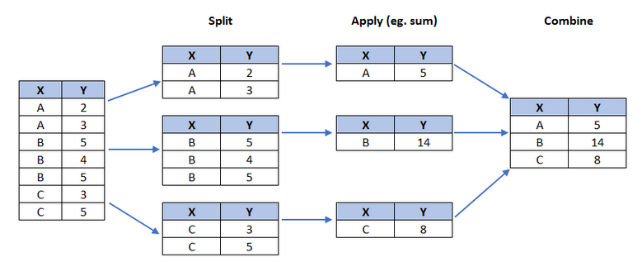
**df.describe(include=['object']) # unique, top freq of type object**

**df['drive-wheels'].value\_counts().to\_frame() # convert series to data frame**

### GroupBy, Pivot and Heatmap

The group by method is used on categorical variables, groups the data into subsets according to the different categories of that variable. You can group by a single variable or you can group by multiple variables by passing in multiple variable names.

|  |  |
| --- | --- |
| **groupby** | **df\_tst = df[[‘A’,’B’,’C’]]**  **df\_grp =df\_tst.groupby([‘A’,’B’,as\_index=False).mean()** |
| **pivot** | **df\_pvt = df.grp.pivot(index= ‘A’, columns = ‘B’)** |
| **heatmap** | **plt.pcolor(df\_pvt, cmap=’RdBu’)**  **plt.colorbar()**  **plt.show()** |

The general process of groupby involves the following steps:

1. **Split:** Splitting the data into groups based on some criteria.
2. **Apply:** Applying a function to each group independently:

**.sum()**

**.count()**

**.mean()**

**.std()**

**.aggregate()**

**.apply()**

**.etc..**

1. **Combine:** Combining the results into a data structure.

### Correlation - Statistics

Correlation is a statistical metric for measuring to what extent different variables are interdependent. In other words, when we look at two variables over time, if one variable changes, how does this affect change in the other variable? Correlation does not imply causation.

**Pearson Correlation**

Measures the strength of the correlation between two features:

* Correlation Coefficient:
* close to +1: large positive relationship
* close to -1: large negative relationship
* close to 0: no relationship
* P-Value:
* p-value < 0.001: strong certainty in the result
* p-value < 0.05: moderate certainty in the result
* p-value < 0.1: weak certainty in the result
* p-value > 0.1: no certainty in the result

**stats.pearsonr(df['horsepower'], df['price']) #** **(0.8095, 6.3690e-48)**

**Pearson Coefficient: 0.8095 # strong correlation**

**P-Value: 6.3690e-48 # high certainty in result**

**pearson\_coef, p\_value = stats.pearsonr(df['wheel-base'], df['price']) # (0.5846, 8.0764e-20)**

**pearson\_coef: 0.5846 # some correlation**

**p\_value: 6.0764e-20 # high certainty in result**

### Analysis of Variance (ANOVA)

ANOVA is used to find the correlation between different groups of a *categorical* variable, it is the statistical comparison of groups. ANOVA test returns two values, the F-test score and the p-value.

* *F-test score* ANOVA assumes the means of all groups are the same, calculates how much the actual means deviate from the assumption, and reports it as the F-test score. A larger score means there is a larger difference between the means. Example: Small between Honda and Subaru, large between Honda and Jaguar
* *p-value* shows whether the result is statistically significant, confidence degree

**anova\_grp = df[['make','price']].groupby(['make'])**

**stats.f\_oneway(anova\_grp.get\_group('honda')['price'], anova\_grp.get\_group('subaru')['price'])**

**# F\_onewayResult(statistic=0.1974, pvalue=0.6609)**

**F-test score: 0.1974 # prices not significantly different**

**P-Value: 0.6604 # moderate/weak certainty**

**stats.f\_oneway(anova\_grp.get\_group('honda')['price'], anova\_grp.get\_group('jaguar')['price'])**

**F\_onewayResult(statistic=400.9258, pvalue=1.0586e-11)**

**# ANOVA**

**f\_val, p\_val = stats.f\_oneway(grouped\_test2.get\_group('fwd')['price'], grouped\_test2.get\_group('rwd')['price'], grouped\_test2.get\_group('4wd')['price'])**

**print( "ANOVA results: F=", f\_val, ", P =", p\_val)**

**ANOVA results: F= 67.95406500780399 , P = 3.3945443577151245e-23**

## Model Development

See Section 8.2

### Simple Linear Regression (SLR)

### Multiple Linear Regression (MLR)

**Multiple Linear Regression** is used to explain the relationship between one continuous response (dependent) variable and **two or more** predictor (independent) variables. Most of the real-world regression models involve multiple predictors. multiple linear regression is the extension of the simple linear regression model. We will illustrate the structure by using four predictor variables, but these results can generalize to any integer:

𝑌:𝑅𝑒𝑠𝑝𝑜𝑛𝑠𝑒 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒

𝑋1:𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑜𝑟 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 1

𝑋2:𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑜𝑟 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 2

𝑋3:𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑜𝑟 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 3

𝑋4:𝑃𝑟𝑒𝑑𝑖𝑐𝑡𝑜𝑟 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 4

𝑎:𝑖𝑛𝑡𝑒𝑟𝑐𝑒𝑝𝑡

𝑏1:𝑐𝑜𝑒𝑓𝑓𝑖𝑐𝑖𝑒𝑛𝑡𝑠 𝑜𝑓 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 1

𝑏2:𝑐𝑜𝑒𝑓𝑓𝑖𝑐𝑖𝑒𝑛𝑡𝑠 𝑜𝑓 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 2

𝑏3:𝑐𝑜𝑒𝑓𝑓𝑖𝑐𝑖𝑒𝑛𝑡𝑠 𝑜𝑓 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 3

𝑏4:𝑐𝑜𝑒𝑓𝑓𝑖𝑐𝑖𝑒𝑛𝑡𝑠 𝑜𝑓 𝑉𝑎𝑟𝑖𝑎𝑏𝑙𝑒 4

The equation is given by: 𝑌ℎ𝑎𝑡=𝑎+𝑏1𝑋1+𝑏2𝑋2+𝑏3𝑋3+𝑏4𝑋4

**Z = df[['horsepower','curb-weight','engine-size','highway-mpg']]**

**lm.fit(Z,df['price'])**

**Yhat = lm.predict(Z)**

**print("Intercept (lm.intercept\_): {}\nSlope (lm.coef\_): {}".format(lm.intercept\_,lm.coef\_))**

**Intercept (lm.intercept\_): -15806.624626329223**

**Slope (lm.coef\_): [53.49574423 4.70770099 81.53026382 36.05748882]**

**print("PRICE = {} + \n\t{} x HIGHWAY=MPG + \n\t{} x CURB-WEIGHT + \n\t{} x ENGINE-SIZE + \n\t{} x HIGHWAY-MPG".format(lm.intercept\_,lm.coef\_[0],lm.coef\_[1],lm.coef\_[2],lm.coef\_[3]))**

**PRICE = -15806.624626329223 +**

**53.49574422604911 x HIGHWAY=MPG +**

**4.707700994613939 x CURB-WEIGHT +**

**81.53026382122765 x ENGINE-SIZE +**

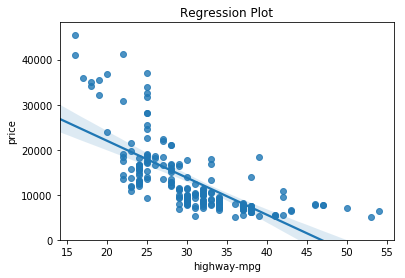
**36.057488816485964 x HIGHWAY-MPG**

## Model Evaluation using Visualization

In-sample evaluation tells us how well our model fits the data already given to train it. It does not give us an estimate of how well the train model can predict new data.

### Regression Plot

Regression plots are a good estimate of the relationship between two variables, the strength of the correlation, and the direction of the relationship (positive or negative). The horizontal axis is the independent variable. The vertical axis is the dependent variable. Each point represents a different target point. The fitted line represents the predicted value.



This plot will show a combination of a scattered data points (a **scatter plot**), as well as the fitted **linear regression** line going through the data. This will give us a reasonable estimate of the relationship between the two variables, the strength of the correlation, as well as the direction (positive or negative correlation).

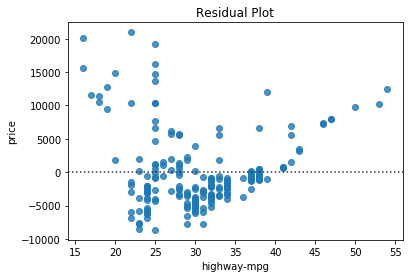
**sns.regplot(data=df, x='highway-mpg', y='price')**

**plt.title('Regression Plot')**

**plt.ylim(0,)**

### Residual Plot

The residual plot represents the error between the actual value. Examining the predicted value and actual value we see a difference. We obtain that value by subtracting the predicted value, and the actual target value. We then plot that value on the vertical axis with the dependent variable as the horizontal axis.

Looking at the plot gives us some insight into our data:

* Results have zero mean, distributed evenly around the x axis with similar variance. There is *no curvature*. This type of residual plot suggests a *linear plot* is appropriate.
* Results have non-zero mean, distributed randomly around the x axis. There *is curvature*. This type of residual plot suggests a *non-linear plot* is appropriate.

**sns.residplot(df['highway-mpg'], df['price'])**

**plt.title('Residual Plot')**

The first parameter is a series of dependent variable or feature. The second parameter is a series of dependent variable or target.

