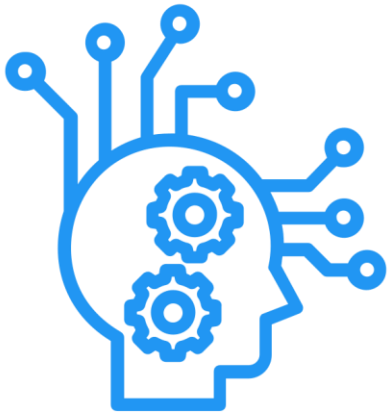
Data Science – End 2 End Beginners Course

Part 1

Machine Learning and Data Analytics



DataWisdomX

Practical Data Science for Everyone

Author: Nitin Singhal
https://datawisdomx.com
nitin@datawisdomx.com

Introduction

- Hi everyone. Welcome to the End 2 End course on Data Science
- **DataWisdomX** is an open, free to join data science learning and knowledge sharing platform
  - It provides training courses for learners and connects businesses to experts to solve their data science problems
- The **Objective** of this course is to teach students how to do an End-2-End data science project
  - From Problem definition, data sourcing, wrangling and modelling
  - To analyzing, visualizing and deploying & maintaining the models
- This course is for anyone interested in learning data science – analyst, programmer, non-technical professional, student, etc
- Having seen available data science courses and books, we feel there is a lack of an End 2 End approach
  - Quite often you learn the different algorithms but don't get a holistic view, especially around the process and deployment
  - Also, either too much or limited mathematical details are provided for different algorithms
- The End 2 End Data science course will be divided into 4 parts
  - **Part 1** is a Beginner's course that covers basic **Machine Learning and Data Analytics**
  - **Part 2** will cover Intermediate and Advanced machine learning techniques – **Deep Learning and NLP (Natural Language Processing)**
  - **Part 3** will cover **Advanced** machine learning techniques - **Reinforcement Learning and Computer Vision**
  - **Part 4** will cover **Data Engineering** – **Databases and Big Data tools (Hadoop, Spark)**
- Throughout the course detailed lectures covering the maths/logic of the algorithms, python code examples and online resources are provided to support the learning process
- More details are available on our website - https://datawisdomx.com/
- Course material including python code and data is available at -
  - https://github.com/datawisdomx/DataScienceCourse