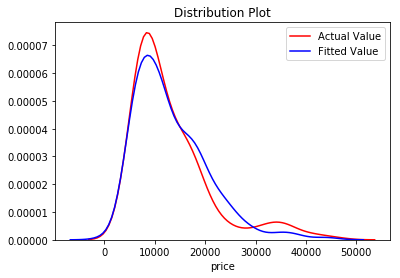
### Distribution Plots

A distribution plot counts the predicted value versus the actual value. These plots are extremely useful for visualizing models with more than one independent variable or feature.

**ax1 = sns.distplot(df['price'], hist=False, color='r',label='Actual Value')**

**sns.distplot(Yhat, hist=False, color='b',label='Fitted Value',ax=ax1)**

**plt.title('Distribution Plot')**



## Polynomial Regression and Pipelines

### Polynomial Regression

Polynomial Regression is a special case of general SLR/MLR models and is useful for describing curvilinear[[1]](#footnote-1) relationships.

What is a curvilinear relationship? It's what you get by squaring or setting higher-order terms of the predictor variables in the model transforming the data. The model can be quadratic (second-order polynomial), cubic (third-order polynomial) higher order polynomial regressions when a good fit hasn't been achieved by second or third-order.

### Pipelines

**Pipeline** is just an abstract notion, it's not some existing ml algorithm. Often in ML tasks you need to perform sequence of different transformations (find set of features, generate new features, select only some good features) of raw dataset before applying final estimator. Following is a good example of Pipeline usage. Pipeline gives you a single interface for all 3 steps of transformation and resulting estimator. It encapsulates transformers and predictors inside, and now you can do something like:

**pipeline = Pipeline([**

**('vect', CountVectorizer()),**

**('tfidf', TfidfTransformer()),**

**('clf', SGDClassifier()),**

**])**

**predicted = pipeline.fit(Xtrain).predict(Xtrain)**

**# Now evaluate all steps on test set**

**predicted = pipeline.predict(Xtest)**

### Mean Square Error and R-Squared

Measures of in-sample evaluation are a way to numerically determine how good the model fits on our data. Two important measures that we often use to determine the fit of a model are: *Mean Square Error* (MSE), and *R-squared*.

* **MSE** finds the difference between the actual value y and the predicted value yhat then square it. We then square the number. We then take the Mean or average of all the errors by adding then all together and dividing by the number of samples.
* **R-squared** (coefficient of determination) is a measure to determine how close the data is to the fitted regression line. For the most part, it takes values between 0 (bad fit) and 1 (good fit).

**x = df[['highway-mpg']]**

**y = df[['price']]**

**lm.fit(x,y)**

**print("R-Squared: {}".format(lm.score(x,y)))**

**R-Squared: 0.4965911884339176**

* When comparing models, **the model with the smallest MSE value is a better fit** for the data. R-Squared for SLRs is typically smaller than for MLRs and Polynomials.
* When comparing models, **the model with the higher R-squared value is a better fit** for the data. Higher MSE implies more error present. MSE for SLRs are greater than for MLRs and Polynomials because they have more variables in the model.

### Prediction and Decision Making

The first thing you should do is make sure your model results make sense. You should always use

* visualization,
* numerical measures for evaluation and
* comparing between different models.

**lm.fit(df[['highway-mpg']],df[['price']])**

**lm.predict([[30]]) # need double brackets**

**array([[13771.3045085]])**

Sometimes your model will produce values that don't make sense. For example, if we plot the model out for highway miles per gallon in the ranges of 0 to 100 we get negative values for the price. This could be because the values in that range are not realistic. The linear assumption is incorrect or we don't have data for cars in that range. In this case, it is unlikely that a car will have fuel mileage in that range. So our model seems valid.

Use values for x and see the predicted values. Use regplot, residual and distribution plot. Best indicator is MSE and R-Squared, as the points get further from the line, the MSE increases (not good)

## Model Evaluation and Refinement

Model evaluation tells us how our model performs in the real world. In-sample evaluation tells us how well our model fits the data already given to train it. It does not give us an estimate of how well the train model can predict new data.

The solution is to split our data up, use the in-sample data or training data to train the model. The rest of the data, called Test Data, is used as out-of-sample data. This data is then used to approximate how the model performs in the real world. Separating data into training and testing sets is an important part of model evaluation.

We use the test data to get an idea how our model will perform in the real world. We use training set (70%) to build a model and discover predictive relationships. We then use a testing set (30%) to evaluate model performance. When we have completed testing our model, we should use all the data to train the model.

### Train/Test Split Function

A popular function in the scikit-learn package for splitting datasets is the train test split function. This function randomly splits a dataset into training and testing subsets.

**from sklearn.model\_selection import train\_test\_split**

**x\_train, x\_test, y\_train, y\_test = train\_test\_split(x\_data, y\_data, test\_size=0.3, random\_state=0)**

* + **x\_data features or independent variables**
  + **y\_data dataset target**
  + **x\_train, y\_train training set**
  + **x\_test, y\_test testing set**
  + **test\_size testing data set (30%)**
  + **random\_state random seed**

*Generalization error* is a measure of how well our data does at predicting previously unseen data. The error we obtain using our testing data is an approximation of this error.

### Cross Validation Score and Predict

*Cross Validation Score* splits the dataset into K equal groups. Each group is referred to as a fold. Some of the folds can be used as a training set which we use to train the model and the remaining parts are used as a test set, which we use to test the model. For example, we can use three folds for training, then use one fold for testing (four fold). This is repeated until each partition is used for both training and testing. At the end, we use the average results as the estimate of out-of-sample error.

**from sklearn.model\_selection import cross\_val\_score**

**score = cross\_val\_score(model, x\_data, y\_data, cv=3)**

**np.mean(scores) # mean of scores**

* + **model regression model, example: LinearRegression**
  + **x\_data predictor data**
  + **y\_data target data**
  + **cv=3 partitions, folds. 2 for training, 1 for testing**

**Array of 3 values returned**

*Cross Validation Predict* returns predicted y values for the test fold.

**from sklearn.model\_selection import cross\_val\_predict**

**yhat = cross\_val\_predict(model, x\_data, y\_data, cv=3)**

### Over-fitting, Under-fitting and Model Selection

In this section, we will discuss how to pick the best polynomial order and problems that arise when selecting the wrong order polynomial.

The goal of Model Selection is to determine the order of the polynomial to provide the best estimate of the function y(x). If we try and fit the function with a linear function, the line is not complex enough to fit the data. As a result, there are many errors. This is called *underfitting*, where the model is too simple to fit the data.

If we increase the order of the polynomial, the model fits better, but the model is still not flexible enough and exhibits underfitting. *Overfitting* is where the model is too flexible and fits the noise rather than the function.

Possible cause of errors:

* Noise, cannot be predicted and is called irreducible error
* Polynomial assumption can be wrong
* Sample may have come from a different function
* Model may be too difficult to fit
* Correct type of data to estimate the function may not be available

Plot R-Squared (y-axis) with polynomial orders on the x-axis

**Rsquare \_test = []**

**Order = [1,2,3,4]**

**for n in order:**

**pr = PolynomialFeatures(degree = n)**

**x\_train\_pr = pr.fit\_transform(x\_train[[‘horsepower’]])**

**x\_test\_pr = pr.fit\_transform(x\_test[[‘horsepower’]])**

**lr.fit(x\_train\_pr,y\_train)**

**Rsquare\_test.append(lr.score(x\_test\_pr,y\_test)**

### Ridge Regression

Ridge regression prevents overfitting. In many cases real data has outliers. If we use a tenth order polynomial function to fit the data, the estimated function in blue is incorrect, and is not a good estimate of the actual function. If we examine the expression for the estimated function, we see the estimated polynomial coefficients have a very large magnitude. This is especially evident for the higher order polynomials. Ridge regression controls the magnitude of these polynomial coefficients by introducing the parameter *alpha*. Alpha is a hyperparameter we select before fitting or training the model.

Alpha must be selected carefully. If alpha is too large, the coefficients will approach zero and underfit the data. If alpha is zero, the overfitting is evident. In order to select alpha, we use *cross validation*.

**from sklearn.linear\_model import Ridge**

**RigeModel = Ridge(alpha = 0.1)**

**RigeModel.fit(X,y)**

**Yhat = RigeModel.predict(X)**

In order to determine the parameter alpha, we use some data for training. We use a second set called validation data. This is similar to test data, but it is used to select parameters like alpha.

1. start with a small value of alpha.
2. train the model, make a prediction using the validation data, then calculate the R-squared and store the values.
3. Repeat the value for a larger value of alpha.
4. train the model again, make a prediction using the validation data, then calculate the R-squared and store the values of R-squared.
5. repeat the process for a different alpha value, training the model, and making a prediction.
6. select the value of alpha that maximizes the R-squared.
7. Note that we can use other metrics to select the value of alpha, like mean squared error.

### Grid Search

Grid Search is a method in Scikit-learn that automatically iterates over hyperparameters (like alpha) using cross-validation. Grid Search takes the model or objects you would like to train and different values of the hyperparameters. It then calculates the mean square error or R-squared for various hyperparameter values, allowing you to choose the best values.