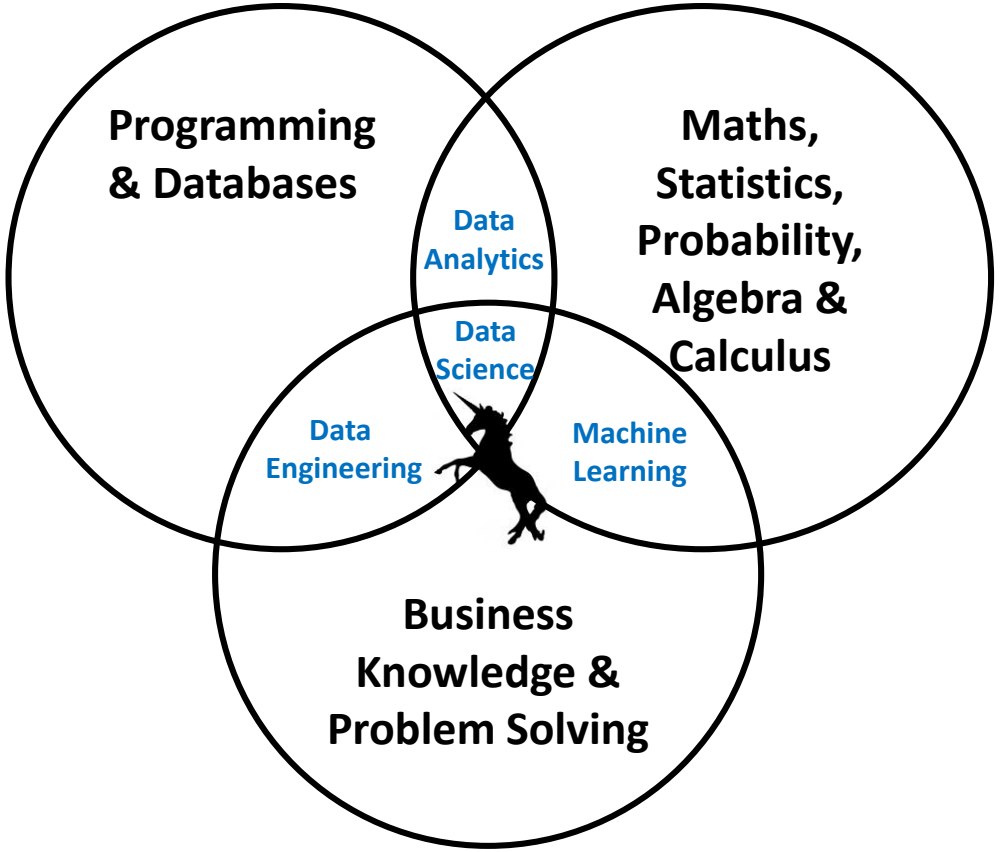
Table of Contents

Why Data Science

- Exponential growth in data being generated globally by organisations and individuals
  - Exabytes (10^{18} bytes or 1bn GB) /day
  - And its still increasing
- Companies like Amazon, Netflix, Google and Quantitative funds have disrupted traditional industries like retail, entertainment, advertising and investing
  - By using data & analytics as a competitive advantage
- Quite often companies have large amounts of data but don't know how to use it as a competitive advantage
- That's where Data scientists can play a key role, guiding the companies through the entire process
  - Sourcing, wrangling, modelling and analysing the data
  - To generate insights for decision making
- Customers want hyper customised products & services in real-time at competitive price
  - Which can be delivered only by using all available data points to make the right decisions
  - Given the size of data sets, it is difficult to do this without using data science
- Organisations that fail to make data science part of their business processes and decision making will be unable to compete in the future
- Data science is no longer a buzzword, it's a necessity

What is Data Science

- **Data Science** is a multi-disciplinary profession which requires a combination of 3 core skillsets
  - Maths, Statistics, Probability, Algebra & Calculus
  - Programming & Databases
  - Business Knowledge & Problem Solving
- These are used in combinations in **3 separate professions**
  - Machine Learning
  - Data Analytics
  - Data Engineering
- **Data Scientists (**and so-called **Unicorn data scientists)** are now expected to have all 3 professional skillsets
- **Objective** of this course is to teach students how to do an End-2-End data science project from
  - From Problem definition, data sourcing, wrangling and modelling
  - To analyzing, visualizing and deploying & maintaining the models



How to learn Data Science

- **Use this course to learn data science**
- **College** bachelors, masters, phd – maths, statistics, computers, analytics, research
- **Online courses**, tutorials, books, blogs, youtube channels
- **Hackathons**, study groups, self guided projects, online free datasets
- **Start** with the main concepts and libraries
  - Programming – Python (or R), pandas, numpy, SQL
  - Maths – Statistics, Probability, Calculus, Algebra
  - Algorithms – Supervised/Unsupervised, Regression/Classification/Clustering, Deep Learning, NLP, etc
- **Learn** and apply theoretical knowledge using latest libraries and free datasets
  - You are expected to know the latest libraries, platforms and processes
  - New one's are released regularly and some are gaining wide adoption
- **Libraries to learn**
  - Common - scikit-learn, pandas, numpy
  - Deep Learning - keras, TensorFlow, PyTorch
  - NLP – NLTK/Word2Vec/Glove/BERT/GLP
- **Common Platforms to learn**
  - Data Science – Anaconda, RapidMiner, Tableau, etc
  - Cloud based - AWS Sagemaker, Microsoft Azure AI, Google Cloud AI, etc
  - Big Data – Apache Spark, Hadoop
  - Data Warehouse – Apache Hive, AWS Redshift
  - Databases - MySQL/PostgreSQL/MongoDB

Guidelines and Expectation

- **Part 1 will cover** the main principles/tools that are required for data science
  - Programming (python)
  - Data analytics (pandas)
  - Maths, Statistics and Probability
  - Data science process, data wrangling, main machine learning algorithms
  - How to visualize, evaluate and deploy models
  - End-2-End project
- It has python code examples for each algorithm and a full model building and deployment example
- **However, students are expected** to have some basic knowledge of:
  - Any programming language – Python/R, Java, JavaScript/PHP, etc
  - Or programming concepts – data types (integer, string), variables, conditional loops (if else, for), functions, list, array
  - Databases, SQL (Select, From, Where), Excel, Charts/Graphs
  - School level maths, statistics, probability and algebra
- Links to a huge collection of online resources (courses, tutorials, books, blogs, videos) will be provided throughout
- All **python code and data for examples** is in the below github location:
- https://github.com/datawisdomx/DataScienceCourse
- ***Parts 2,3,4 of the course*** *will cover Deep learning & NLP, Reinforcement Learning & Computer Vision and Data engineering techniques – to be published later*

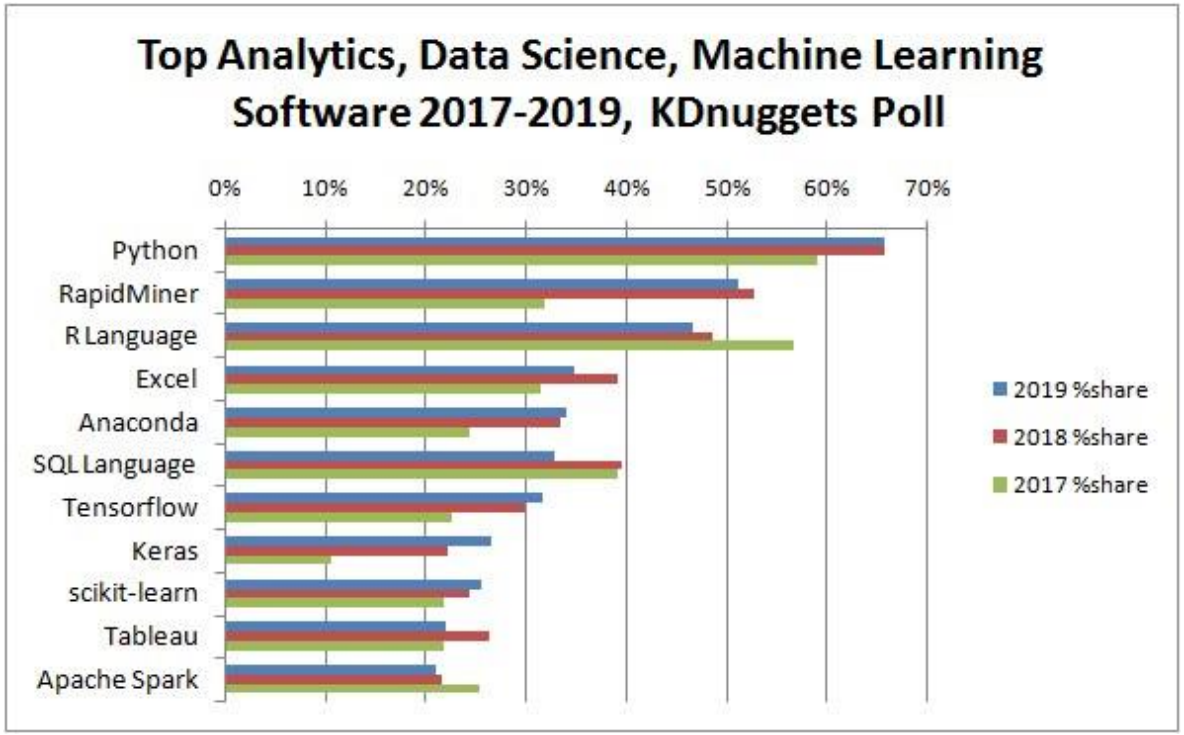
Chapter 1 – Programming - Why Python

- Python is an **interpreter** based language. You can execute the code immediately, line by line on the command prompt using python interpreter
- No need for compiling. Makes it a very good language for quick prototyping
- It is very simple to use with an English language based syntax
- It's the most widely used programming language for data science
- It easily integrates with multiple standard maths, statistics, data visualization and database libraries, most of them being written in python
- It supports all high-level programming features like OOPS, multi-threading, statistics, etc
- Python has a large global online community of contributors with a wide array of free resources