GridSearch takes Ridge(), Scoring, Number of Folds and Alpha: OUT: Alpha, R-Square:

**from sklearn.linear\_model import Ridge**

**from sklearn.model\_selection import GridSearchCV**

**parameters1 = [{‘alpha’:[1,10,100,1000,10000,100000],**

**‘normalize’:[True, False]}]] # dictionary**

**RR = Ridge()**

**Grid1 = GridSearchCV(RR, parameters1, cv = 4)**

**Grid1.fit(x\_data[[‘horsepoer’,’curb-weight’,’engine-size’]], y\_data)**

**Grid1.best\_estimator\_**

**scores = Grid1.cv\_results\_**

**scores[‘mean\_test\_score’]**

1. start off with one value for hyperparameters and train the model.
2. use different hyperparameters to train the model.
3. continue the process until we have exhausted the different free parameter values.
4. Each model produces an error.
5. select the hyperparameter that minimizes the error.
6. To select the hyperparameter, we split our dataset into three parts, the training set, validation set, and test set.
7. train the model for different hyperparameters.
8. use the R-squared or mean square error for each model.
9. select the hyperparameter that minimizes the mean squared error or maximizes the R-squared on the validation set.
10. finally test our model performance using the test data.

# Data Visualization with Python

Less is more effective, less is more attractive, less is more impactive.

## Matplotlib - Introduction

John Hunter (1968-2012)

Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shell, the jupyter notebook, web application servers, and four graphical user interface toolkits.

One of the core aspects of Matplotlib is **matplotlib.pyplot**. It is Matplotlib's scripting layer which we studied in details in the videos about Matplotlib. Recall that it is a collection of command style functions that make Matplotlib work like MATLAB. Each **pyplot** function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.

Plotting in pandas[¶](https://e-4baaecaedd.cognitiveclass.ai/lab#Plotting-in-%3Cem%3Epandas%3C%2Fem%3E): pandas has a built-in implementation of Matplotlib that we can use. Plotting in pandas is as simple as appending a **.plot()** method to a series or dataframe.

## Matplotlib - Architecture Layers

**Backend Layer**

* *FigureCanvas*: matplotlib.backend\_bases.FigureCanvas

Encapsulates the concept of a surface to draw onto (e.g. "the paper")

* *Renderer*: matplotlib.backend\_bases.Renderer

Does the drawing (e.g. "the paintbrush")

* *Event*: matplotlib.backend\_bases.Event

Handles user inputs such as keyboard strokes and mouse clicks

**Artist Layer** -Object oriented method (axes instance from matplotlib)

* Comprised of one main object - *Artist*

Knows how to use the Renderer to draw on the canvas

* Tiles, lines, tick labels and images all correspond to individual Artist instance
* Two types of Artist objects:
  1. **Primitive:** Line2D, Rectangle, Circle and Text
  2. **Composite:** Axis, Ticks, Axes and Figure
* Each *composite* artist may contain other *composite* artists as well as *primitive* artist

**Scripting Layer -** *Procedural Method* **(**matplotlib.pyplot as 'plt')

* Comprised mainly of pyplot, a scripting interface that is lighter than the Artist layer

### Scripting layer

You can use **plt** i.e. **matplotlib.pyplot** and add more elements by calling different methods procedurally; for example, **plt.title(...)** to add title or **plt.xlabel(...)** to add label to the x-axis. (procedural method)

**df\_top5.plot(kind='area', alpha=0.35, figsize=(10, 6))**

**plt.title('Immigration trend of top 5 countries')**

**plt.ylabel('Number of immigrants')**

**plt.xlabel('Years')**

**plt.show()**

### Artist layer

You can use an Axes instance of your current plot and store it in a variable (eg. **ax**). You can add more elements by calling methods with a little change in syntax (by adding "set\_" to the previous methods). For example, use **ax.set\_title()** instead of **plt.title()** to add title, or **ax.set\_xlabel()** instead of **plt.xlabel**() to add label to the x-axis. (Object oriented method)

**ax = df\_top5.plot(kind='area', alpha=0.35, figsize=(20, 10))**

**ax.set\_title('Immigration Trend of Top 5 Countries')**

**ax.set\_ylabel('Number of Immigrants')**

**ax.set\_xlabel('Years')**

This option sometimes is more transparent and flexible to use for advanced plots (in particular when having multiple plots).

## Matplotlib - Types of Plots

### Line Plots (Series/Dataframe)

A line chart or line plot is a type of plot which displays information as a series of data points called 'markers' connected by straight line segments. It is a basic type of chart common in many fields. Use line plot when you have a *continuous data set*. These are best suited for trend-based visualizations of *data over a period of time*.

### Area Plots - accumulated totals

An area plot also known as an area chart or graph is a type of plot that depicts accumulated totals using numbers or percentages over time. It is based on the line plot and is commonly used when trying to compare two or more quantities.

Area plots are stacked by default. And to produce a stacked area plot, each column must be either all positive or all negative values (any NaN values will defaulted to 0). To produce an unstacked plot, pass **stacked=False**

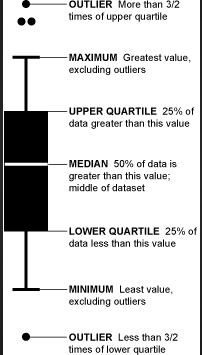
### Histogram – frequency distribution

A histogram is a way of representing the frequency distribution of a numeric dataset. The way it works is it partitions the spread of the numeric data into bins, assigns each datapoint in the dataset to a bin, and then counts the number of datapoints that have been assigned to each bin. So the vertical axis is actually the frequency or the number of datapoints in each bin

### Bar Chart – proportional representation

bar plot is a way of representing data where the length of the bars represents the magnitude/size of the feature/variable. Bar graphs usually represent numerical and categorical variables grouped in intervals. **kind= bar, kind=barh**

### Pie Chart

Pie chart is a circular statistical graphic divided into slices to illustrate numerical proportion.

### Box Plot

A box plot is a way of statistically representing the distribution of given data through five main dimensions. A **box plot** is a way of statistically representing the *distribution* of the data through five main dimensions:

* **MINIMUM:** Smallest number in the dataset.
* **LOWER/First quartile:** Middle number between the **minimum** and the **median**.
* **MEDIAN/Second quartile:** Middle number of the (sorted) dataset.
* **UPPER/Third quartile:** Middle number between **median** and **maximum**.
* **MAXIMUM:** Highest number in the dataset.
* **Inter Quartile Range (IQR) =** Upper Quartile – Lower Quartile
* **OUTLIERS:**
  + larger than Q3 by at least 1.5 times the interquartile range (IQR), or,
  + smaller than Q1 by at least 1.5 times the IQR.

### Scatter Plot

A **scatter plot (2D)** is a useful method of comparing variables against each other. **Scatter** plots look similar to **line plots** in that they both map independent and dependent variables on a 2D graph. While the datapoints are connected together by a line in a line plot, they are not connected in a scatter plot. The data in a scatter plot is considered to express a trend. With further analysis using tools like regression, we can mathematically calculate this relationship and use it to predict trends outside the dataset.

### Subplots

To visualize multiple plots together, we can create a **figure** (overall canvas) and divide it into **subplots**, each containing a plot. With subplots, we usually work with the *artist layer* *instead of the scripting layer*. Typical syntax is :

**fig = plt.figure() # create figure**

**ax = fig.add\_subplot(nrows, ncols, plot\_number) # create subplots**

Where

* **nrows** and **ncols** are used to notionally split the figure into (**nrows \* ncols**) sub-axes
* **plot\_number** starts at 1, increments across rows first and has a maximum of **nrows \* ncols**

### Bubble Plots

A **bubble plot** is a variation of the **scatter plot** that displays three dimensions of data (x, y, z). The datapoints are replaced with bubbles, and the size of the bubble is determined by the third variable 'z', also known as the weight. In **maplotlib**, we can pass in an array or scalar to the keyword s to **plot(),** that contains the weight of each point.

### Waffle Charts

A waffle chart is a great way to visualize data in relation to a whole or to highlight progress against a given threshold

### Word Cloud

A word cloud is simply a depiction of the importance of different words in the body of text. A word cloud works in a simple way; the more a specific word appears in a source of textual data the bigger and bolder it appears in the world cloud.

### Seaborn (based on matplotlib)

Seaborn is another data visualization library, it is actually based on Matplotlib. It was built primarily to provide a high-level interface for drawing attractive statistical graphics, such as regression plots, box plots, and so on. Seaborn makes creating plots very efficient.

# Machine Learning with Python

Machine learning is the subfield of computer science that gives "computers the ability to learn without being explicitly programmed.” It is inspired by the human learning process, iteratively learn from data, and allow computers to find hidden insights using statistical models without explicitly being programmed. These models help us in a variety of tasks, such as object recognition, summarization, recommendation, and so on.

## Introduction to Machine Learning

### Artificial Intelligence, Machine Learning and Deep Learning:

* *Artificial Intelligence*: AI tries to make computers intelligent in order to mimic the cognitive functions of humans. So, Artificial Intelligence is a general field with a broad scope including: Computer Vision, Language Processing, Creativity, and Summarization
* *Machine Learning*: Machine Learning is the branch of AI that covers the statistical part of artificial intelligence. It teaches the computer to solve problems by looking at hundreds or thousands of examples, learning from them, and then using that experience to solve the same problem in new situation
* *Deep Learning*: Deep Learning is a very special field of Machine Learning where computers can actually learn and make intelligent decisions on their own. Deep learning involves a deeper level of automation in comparison with most machine learning algorithms. Neural network

### Machine Learning Techniques:

* *Regression/Estimation*: predicting a continuous value, for example, predicting the price of a house based on its characteristics, or estimating the Co2 emission from a car’s engine
* *Classification*: predicting the class or category of a case, for example, if a cell is benign or malignant, or whether or not a customer will stay or leave
* *Clustering*: finding structure of data; summarization, groups of similar cases, for example, can find similar patients, or can be used for customer segmentation in the banking field
* *Association*: finding items/events that often co-occur, for example, grocery items that are usually bought together by a particular customer
* *Anomaly detection*: discover abnormal and unusual cases, for example, it is used for credit card fraud detection
* *Sequence mining*: predicting the next event, for instance, the click-stream in websites (Markov Model, HMM)
* *Dimension reduction*: reduce the size of data
* *Recommendation systems*: recommending items. Associates people's preferences with others who have similar tastes, and recommends new items to them, such as books or movies.

### Supervised vs Unsupervised Machine Learning

**Supervised knows basic rules, task driven**

**Unsupervised watch what others are doing, data driven**

**Reinforcement trial and error, learn from mistakes**

#### Supervised Machine Learning – labeled data

Supervise, means to observe, and direct the execution of a task, project, or activity. We do this by teaching the model with some data from a **labeled** dataset, that is we load the model with knowledge so that we can have it predict future instances.

Two types of supervised learning:

* *Regression* is the process of predicting a continuous value as opposed to predicting a categorical value in classification. Predicts trends using previous labeled data
* *Classification* is the process of predicting a discrete class label, or category. Classifies labeled data

Supervised learning has more evaluation methods than unsupervised learning, and takes place in a controlled environment.

#### Unsupervised Machine Learning – unlabeled data

Unsupervised learning deals with finding patterns and groupings from the **unlabeled** data, and uses methods such as clustering. Unsupervised learning has fewer models, and fewer evaluation methods that can be used to ensure that the outcome of the model is accurate. As such, unsupervised learning creates a less controllable environment as the machine is creating outcomes for us.

### Machine Learning Statistical Models

|  |  |  |  |
| --- | --- | --- | --- |
| Regression | Classification | Clustering | Recommender |
| ***supervised*** | ***supervised*** | ***unsupervised*** | ***unsupervised*** |
|  |  |  |  |
| **Simple Linear Regression (SLR)** | **K-Nearest Neighbor (KNN)** | **K-Means** | **Content-Based** |
| **Multiple Linear Regression (MLR)** | **Decision Tree** | **Hierarchical** | **Collabrative-Filtering** |
| **Polynomial Regression** | **Logistic Regression (LR)** | **DBSCAN** |  |
| **Non-Linear Regression** | **Support Vector Machine (SVM)** |  |  |
|  |  |  |  |
| Evaluation |  |  |  |
| **K-Fold Cross Validation** | **Jaccard Index** |  |  |
| **Mean Square Error** | **F1-Score/Confusion Matrix** |  |  |
| **R-Squared** | **Log Loss** |  |  |

## Regression Models

|  |  |
| --- | --- |
| Regression | Explanation |
| ***supervised*** |  |
|  |  |
| **Simple Linear Regression (SLR)** |  |
| **Multiple Linear Regression (MLR)** |  |
| **Polynomial Regression** |  |
| **Non-Linear Regression** |  |
|  |  |
| Evaluation |  |
| **K-Fold Cross Validation** |  |
| **Mean Square Error** |  |
| **R-Squared** |  |

### Regression Models Overview

A model or estimator can be thought of as a mathematical equation used to predict a value given one or more other values. Regression is the process of predicting a *continuous value*. In regression, there are two types of variables,

**A dependent (response) variable; Y-axis, continuous only**

* dependent variable, can be seen as the state, target, or final goal we study, and try to predict
* dependent variable is notated by Y
* regression models relates Y (dependent variable), to a function of X (independent variables)
* dependent value should be continuous, and cannot be a discrete value

**One or more independent (predictor) variables; X-axis, categorical or continuous**

* also known as explanatory variables, can be seen as the causes of those states
* independent variables are shown conventionally by X
* independent variable or variables, can be measured on either a categorical, or continuous measurement scale

Objective of linear regression, is to minimize the MSE by finding the best coefficients of the linear equation (fit line).

#### Polynomial Regression Equations

Many different regressions exist that can be used to fit whatever the dataset looks like, for example, linear, binomial, cubic, quadratic or infinite degrees regression. In essence, all of these are *polynomial regression*, where the relationship between the independent variable X and the dependent variable Y is modeled as an Nth degree polynomial in X.

Polynomial Equation:

Polynomial Function:

A polynomial regression model can be transformed into a linear regression model if we assume then we can rewrite the polynomial equation as: . This now becomes a special case of multiple linear regression and can be solved using linear regression.

Therefore, polynomial regression models can fit using the model of least squares. Least squares is a method for estimating the unknown parameters in a linear regression model by minimizing the sum of the squares of the differences between the observed dependent variable in the given dataset and those predicted by the linear function.

#### Ordinary Least Squares (OLS)

OLS is a method for estimating the unknown parameters in a linear regression model. OLS chooses the parameters of a linear function of a set of explanatory variables by minimizing the sum of the squares of the differences between the target dependent variable and those predicted by the linear function. In other words, it tries to minimizes the sum of squared errors (SSE) or mean squared error (MSE) between the target variable (y) and our predicted output () over all samples in the dataset.

OLS can find the best parameters using of the following methods:

- Solving the model parameters analytically using closed-form equations

- Using an optimization algorithm (Gradient Descent, Stochastic Gradient Descent, Newton’s Method, etc.)

#### Explained Variance Regression Score

If is the estimated target output, y the corresponding (correct) target output, and Var is Variance, the square of the standard deviation, then the explained variance is estimated as follow:

, where best possible score is 1.0, lower values are worse.

#### Avoiding Over Fitting in Regression Models

* Adding too many independent variables without any theoretical justification may result in an overfit model. An overfit model is a real problem because it is too complicated for your data set and not general enough to be used for prediction. So, it is recommended to avoid using many variables for prediction.
* Categorical independent variables can be incorporated into a regression model by converting them into numerical variables. For example, given a binary variables such as car type, the code dummy zero for manual and one for automatic cars.
* Multiple linear regression is a specific type of linear regression. So, there needs to be a linear relationship between the dependent variable and each of your independent variables (use scatter plots to visually check for linearity).
* If the relationship displayed in your scatter plot is not linear, then you need to use non-linear regression.

### Regression Algorithms:

* Ordinal regression
* Poisson regression
* Fast forest quantile regression
* Linear, Polynomial, Lasso, Stepwise, Ridge regression
* Bayesian linear regression
* Neural network regression
* Decision forest regression
* Boosted decision tree regression
* KNN (K-nearest neighbors)

### Simple Linear Regression (SLR)

Simple Linear Regression is a method to help us understand the relationship between two variables:

* is the response variable - *dependent/target*
* *x* is the predictor variable - *independent*
* and are the coefficient of the linear equation
* bias coefficient, refers to the **intercept** of the regression line
* coefficient of predictor variable, refers to the **slope** of the regression line

**lm = LinearRegression() # constructor - need the "()"**

**x = df[['highway-mpg']] # predictor - two "[[]]" DataFrame**

**y = df['price'] # target - one "[]" Series**

**lm.fit(x,y) # fit model (find slope & intercept)**

**Yhat = lm.predict(x) # prediction, Yhat is the estimated value**

**Yhat[0:5]**

**array([13699.11161184, 13699.11161184, 19051.65470233, 10620.36193015,**

**15521.31420211])**

**print("Intercept (lm.intercept\_): {}\nSlope (lm.coef\_): {}".format(lm.intercept\_,lm.coef\_))**

**Intercept (lm.intercept\_): 38423.3058581574**

**Slope (lm.coef\_): [-821.73337832]**

**print("PRICE = {} + {} x HIGHWAY-MPG".format(lm.intercept\_,lm.coef\_))**

**PRICE = 38423.3058581574 + [-821.73337832] x HIGHWAY-MPG**

Simple Linear Regression is very fast, requires no parameter tuning, is easy to understand and is highly interpretable.

### Multiple Linear Regression (MLR)

**Multiple Linear Regression** uses multiple variables called independent (predictors) variables that best predict the value of the dependent (target) variable. There are two applications for multiple linear regression:

* First, it can be used when we would like to identify the strength of the effect that the independent variables have on the dependent variable.
* Second, it can be used to predict the impact of changes, that is, to understand how the dependent variable changes when we change the independent variables.

, or (transpose)

* are the parameters or the weight vectors of the regression equation
* is the bias coefficient and are the coefficient of the predictor variable

Multiple linear regression is the extension of the simple linear regression model. Most of the real-world regression models involve multiple predictors.

#### Estimating Multiple Linear Regression Parameters

Methods for estimating the parameter or coefficients (theta - θ)for multiple linear regression, when we have very large dataset:

* Ordinary Least Squares
  + Linear algebra operations
  + Takes a long time for large datasets (10k+ row)
* Optimization algorithm
  + Gradient descent
  + *Proper approach* for large datasets

### Polynomial Regression

Sometimes, the trend of data is not really linear, and looks curvy. In this case we can use Polynomial regression methods. In fact, many different regressions exist that can be used to fit whatever the dataset looks like, such as quadratic, cubic, and so on, and it can go on and on to infinite degrees.

In essence, we can call all of these, *polynomial regression*, where the relationship between the independent variable x and the dependent variable y is modeled as an nth degree polynomial in x:

Polynomial Equation:

**PolynomialFeatures()** function in Scikit-learn library, drives a new feature sets from the original feature set. That is, a matrix will be generated consisting of all polynomial combinations of the features with degree less than or equal to the specified degree. For example, lets say the original feature set has only one feature, *ENGINESIZE*. Now, if we select the degree of the polynomial to be 2, then it generates 3 features, degree=0, degree=1 and degree=2.