Useful books and websites

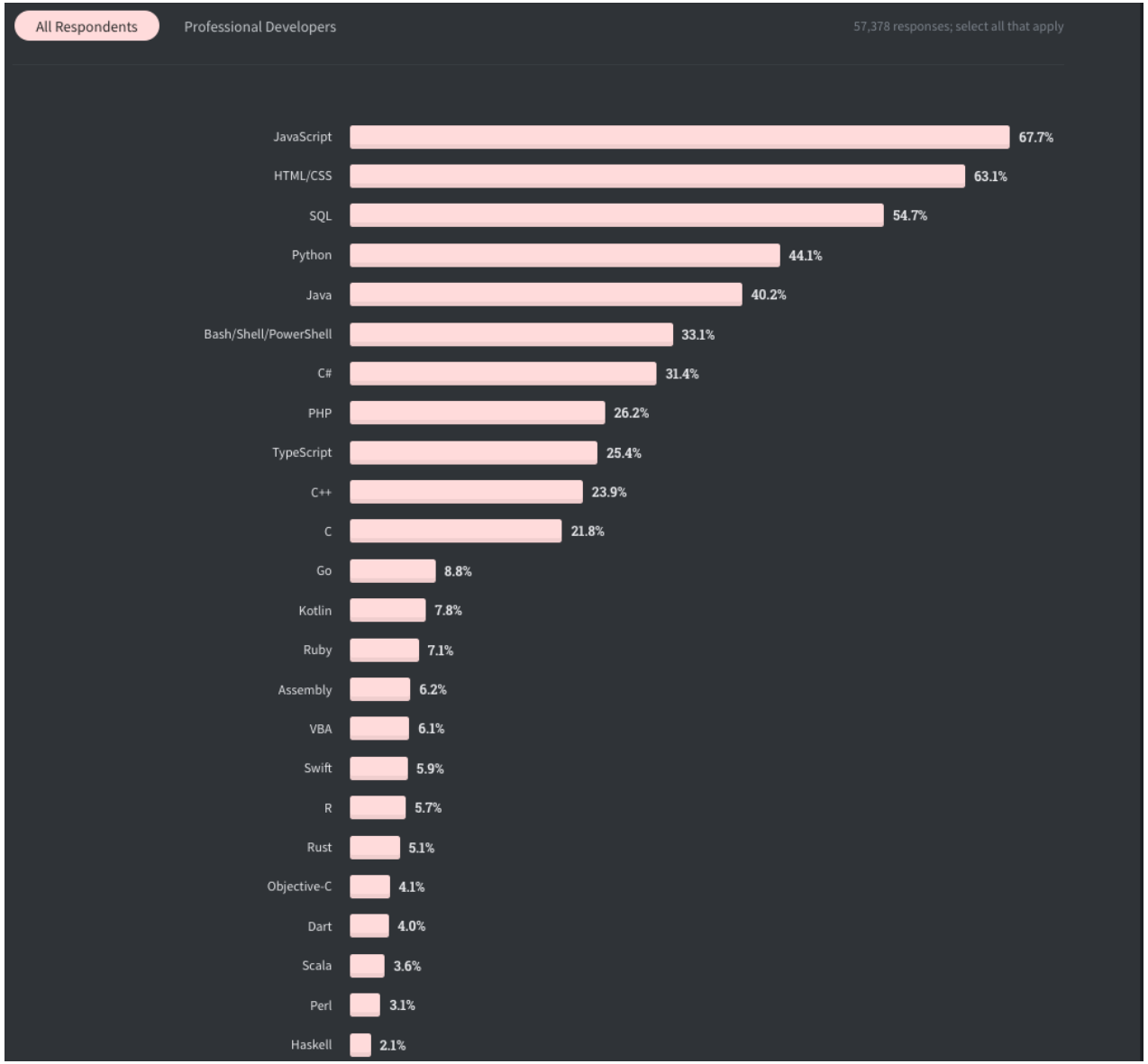
- Python guide tutorial - https://docs.python.org/3/tutorial/
- Beginner's book - http://greenteapress.com/thinkpython2/thinkpython2.pdf
- Comprehensive resource - https://docs.python-guide.org/intro/learning/
- Lots of online courses (Udemy, etc.), tutorials, youtube videos, blogs, stackflow, github