**fit\_transform** takes our x values, and output a list of our data raised from power of 0 to power of 2 (since we set the degree of our polynomial to 2):

Polynomial regression is a special case of linear regression, with the main idea of how do you select your features. Just consider replacing the x with x1, with x2,, and so on. Then the degree 2 equation would be turn into: . Now, we can deal with it as 'linear regression' problem. Therefore, this polynomial regression is considered to be a special case of traditional multiple linear regression. So, you can use the same mechanism as linear regression to solve such a problems.

### Non-Linear Regression

Following two conditions should indicate the use of non-linear regression:

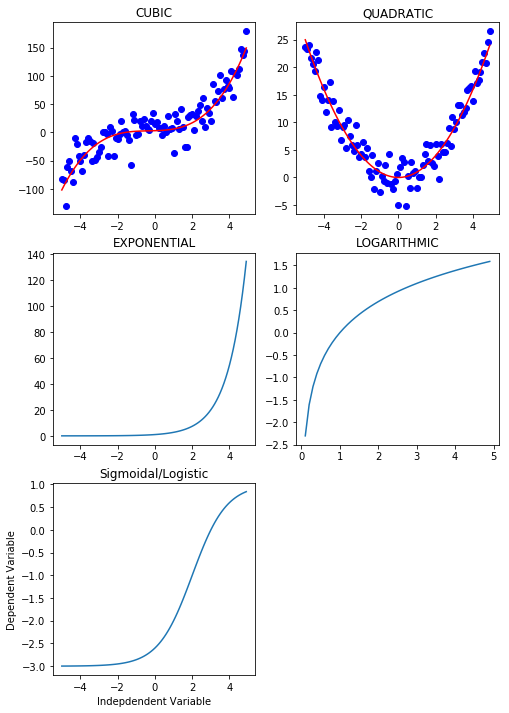
* First, non-linear regression is a method to model a non-linear relationship between the dependent variable and a set of independent variables
* Second, for a model to be considered non-linear, Y hat must be a non-linear function of the parameters Theta, not necessarily the features X

When it comes to non-linear equation, it can be the shape of exponential, logarithmic, and logistic, or many other types. Ordinary least squares method cannot be used to fit the data in non-linear regression and estimation of non-linear parameters is not easy.

How can a problem be determined to be a linear or non-linear:

* Visually figure out if the relation is linear or non-linear. It's best to plot bivariate plots of output variables with each input variable
* Calculate the correlation coefficient between independent and dependent variables, and if, for all variables, it is 0.7 or higher, there is a linear tendency and thus, it's not appropriate to fit a non-linear regression
* Unable to accurately model the relationship with linear parameters

If the model displays non-linear on a scatter plot, then you have to use either a polynomial regression, use a non-linear regression model, or transform your data. Essentially any relationship that is not linear can be termed as non-linear, and is usually represented by the polynomial of k degrees (maximum power of x): **y = ax3 + bx2 + cx + d**



Non-linear functions can have elements like exponentials, logarithms, fractions, and others. For example: **y = log(x)**. Or even, more complicated such as : **y = log(ax3 + bx2 + cx + d)**

#### Non-Linear: Cubic

A cubic function is a polynomial whose greatest exponent is 3. y = x3

#### Non-Linear: Quadratic

A quadratic function is a polynomial whose greatest exponent is 2. y = x2

#### Non-Linear: Exponential

An exponential function with base c is defined by **y = a + bcx** where b ≠0, c > 0 , c ≠1, and x is any real number. The base, c, is constant and the exponent, x, is a variable.

#### Non-Linear: Logarithmic

The response y is a results of applying logarithmic map from input *𝑥*'s to output variable *𝑦*. It is one of the simplest form of **log()**: i.e. *𝑦*=log(*𝑥*)

Please consider that instead of *𝑥*, we can use *𝑋*, which can be polynomial representation of the *𝑥*'s. In general form it would be written as **y = log(x)**

#### Non-Linear: Sigmoidal/Logistic

A sigmoid function is a mathematical function having a characteristic "S"-shaped curve or sigmoid curve. *𝑌*=*𝑎*+*𝑏*1+*𝑐*(*𝑋*−*𝑑*)

### Model Evaluation in Regression Models

The goal of regression is to build a model to accurately predict an unknown case. To this end, we have to perform regression evaluation after building the model.

* **Training Accuracy** is the percentage of correct predictions that the model makes when using the test dataset. However, a high training accuracy isn't necessarily a good thing and may result in over-fitting. *Over-fitting* is when the model is overly trained to the dataset, which may capture noise and produce a non-generalized model.
* **Out-of-Sample Accuracy** is the percentage of correct predictions that the model makes on data that the model has not been trained on.

#### Train and Test on the Same Dataset

Train the model on the entire dataset, then test the model on a subset of the dataset. Evaluate the model based on the error between the actual values and the predicted values by the model.

Train and Test on the same dataset method will have a *high training accuracy* and a *low out-of-sample accuracy*, most likely due to over-fitting.

#### Train/Test Split

Train/test split involves splitting the dataset into training and testing sets respectively, which are mutually exclusive. This will provide a more accurate evaluation on out-of-sample accuracy because the testing dataset is not part of the dataset that has been used to train the data. However, the test dataset should also be used afterwards to train the model so as not to lose valuable data.

Train/Test Split method is highly dependent on which dataset the data is trained and tested on, and will have a *high out-of-sample accuracy*.

Since this data has not been used to train the model, the model has no knowledge of the outcome of these data points. It is more realistic for real-world problems and, in essence, is out-of-sample testing.

#### K-Fold Cross-Validation

K-fold cross-validation in its simplest form *performs multiple train/test splits*, using the same dataset where each split is different. Then, the result is average to produce a more consistent out-of-sample accuracy.

1. In the first fold for example, we use the first 25 percent of the dataset for testing and the rest for training. The model is built using the training set and is evaluated using the test set
2. Then, in the next round or in the second fold, the second 25 percent of the dataset is used for testing and the rest for training the model
3. Again, the accuracy of the model is calculated
4. We continue for all folds
5. Finally, the result of all four evaluations are averaged, keeping in mind that each fold is distinct, where no training data in one fold is used in another

### Evaluation Metrics in Regression Models

Evaluation metrics are used to explain the performance of a model. Basically, we can compare the actual values and predicted values, to calculate the accuracy of our regression model. Evaluation metrics, provide a key role in the development of a model, as it provides insight to areas that require improvement.

In the context of regression, the error of the model is the difference between the data points and the trend line generated by the algorithm. Since there are multiple data points, an error can be determined in multiple ways:

* **Mean absolute error (MAE)** is the mean of the absolute value of the errors
* **Mean squared error (MSE)** is the mean of the squared error - the squared term, exponentially increasing larger errors in comparison to smaller ones
* **Root mean squared error (RMSE)** is the square root of the mean squared error. This is one of the most popular of the evaluation metrics, because root mean squared error is interpretable in the same units as the response vector or y units, making it easy to relate its information.
* **Relative Absolute Error (RAE, or Residual Sum of Square)** takes the total absolute error and normalizes it by dividing by the total absolute error of the simple predictor.
* **Relative Square Error (RSE)** is very similar to relative absolute error, but is widely adopted by the data science community, as it is used for calculating R squared
* **R squared** is not an error per se, but is a popular metric for the accuracy of your model. It represents how close the data values are, to the fitted regression line. The higher the R-squared, the better the model fits your data.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

Each of these metrics can be used for quantifying of your prediction. The choice of metric, completely depends on the type of model, your data type and domain of knowledge.

## Classification

|  |  |
| --- | --- |
| Classification |  |
| ***supervised*** |  |
|  |  |
| **K-Nearest Neighbor (KNN)** |  |
| **Decision Tree** |  |
| **Logistic Regression (LR)** |  |
| **Support Vector Machine (SVM)** |  |
|  |  |
| Evaluation |  |
| **Jaccard Index** |  |
| **F1-Score/Confusion Matrix** |  |
| **Log Loss** |  |

In machine learning, classification is a supervised learning approach which can be thought of as a means of categorizing or classifying some unknown items into a discrete set of classes. Classification attempts to learn the relationship between a set of feature variables and a target variable of interest. The target attribute in classification is a categorical variable with discrete values. Given a set of training data points along with the target labels, classification determines the class label for an unlabeled test case.

There are many types of classification algorithms and machine learning, such as decision trees, naive bayes, linear discriminant analysis, k-nearest neighbor, logistic regression, neural networks, and support vector machines.

A good sample of classification is the loan default prediction. Suppose a bank is concerned about the potential for loans not to be repaid? If previous loan default data can be used to predict which customers are likely to have problems repaying loans, these bad risk customers can either have their loan application declined or offered alternative products. The goal of a loan default predictor is to use existing loan default data which has information about the customers such as age, income, education et cetera, to build a classifier, pass a new customer or potential future default to the model, and then label it, i.e the data points as defaulter or not defaulter. Or for example zero or one. This is how a classifier predicts an unlabeled test case. Please notice that this specific example was about a binary classifier with two values. We can also build classifier models for both binary classification and multi-class classification.

There are many types of classification algorithms and machine learning, such as decision trees, naive bayes, linear discriminant analysis, k-nearest neighbor, logistic regression, neural networks, and support vector machines.

### K-Nearest Neighbors Algorithm (KNN)

The K-Nearest Neighbors is an algorithm for *supervised learning* and is a *classification algorithm* that takes a bunch of labeled points and uses them to learn how to label other points. This algorithm classifies cases based on their similarity to other cases. In K-Nearest Neighbors, data points that are near each other are said to be neighbors. K-Nearest Neighbors is based on this paradigm. Thus, the distance between two cases is a measure of their dissimilarity. There are different ways to calculate the similarity or conversely, the distance or dissimilarity of two data points. For example, this can be done using Euclidean distance.

In a classification problem, the K-Nearest Neighbors algorithm works as follows:

1. Pick a value for K
2. Calculate the distance from the new case hold out from each of the cases in the dataset
3. Search for the K-observations in the training data that are nearest to the measurements of the unknown data point
4. Predict the response of the unknown data point using the most popular response value from the K-Nearest Neighbors

**Pick a value for K** –

* low value of K results in a highly complex model and may result in overfitting.
* very high value of K such as K equals 20, then the model becomes overly generalized.
* solution is to reserve a part of your data for testing the accuracy of the model. Once you've done so, choose K equals one and then use the training part for modeling and calculate the accuracy of prediction using all samples in your test set. Repeat this process increasing the K and see which K is best for your model.

**Calculate similarities between two data points** –

* use a specific type of Minkowski distance to calculate the distance of these two customers, which is the Euclidean distance.
* Euclidean distance:

Nearest neighbors analysis can also be used to compute values for a *continuous target*. In this situation, the average or median target value of the nearest neighbors is used to obtain the predicted value for the new case.

### Evaluation Metrics in Classification

Evaluation metrics explain the performance of a model.

#### Jaccard Index - best at 1

Jaccard Index (or Jaccard similarity coefficient) is defined as size of the intersection divided by the size of the union of two label sets (picture a Venn diagram).

For example, if y is actual labels (10 data points) and is the predicted labels (10 data points), and 8 data points are accurately predicted by the model, than;

#### F1-Score (Confusion Matrix) - best at 0

This matrix shows the corrected and wrong predictions, in comparison with the actual labels. Each confusion matrix row shows the Actual/True labels in the test set, and the columns show the predicted labels by classifier. A good thing about the confusion matrix is that it shows the model’s ability to correctly predict or separate the classes.

* *Precision* is a measure of the accuracy, provided that a class label has been predicted. It is defined by:

**precision = True Positive/(True Positive + False Positive)**

* *Recall* is the true positive rate:

**recall = True Positive/(True Positive + False Negative)**

* F1-Score is the harmonic average of the precision and recall, where an F1 score reaches its best value at 1 (which represents perfect precision and recall) and its worst at 0

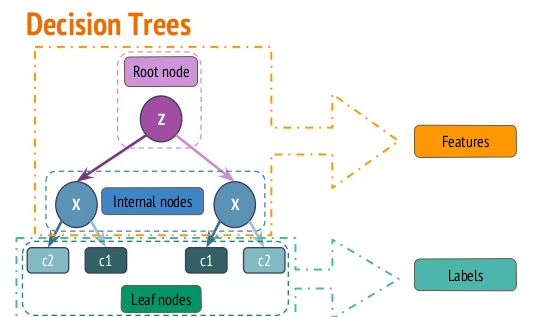
**F1-Score = 2x (precision x recall)/ (precision + recall)**

#### Logarithmic Loss (Log Loss) - best at 0

Logarithmic loss (also known as Log loss) measures the performance of classifier where the predicted output is a probability value between 0 and 1. Classifiers with lower log loss has better accuracy.

### Decision Tree

Decision trees are built using recursive partitioning to classify the data, i.e., by splitting the training set into distinct nodes, where one node contains all of or most of one category of the data. A decision tree can be constructed by considering the attributes one by one:

* First, choose an attribute from our dataset.
* Calculate the significance of the attribute in the splitting of the data.
* Next, split the data based on the value of the best attribute,
* then go to each branch and repeat it for the rest of the attributes.
* After building this tree, you can use it to predict the class of unknown cases

Decision trees are about testing an attribute and branching the cases based on the result of the test:

1. each *internal node* corresponds to a test
2. each *branch* corresponds to a result of the test
3. each *leaf node* assigns a patient to a class

#### Best Attributes Selection

Best attributes have more *predictiveness*, less *impurity* and less *entropy*. Selection of the attribute is based on the tree with the higher **information gain** after splitting. Constructing a decision tree is all about finding attributes that return the highest information gain.

The choice of attribute to split data is very important and it is all about **purity** of the leaves after the split. A node in the tree is considered pure if in 100 percent of the cases, the nodes fall into a specific category of the target field. In fact, the method uses recursive partitioning to split the trading records into segments by minimizing the impurity at each step. Impurity of nodes is calculated by **entropy** of data in the node.

**Entropy** is the amount of randomness or uncertainty in the data. In decision trees, we're looking for trees that have the smallest entropy in their nodes *after the split.* The entropy is used to calculate the *homogeneity* of the samples in that node.

* completely homogeneous, entropy is *zero*
* equally divided (50% each), entropy is *one*

**Information gain** is the information that can increase the level of certainty after splitting. It is the entropy of a tree before the split minus the weighted entropy after the split by an attribute. As entropy or the amount of randomness decreases, the information gain or amount of certainty increases and vice versa.

### Logistic Regression

Logistic regression is a classification algorithm for categorical variables. Logistic regression is analogous to linear regression, but tries to predict a categorical or discrete target field, such as 0 or 1, yes or no, etc., instead of a numeric one. Dependent variables should be continuous. If categorical, they should be dummy or indicator coded. This means we have to transform them to some continuous value. Logistic regression can be used for both binary classification and multi-class classification. Sigmoid functions are a main part of logistic regression.

Logistic regression can be used for the following:

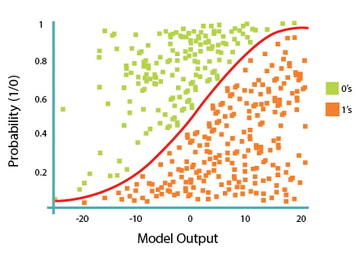
* First, when the target field in your data is categorical or specifically is binary. Such as zero/one, yes/no, true/false, churn/no churn, positive/negative and so on.
* Second, you need the probability of your prediction. Logistic regression returns a probability score between zero and one for a given sample of data.
* Third, if your data is linearly separable. The decision boundary of logistic regression is a line or a plane or a hyper plane. A classifier will classify all the points on one side of the decision boundary as belonging to one class, and all those on the other side as belonging to the other class.
* Fourth, you need to understand the impact of a feature. You can select the best features based on the statistical significance of the logistic regression model coefficients or parameters.

#### Logistic Regression vs. Linear Regression

While Linear Regression is suited for estimating continuous values (e.g. estimating house price), it is not the best tool for predicting the class of an observed data point. In order to estimate the class of a data point, we need some sort of guidance on what would be the **most probable class** for that data point. For this, we use **Logistic Regression**.

The difference between linear and multiple linear regression is that the linear regression contains only one independent variable while multiple regression contains more than one independent variables. The best fit line in linear regression is obtained through least square method.

**Linear regression** finds a function that relates a continuous dependent variable, **y**, to some predictors.



**Logistic Regression** is a variation of Linear Regression, useful when the observed dependent variable, **y**, is categorical. It produces a formula that predicts the probability of the class label as a function of the independent variables. Logistic regression fits a special s-shaped curve by taking the linear regression and transforming the numeric estimate into a probability with the following function, which is called *sigmoid function* 𝜎. Logistic Regression passes the input through the logistic/sigmoid but then treats the result as a probability:

* The Linear regression models data using continuous numeric value. As against, logistic regression models the data in the binary values.
* Linear regression requires to establish the linear relationship among dependent and independent variable whereas it is not necessary for logistic regression.
* In the linear regression, the independent variable can be correlated with each other. On the contrary, in the logistic regression, the variable must not be correlated with each other.

Linear regression models data using a straight line where a random variable, Y (response variable) is modelled as a linear function of another random variable, X (predictor variable). On the other hand, the logistic regression models the probability of the events in bivariate which are essentially occurring as a linear function of a set of dependent variables.

### Logistic Regression Training

The main objective of training and logistic regression is to change the parameters of the model, so as to be the best estimation of the labels of the samples in the dataset.

1. Initialize the parameters randomly
2. Feed the cost function with training set and calculate the error
3. Calculate the gradient of the cost function
4. Update the weights with the new values
5. Go to step 2 until the cost is small enough
6. Predict the new customer X

* Cost Function
* Cost Function -MSE

Using the Gradient Descent method will minimize the cost function.

Minimizing the cost function will give the best parameter for the model.

#### Gradient Descent

Gradient Descent is a technique to use the derivative of a cost function to change the parameter values in order to minimize the cost or error. Minimum point of the Error bowl.

### Support Vector Machine

Support Vector Machine (SVM) is a supervised algorithm that can classify cases by dividing a data set into two or more classes using a separator. SVM works by:

* mapping data to a high-dimensional feature space so that data points can be categorized (*kernelling)*, even when the data are not otherwise linearly separable.
* A separator between the categories is found,
* then the data is transformed in such a way that the separator could be drawn as a hyperplane.
* Following this, characteristics of new data can be used to predict the group to which a new record should belong.

**kernel function** is the mathematical function used for mapping data into a higher dimensional space, in such a way that can change a linearly inseparable dataset into a linearly separable dataset, and can be of different types, such as linear, polynomial, Radial Basis Function, or RBF, and sigmoid.