Chapter 1 – Programming - Python



Reference: *KDnuggets Analytics/Data Science 2019 Software Poll*

- **Python** is being used for this course as it's the most popular for data science and has the biggest collection of relevant libraries
- Chapter 1 is for python programming concepts
- Refer these links:
- https://www.kdnuggets.com/2020/06/data-science-tools-popularity-animated.html
- https://insights.stackoverflow.com/survey/2020#technology-programming-scripting-and-markup-languages-all-respondents



Reference: *Stack Overflow's annual Developer Survey 2020 poll*

Chapter 1 – Python - How to get started

- Use PIP install on command prompt to install the latest python package in your development environment: > ~/pip install python
- You can use **Anaconda** for an integrated development environment
  - It's free for individuals and one of the most popular dev env for data science
  - It comes with most required packages pre-installed and can be updated easily
    - Python, pandas, numpy, scikit-learn, matplotlib, Tensorflow, etc
  - You can create separate dev environments for different projects as required
  - For coding IDE's it has both Spyder and Jupyter
- **Spyder** is more functional with advanced editing, debugging, testing run-time features
- **Jupyter** is a more popular web-based notebook
- Easiest way to get started – download anaconda and start coding in python!
- https://www.anaconda.com/products/individual

Chapter 1 – Python - Basic syntax

- Begin with the famous “Hello, World!” statement used by all programming languages
  - Start python **$ python**. Type on the command prompt **>>> print(“Hello, World!”)**
  - It will print **>>> Hello, World!**
- Print() is python’s command to print the output to the screen
- Comments are preceded with **# This is a comment**
- Python uses new lines and indentation for completing commands and scope (for nesting)
  - No need for end of line semicolon or loops enclosed in brackets

```
if x > y:
    print(x)
else:
    print(y)
```

- All **python code and data for examples** is in the below github location:
  - https://github.com/datawisdomx/DataScienceCourse

Chapter 1 – Python - Data Types

- **Numbers** can be real (integers, float) or complex
  - Integers are non decimal values (no fractions). Can be +ve/-ve.
    - Eg: -74, -6, 0, 1, 3, 25, 100…
  - Floats have a decimal component (fractions). Can be +ve/-ve.
    - Eg: -35.5, -4, 0, 0.75, 3.2, 64.897, 1183.99…
  - Complex have an imaginary component ‘j’, as they cannot be expressed as real numbers
    - Eg: m = 792 – 68j
- Standard mathematical operations (+,-,*,/,%,etc) can be applied on different data types
  - Division returns floating no by default
  - **Power of** is denoted by ** (6**4 = 6*6*6*6)
  - It has a **math** module that provides access commonly used mathematical functions
- You can change the data type of a number by casting it
  - Int(83.5) will give 83
  - Float(98) will give 98.0
- Python code examples are given

Chapter 1 – Python- Data Types

- **Variables** are objects or containers for storing data values
- They can be declared and assigned without specifying the type
  - Unlike other programming languages
  - '=' is used to assign a variable
  - Variable can have any name
  - x = 10, val = 5.1, language = 'Python'.
    - x, val, language will be integer, float and string by default
  - You can assign more than one variable in a single command
    - a, b, c = 10, 5.3, 120.34
  - Perform math operations while assigning
- **Datetime** is a module (not data type) used for date & time functions
  - You can change data type to date and format year, month, day accordingly
  - Extract year, month, day, time as required using format codes
- Python code examples are given

Chapter 1 – Python - Data structures – List, Array

- **List** – Compound (Collection) data type that is used to group values as a list of comma separated values in square brackets [ ]
  - Indexed, changeable and allows duplicates
  - Eg: mylist = [1,3,6,8,9,9,15]. words =[‘car’, ‘fruit’, ‘ball’]
    - mylist[0] = 1, mylist[5] = 9. words[2] = ball.
  - Generally contains the same data type but can be a combination too (int, float, string)
  - Lists are mutable – values can be changed/moved. Unlike basic types like int
  - Functions to add/remove values are – append(), insert(), remove(), pop()
- **Indexing/Slicing**
  - List values can be retrieved or removed using index no in the list mylist[1]
  - Python indices start at 0 and slicing can be done using index no and colon ‘:’
  - mylist[start:end] is from start to end-1 index no
  - mylist[3:] is the list from 4th element to end of the list
  - mylist[:3] is the list up to and not including 4th element
  - mylist[2:4] is the list from 3rd to 4th element, excluding the 5th element (index 4)
  - mylist[:] is the entire list
  - Python lists allow –ve indices. So -1 starts from the end of the list
  - mylist[-2:] uses the concept of –ve index to give elements from -2 to the end
  - mylist[:-4] gives elements from start to -3 index element and excludes -4
- **Array** – python does not have a native array data type
  - Use lists to create multi-dimensional arrays
  - A 2D array is a 1D array of rows - list of lists
- Python code examples are given

Chapter 1 – Python- Data structures – Sets, Tuples, Dictionary

- **Sets** – Like lists but unindexed
  - Use curly brackets { }
  - Eg: myset = {'cow', 'cat', 'dog'}
  - Cannot use index to access elements
    - Eg: myset[0] is not allowed, will give an error
  - Only add/delete elements, no updates
  - No duplicate elements allowed
- **Tuple** – Like lists but unchangeable
  - Use round brackets ( )
  - Eg: mytuple = (2,4,6,8)
  - mytuple.add(10) will give an error
- **Dictionary** - changeable and indexed lists
  - Use curly brackets {}
  - Eg: mydictionary = {‘Name’: ‘Tom’, ‘Age’: ‘52’ , ‘City’: ‘London’}
  - Can be used for storing different data types (int, str, float) in one list
  - Accessed by using (key:value) pairs
  - Can loop through the dictionary using keys(), items(), values()
  - Some functions - get(), update(), pop()
  - A dictionary can contain a collection of dictionaries
  - No duplicate elements allowed
- Python code examples are given

Chapter 1 – Python- Data Types

- **Strings** are text contained in ‘ ’ or “ “
  - x = ‘Simple’ or x = “Simple” are same
- Strings are lists so you can use indexing (python indices start at 0)
  - x[0] = S. x[1] = i. x[5] = e
- Useful string functions: (entire list can be found in python online guide)
  - len() – gives the length of the string
  - strip() – removes leading/trailing spaces
  - split() – split the string into a list using given separator, whitespace by default
  - replace() – replaces a particular value in the string with a specified value
  - lower(), upper() – converts entire string to lowercase or uppercase
  - Concatenate – Use + to combine two strings into one
  - Cast to string type by doing str(‘42’) will make 42 a string
- Python code examples are given

Chapter 1 – Python - Operators

Maths

- Add +, Subtract -, Divide /,Multiply *, Power of **, Modulus %, Floor division //

Assignment – perform math operation by given variable and assign

- +=, -=, *=, /=, %=, //=, **=, etc

Comparison

- Equal to ==, Not Equal to !=, Greater than >, Less than <, <=, >=

Logical - for conditional statements

- And- if both true, Or- if one is true, Not- if neither is true
- Python code examples are given

Chapter 1 – Python- Control Flow Statements

- **If elif else** – conditional flow statement
  - Multiple conditions can be tested within the same if statement using elif and else
  - The if statement tests the main condition and depending on whether it is true or false
    - Use elif to check another if condition
    - Or use else to check the alternate condition of the statement
  - You can have nested if-elif-else conditions within each if or elif or else condition
- **While** - loop executes till the condition is true
  - Eg: a=1 and then print a while (a<5)
  - Use break to stop the loop earlier, but it is not considered good programming practice
- **For** – iterates over the items of a list or string
  - Python doesn't have start, stop and progression steps feature like other languages
  - You can use range function to create a progression sequence
  - You can use break and continue statements to break out of a loop
    - But it is not considered good programming practice
- **Range** – function that can be used to iterate over a sequence of numbers
  - Can specify start, stop and progression steps
  - Loop runs until Stop value -1 (stops before it)
- **List Comprehension** – A compact way to create lists based on the result of conditions, loops and operations
  - b = a if a is even. a is a list of numbers
  - Example shows long and list comprehension way
- Python code examples are given