1. Take a 1D linearly inseparable dataset (x)
2. Define a function to map to 2D, ϕ(x) = [x,x2]

One reasonable choice as the best hyperplane is the one that represents the largest separation or margin between the two classes. Data points closest to the hyperplane are support vectors. It is intuitive that only support vectors matter for achieving our goal. And thus, other trending examples can be ignored. We tried to find the hyperplane in such a way that it has the maximum distance to support vectors.

the hyperplane is learned from training data using an optimization procedure that maximizes the margin. And like many other problems, this optimization problem can also be solved by gradient descent, which is out of scope of this video.

The two main advantages of support vector machines are that they're:

* accurate in high-dimensional spaces
* use a subset of training points in the decision function called, support vectors, so it's also memory efficient

The disadvantages of Support Vector Machines include the fact that the algorithm is:

* prone for over-fitting if the number of features is much greater than the number of samples
* do not directly provide probability estimates, which are desirable in most classification problems
* not very efficient computationally if your dataset is very big, such as when you have more than 1,000 rows

SVM applications:

* Image recognition
* Test category assignments
* Detecting spam
* Sentiment analysis
* Gene expression classifications
* Regression, outlier detection and clustering

## Clustering

Customer segmentation is the practice of partitioning a customer base into groups of individuals that have similar characteristics. One approach for customer segmentation is clustering. Clustering can group data only unsupervised, based on the similarity of customers to each other. It will partition your customers into mutually exclusive groups.

|  |  |
| --- | --- |
| Clustering |  |
| ***unsupervised*** |  |
|  |  |
| **K-Means (partitioning-based)** | **Uses distance matrix to assign datapoints to centroids** |
| **Hierarchical** |  |
| **DBSCAN** |  |
|  |  |
| Evaluation |  |
| **Minkowski distance, Euclidean distance** | **Sum of Square error** |
| **Elbow method** |  |

Clustering means finding clusters in a dataset, *unsupervised*. A cluster is a group of data points or objects in a dataset that are similar to other objects in the group, and dissimilar to datapoints in other clusters.

**What is different between clustering and classification?**

* *Classification* algorithms predict categorical classed labels. classification is a supervised learning where each training data instance belongs to a particular class
* In *clustering*, the data is unlabeled and the process is unsupervised. For example, we can use a clustering algorithm such as k-means to group similar customers as mentioned, and assign them to a cluster, based on whether they share similar attributes, such as; age, education, and so on

**Clustering Algorithms:**

* *Partitioned-based clustering* *algorithms* produces fear like clusters:
  + K-Means, K-Medians or Fuzzy c-Means
  + relatively efficient
  + used for medium and large sized databases
* *Hierarchical clustering algorithms* produce trees of clusters, such as agglomerative[[2]](#footnote-2) and divisive algorithms:
  + very intuitive
  + generally good for use with small size datasets
* *Density-based clustering algorithms* produce arbitrary shaped clusters:
  + good when dealing with spatial clusters or when there is noise in your data set
  + DB scan algorithm.

### K-Means Clustering

K-Means is a type of *partitioning clustering*, that is, it divides the data into K non-overlapping subsets or clusters without any cluster internal structure or labels. Objects within a cluster are very similar, and objects across different clusters are very different or dissimilar.

Real-world applications of k-means:

* Customer segmentation
* Understand what the visitors of a website are trying to accomplish
* Pattern recognition
* Machine learning
* Data compression

K-Means is heuristic algorithm, there is no guarantee that it will converge to the global optimum, but the result may be a *local* optimum. The algorithm can be run multiple times in order to get better outcomes.

#### K-Means uses dissimilarity metrics:

Though the objective of K-means is to form clusters in such a way that similar samples go into a cluster and dissimilar samples fall into different clusters, it can be shown that instead of a similarity metric, we can use *dissimilarity metrics*. In other words, conventionally, the distance of samples from each other is used to shape the clusters.

* distance of samples from each other is used to shape the clusters
* tries to *minimize the* *intra-cluster* distances
* tries to *maximize the* *inter-cluster* distances
* divide the data into non-overlapping clusters without any cluster-internal structure
* Minkowski distance to calculate the distance of these two customers. Euclidean distance:

#### Centroid of Cluster (random centers of clusters) – calculation steps:

Initializing K is not an easy task.

* 1. Initialize k=3, number of sphere-like clusters=3. Randomly place three centroids, one for each cluster
  2. Use distance matrix (calculations) to calculate distance of each point from each centroid
  3. Assign each point in the data to the closest centroid, creating a cluster
  4. Calculate the position of the k centroids
  5. Repeat 2-4 until centroids no longer move/algorithm converges

Since centroids are chosen randomly, the model will have high error. Clusters can be reshaped in such a way that the total distance of all members of a cluster from its centroid can be minimized (using the Sum of Square Error (SSE)) to create better clusters with less error:

#### K-Means Accuracy

The correct choice of K is often ambiguous because it's very dependent on the shape and scale of the distribution of points in a dataset. One way is to run different values of k in a loop. Increasing k will always decrease the error, and where the rate of decrease sharply shifts is selected as the best k (*elbow method*).

External Approach: compare clusters with the ground truth, if available

Internal Approach: average the distance between data points within a cluster

### Hierarchical Clustering

Hierarchical clustering algorithms build a hierarchy of clusters where each node is a cluster consisting of the clusters of its daughter nodes. Strategies for hierarchical clustering generally fall into two types, *divisive* and *agglomerative*:

* **Divisive** is dividing the cluster (top down) and starts with a large cluster and break it down into smaller pieces
* **Agglomerative** is pairs of clusters are merged together as they move up the hierarchy (bottom up) – more popular

Represented by dendrogram. Horizontal lines depict merged clusters. The numbers of clusters is not initially defined.

#### Agglomerative Algorithm – bottom up:

1. create n clusters, one for each data point
2. Compute the proximity matrix - measure distances between clusters and merge the clusters that are nearest
3. Repeat
   1. Merge the two closest clusters
   2. Update the proximity matrix
4. Continue until only a single cluster remains

|  |  |  |
| --- | --- | --- |
| **Hierarchical Clustering - Agglomerative Algorithm** | | |
| **ADVANTAGES** |  | **DISADVANTAGES** |
| **Does not require numbers of clusters to be specified** |  | **Can never undo any previous steps throughout the algorithm** |
| **Easy to implement** |  | **Generally has long runtimes** |
| **Produces a dendrogram, which helps with understanding the data** |  | **Large dataset - difficult to identify the number of clusters by the dendrogram** |

#### Similarity/Distance

We can use different distance measurements to calculate the proximity matrix. For instance, Euclidean distance. So, if we have a data set of n patience, we can billed an n by n dissimilarity distance matrix. It will give us the distance of clusters with one data point. *Dataset to Dissimilarity Matrix*

Different criteria can be used to find the closest clusters and merge them. It depends on the data type, dimensionality of data and most importantly, the domain knowledge of the data set. Different approaches to defining the distance between clusters distinguish the different algorithms.

Types of **dissimilarity calculation** between clusters:

* Single-Linkage Clustering – minimum distance between clusters
* Complete-Linkage Clustering – maximum distance between clusters
* Average-Linkage Clustering – average distance between clusters
* Centroid-Linkage Clustering – distance between cluster centroids

### K-Means vs. Hierarchical Clustering

|  |  |  |
| --- | --- | --- |
| **K-Means versus Hierarchical Clustering Algorithms** | | |
| **K-MEANS** |  | **HIERARCHICAL CLUSTERING** |
| **Much more efficient for large datasets** |  | **Can be slow for large datasets** | |
| **Requires number of clusters to be specified** |  | **Does not require number of clusters to run** |
| **Gives only one partitioning of the data based on the predefined number of clusters** |  | **Gives more than one portioning depending on the resolution** |
| **Potentially returns different clusters each time it is run due to random initialization of centroids** |  | **Always generates the same clusters** |

### DBSCAN-based Clustering

Density Based clustering algorithms are proper for arbitrary shape clusters and are used to locates regions of high-density that are separated from one another by regions of low density, and separates outliers. K-Means assigns all points to a cluster, even if they do not belong in any.

Density-Based Spatial Clustering of Applications with Noise (DBSCAN) is a very popular Density-Based clustering algorithm. DBSCAN is particularly effective for tasks like class identification on a spatial context and can:

* can find arbitrarily shaped clusters
* find a cluster completely surrounded by a different cluster
* has a notion of noise and is robust to outliers
* does not require one to specify the number of clusters such as in K-means

DBSCAN works on the idea that if a particular point belongs to a cluster it should be near to lots of other points in that cluster. It works based on two parameters;

* *Radius of Neighborhood* (R, Epsilon) determines a specified radius that if it includes enough points within it, we call it a dense area.
* *Minimum Number of Neighbors* (M, minimumSamples) determines the minimum number of data points we want in a neighborhood to define a cluster.

The idea behind the DBSCAN algorithm is to visit each point and find its type first, then we group points as clusters based on their types. Each point is either:

* **Core** point - within neighborhood of the point there are at least M points
* **Border** point - neighborhood contains less than M data points, or it is reachable from some core point. Reachability means it is within our distance from a core point
* **Outlier** point - not a core point and also is not close enough to be reachable from a core point
* Identify all the data points as core, border or outlier points
* Next step is to connect core points that are neighbors and put them in the same cluster
* A cluster is formed as at least one core point plus all reachable core points plus all their borders

## Recommender Systems

Recommender systems try to capture patterns of peoples’ behaviors and use it to predict what else they may like. Advantages of recommender systems are broader exposure, possibility of continual usage or purchase of products, better experience, increased potential revenue and better security for its customers.

|  |  |
| --- | --- |
| Recommender |  |
| ***unsupervised*** |  |
|  |  |
| **Content-Based** |  |
| **Collabrative-Filtering** |  |
|  |  |
| Evaluation |  |
|  |  |
|  |  |
|  |  |

**Types of Recommender Systems**

* *Content-based* - systems try to figure out what a user's favorite aspects of an item are, and then make recommendations on items that share those aspects – *similar items*
* *Collaborative filtering* - techniques find similar groups of users, and provide recommendations based on similar tastes within that group – *similar preferences*

**Implementing Recommender Systems**

* *Memory based* – uses entire user-item dataset to generate a recommendation. Approximates users or items using statistical techniques such as Pearson Correlation, Cosine Similarity and Euclidean Distance, among others.
* *Model-based* - a model of users is developed in an attempt to learn their preferences. Models can be created using Machine Learning techniques like regression, clustering, classification, and so on.