Chapter 1 – Python- Functions, Error handling

- **Functions - User defined**
  - Block of reusable code that performs specific tasks
  - Use def (): command to create a function with parameters being passed to it
    - Pass data to be used by the function as parameters
    - Eg: def myfunction(a,b). a,b can be any data structure and type
  - The function performs the tasks and returns the result
  - Functions must be defined and executed before being called
  - Functions can call other functions, with results being passed from one to the other
- **Lambda -** A small anonymous function used for a simple operation on variables in a given scope
  - It is generally written in one line and performs one operation
- **Error Handling –** Use **try except** blocks for handling errors and exceptions
  - Errors – User input, syntax errors
  - Exception – Code execution errors. You can use the built-in exceptions or can be user defined
- Python code examples are given

Chapter 1 – Python- Read/Write - Console, File, Database

- **Console Read/Write**
  - Allows users to provide keyboard input
  - Display results to users on screen
  - Functions - Input(), print()
- **File Read/Write**
  - Allows programs to read/write data from/to files (txt, csv, etc)
  - Enables data processing across files/systems
  - Some functions - Open(), read(), write()
  - Open() parameter ‘r’ reads, ‘a’ appends and ‘w’ overwrites the file
- **Database Read/Write**
  - Allows programs to read/write data from/to databases (MySQL, MongoDB, etc)
  - Enables data processing on scale
  - Install ‘mysql.connector’, the MYSQL driver that allows python to r/w to it
  - Once connected to db, user can execute all SQL commands like – CREATE TABLE, INSERT, SELECT FROM WHERE, DELETE, JOIN, ORDER BY, etc
- Python code examples are given

Chapter 1 – Python - Object Oriented Programming OOPs

- Python supports all object oriented programming principles **OOPs**
  - Useful for creating modular programs that are easily extended and minimise repetition
- **Class** is an extensible template for combining properties (variables, data) and methods (functions)
  - Use **class:** keyword
  - Eg: Create the **class Book** with some methods and properties
- **Object** is an instance of a class which contains its own state attributes
  - Properties have local scope within the object
  - When you create an object the method and properties belong to it
- Everything in python is an object - string, list, functions, etc
- **__init__** is a built in function that is called for every class instance
  - Used to assign values to properties or perform other operations when an object is created
  - They are reserved methods so cannot be used for user defined functions
- ***Self*** parameter references the current instance of the class (object)
  - It is generally used to initialize variables
  - It is unique to each instance and is used to reference the variables of the instance
- **Inheritance** allows creating a new class (child) by inheriting from an existing class (parent)
  - Child class adds its own methods & properties
  - This allows the extension of existing classes to create new functionality
  - Eg: **class Fiction** inherits from **class Book** and adds methods of its own
- **Namespace** – Defines the scope of names (objects, functions, properties)
  - Functions have their own local namespace which is created when they are called
  - It maps names to objects. This allows names to be same across different classes, in the same or different modules
  - It is done by using the hierarchical namespace dot notation
  - Eg: **class Book** has **method details()**. It will be identified by Book.details
  - There can be another **class Pen** with **method details()** in the same module. It will be identified by Pen.details
- Python code examples are given