### Content-Based Recommender Systems

A Content-based recommendation system tries to recommend items to users based on their profile build upon their preferences and tastes. It is shaped based on user ratings, including the number of times that user has clicked on different items or perhaps even liked those items. The recommendation process is based on the similarity between those items. Similarity or closeness of items is measured based on the similarity in the content of those items. When we say content, we're talking about things like the items category, tag, genre, and so on.

|  |  |  |
| --- | --- | --- |
| **Recommender Systems – Content-Based Filtering** | | |
| **ADVANTAGES** |  | **DISADVANTAGES** | |
| **Learns user's preferences** |  | **Doesn't take into account what others think of the item, so low quality item recommendations might happen** | |
| **Highly personalized for the user** |  | **Extracting data is not always intuitive** | |
|  |  | **Determining what characteristics of the item the user dislikes or likes is not always obvious** | |

### Collaborative-Filtering Recommender Systems

Collaborative filtering is based on the fact that relationships exist between products and people's interests. Many recommendation systems use collaborative filtering to find these relationships and to give an accurate recommendation of a product that the user might like or be interested in.

Collaborative filtering has basically two approaches:

* *User-based collaborative filtering* is based on the user similarity or neighborhood. Recommendation is based on users of the same neighborhood with whom he or she shares common preferences
* *Item-based collaborative filtering* is based on items’ similarity. Similar items build neighborhoods on the behavior of users, however, that it is not based on their contents

The collaborative filtering engine first looks for users who are similar. That is users who share the active users rating patterns. Collaborative filtering basis this similarity on things like history, preference, and choices that users make when buying, watching, or enjoying something.

Challenges of collaborative filtering:

* *Data Sparsity* – users in general rate only a limited number of items
* *Cold Start* – difficulty in recommendation to new user or new items
* *Scalability* –performance degrades and the number of users/items increase

# Applied Data Science Capstone

## Week 1 - Introduction to Capstone Project

### Introduction to Capstone Project

City segmentation and Clustering using the geographical coordinates of the center of each neighborhood, and then using a combination of location data and machine learning, you will group the neighborhoods into clusters like this.

Modedule 1-3 learning additional skills

Module 4 & 5 Project

### Location Data Providers

Location data is data describing places and venues, such as their geographical location, their category, working hours, full address, and so on, such that for a given location given in the form of its geographical coordinates (or latitude and longitude values) one is able to determine what types of venues exist within a defined radius from that location.

Location Data Providers: FourSquare, Google, Yelp

API features: Rate Limits, Cost, Geospatial Services, Accuracy, Update Frequency

Foursquare location dataset – continually updated

Install GibBash – needed for first project:

<https://medium.com/@aklson_DS/how-to-properly-setup-your-github-repository-windows-version-ea596b398b>

**ACER@ACER-PC MINGW64 ~**

**$ ssh-keygen -t rsa -b 4096 -C "asimislam@yahoo.com"**

**Generating public/private rsa key pair.**

**Enter file in which to save the key (/c/Users/ACER/.ssh/id\_rsa):**

**Your identification has been saved in /c/Users/ACER/.ssh/id\_rsa.**

**Your public key has been saved in /c/Users/ACER/.ssh/id\_rsa.pub.**

**The key fingerprint is:**

**SHA256:UGvw/uwKqt81Q6Zq6BOGV3+KFBsTN0BeHzcRX3i6gx8 asimislam@yahoo.com**

**“id\_rsa” - private version of key**

**“id\_rsa.pub” - public version of key**

**ACER@ACER-PC MINGW64 ~/.ssh**

**$ eval "$(ssh-agent -s)"**

**Agent pid 878**

**ACER@ACER-PC MINGW64 ~/.ssh**

**$ ssh-add ~/.ssh/id\_rsa**

**Identity added: /c/Users/ACER/.ssh/id\_rsa (asimislam@yahoo.com)**

## Week 2 - Foursquare API

### Introduction to Foursquare

Foursquare crowd-sourced their data and had people use their app to build their dataset and add venues and complete any missing information they had in their dataset. Currently its location data is the most comprehensive out there, and quite accurate that it powers location data for many popular services like Apple Maps, Uber, Snapchat, Twitter and many others, and is currently being used by over 100,000 developers, and this number is only growing.

To try out Foursquare you can either use their phone app or their online website. So you can go to Foursquare com, and this will be the landing page. In the center, you will find two fields: one for what you're looking for and another one for the location of interest.

### Getting Foursquare API Credentials

**MyApps:** [**coursera-capstone-project**](https://foursquare.com/developers/app/STNCYVGISTOCFSEXUEVYJMF4KXKR2JQX1OSKUYFMJRYG44WZ)

**Client ID STNCYVGISTOCFSEXUEVYJMF4KXKR2JQX1OSKUYFMJRYG44WZ**

**Client Secret 4N2OX52CB5RZFEVVTJIZ1QFIHFQSS23CW50C4PNTGON3RIIO**

Docs, Places API, Endpoints -= ‘search’

### Using Foursquare API

Communicating with the Foursquare database is really very easy, all thanks to their RESTful API. You simply create a uniform resource identifier, or URI, and you append it with extra parameters depending on the data that you are seeking from the database. Any call request you make is composed of, we can call this base URI, which is api.foursquare.com/v2, and you can request data about venues, users, or tips. But, every time you make a call request, you have to pass your developer account credentials, which are your Client ID and Client Secret as well as what is called the version of the API, which is simply a date.

**RESTful API: communicate with database via groups and endpoints using URI**

**URI api.foursquare.com/v2/**

* **Client ID**
* **Client Secret**
* **Version (date)**

**Personal account uses regular calls**

**Example 1: Search for coffee near Conrad Hotel (40.73,-74.01):**

**(personal account limit: 99k free msg/day)**

**GET:** <https://api.foursquare.com/v2/venues/search>**?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619 &**

**ll = 70.73,-74.01 &**

**query = coffee**

* **receives a ‘Coffee Shops’ JSON data file**
* **Venue has UniqueID, Name, Location and Category**

**Example 2: Venue information for “Kaffe 1668”:**

**(premium account limit, 500 free msg/day)**

**GET:** [https://api.foursquare.com/v2/venues/<uniqueID for Kaffee 1668>](https://api.foursquare.com/v2/venues/%3cuniqueID%20for%20Kaffee%201668%3e)**?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619**

* **receives a ‘Kaffe 1668’ JSON data file**
* **Venue has UniqueID, Name, Location, contact info, statistics, rating, tips**

**Example 3: Tips for “Kaffe 1668”:**

**(personal account limit: 99k free msg/day)**

**GET:** [https://api.foursquare.com/v2/venues/<uniqueID for Kaffee 1668>](https://api.foursquare.com/v2/venues/%3cuniqueID%20for%20Kaffee%201668%3e)**/tips?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619**

* **receives a ‘Kaffe 1668’ JSON data file**
* **Tip has only 2 tips since we’re using a personal account**

**Example 4: Look up a specific user:**

**(personal account limit: 99k free msg/day)**

**GET:** [https://api.foursquare.com/v2/users/<uniqueID for user>](https://api.foursquare.com/v2/users/%3cuniqueID%20for%20user%3e)**?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619**

* **receives a ‘Kaffe 1668’ JSON data file**
* **User has UniqueID, first & last name, friends, contact info, gender, tips**

**Example 5: Explore the area around Conrad Hotel:**

**(personal account limit: 99k free msg/day)**

**GET:** <https://api.foursquare.com/v2/venues/explore>**?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619 &**

**ll = 70.73,-74.01**

* **receives a ‘Coffee Shops’ JSON data file**
* **Venue has UniqueID, Name, Location and Category**

**Example 6: Trending venues area around Conrad Hotel:**

**(personal account limit: 99k free msg/day)**

**GET:** <https://api.foursquare.com/v2/venues/trending>**?**

**client\_id = STNCY\*\*\*\*G44WZ &**

**client\_secret = 4N2OX\*\*\*\*3RIIO &**

**v = 20190619 &**

**ll = 70.73,-74.01**

* **receives a ‘Coffee Shops’ JSON data file**
* **Venue has UniqueID, Name, Location and Category**

## Week 3 - Neighborhood Segmentation and Clustering

### Clustering: K-Means Clustering

### Lab: Clustering

### Lab: Segmenting and Clustering Neighborhoods in New York City

### Peer-review Assignment: Segmenting and Clustering Neighborhoods in Toronto

## Week 4 - Capstone Project

## Week 5 - Capstone Project (Cont'd)

1. A **curvilinear line** is a smooth curve like a parabola or logarithmic curve. It is actually a group/approximation of a large number of curves. More formal definition: something that is formed or characterized by a set of curved lines. [↑](#footnote-ref-1)
2. Agglomerate - to collect or gather into a cluster or mass [↑](#footnote-ref-2)