Chapter 1 – Python – Modules, Packages

- **Modules**
  - A file containing executable statements, functions and variables
  - Has a .py extension
  - Useful for large programs that can be stored and executed as files
  - Can be run on its own or **imported** into another module
  - They are imported into other modules for using their functionality
    - Instead of creating that again in the new module (extensibility)
  - Python has no main() function, which indicates the entry point in the program
    - There is no need for that in python as it executes the code line by line starting at line 0
  - Two different modules' classes can have the same method, property names
- We have already seen some standard modules like math, datetime, etc
- **__name__** is a built-in variable which evaluates to the name of the current module
  - If source file is executed as the main program, interpreter sets __name__ = "__main__"
  - If source file is imported from another module, __name__ = module's name
- Modules can be **executed** as **scripts** by using the **python** module.py command
  - interpreter sets __name__ = "__main__"
- **Packages**
  - Structure python's modules namespace hierarchically using dot notation
  - Just like a directory/subdirectory/file structure
  - Eg: There are **modules Student** and **Teacher**. We need them together. So we can create a **package A** which contains both modules
  - **A.Student** means Package A's module Student. **A.Teacher** means Package A's module Teacher.
  - Now we can have another **package B** which contains **module Student**. So **B.Student** means package B's module Student.
  - **Module Student** functionality will be **different in packages A & B**
- The **__init__.py** files are required to make Python treat directories containing the file as packages
  - This file is placed inside the main package directory
  - It contains the initialization code, for example, indicating the files to be imported
- Python code examples are given

Chapter 1 – Python- Sort Algorithm, Recursion, Regex

- Some useful general programming concepts - sort algorithms, recursion and regex
  - They are widely used in programming, so its beneficial to learn about them
- **Sort algorithm** – Arranges data in ascending / descending numerical or lexical order
  - Bubble sort is simple but inefficient
    - Iterates repeatedly through the list, comparing adjacent elements, swaps them if in wrong order
  - Better ones – merge, insertion, quick...
- **Recursion** – Solve problems by combining solutions to smaller versions of the same problem
  - Function calls itself in a loop storing the results in a Stack which are returned in LIFO(Last In First Out) order once the loop ends
  - Uses a base condition to terminate the loop, else it will continue indefinitely
  - Eg: Factorial !. 5! = 5*4*3*2*1 = 120
- **RegEx** – Regular Expression used to form a search pattern in a string
  - Python has a built-in module **re** for this
  - Search() checks entire string for a specified pattern
  - Match() checks only at the beginning of the string
  - Special characters determine how to interpret other characters around the regular expression
    - ^str - string starts with ‘str’
    - str+ - one or more occurrences of ‘str’
  - Findall() - all instances of the string ‘str’
- Python code examples are given

Chapter 2 – Data Analytics using Pandas

- Pandas is an open source, easy to use, high Performance Data structure and Analysis library for python
- It can handle most data sources/types – databases/files/json, array/string/datetime/dictionary
- Built for python, so integrates easily with it. Comes pre-installed in anaconda platform
- Very efficient for performing in-memory operations on data structures
- Data Analytics involves analysing large amounts of data in different formats and from different sources to make business decisions
- It is widely used for machine learning due to its ease & flexibility in performing data manipulation and analysis tasks on a variety of data formats
- Data science uses the data generated from data analytics to make business predictions

Useful books and websites

- Comprehensive resource - https://pandas.pydata.org/docs/getting_started/tutorials.html
- Beginners book - https://pandas.pydata.org/docs/user_guide/cookbook.html#cookbook
- Cheatsheet - https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf
- Lots of online courses, tutorials, youtube videos, blogs, stackflow, github

Chapter 2 – Pandas – Data Structures - Series

- **Pandas** two main data structures are Series and DataFrame
  - Series is 1-Dimensional and DataFrame is 2-D
  - Each column in a dataframe is a series
  - **Numpy** (scientific computing python library) will be used in some examples
- **Series** – 1-D structure (list) that can contain any data type – int, float, string, dictionary, python object, etc
  - Axis (row or column) labels are called index - which can be numeric or text
  - s = Series(data, index=index)
  - If no index is given, a numeric value equal to length of data is assigned
  - Values can be accessed using the index
  - Basic maths operations can be done directly or numpy functions can be used which accept series data type
  - Operations can be done on an entire series without looping through each value

Chapter 2 – Pandas – Data Structures - DataFrame

- **DataFrame** – 2-D structure that can contain multiple rows, columns of different data types like a csv file or database table
  - It can contain a dictionary of data types, series, numpy arrays, other dataframes, etc
  - df = DataFrame(data, index=index, columns=cols)
  - Axis (row, column) labels are called index, which are used to access values
  - Maths and numpy operations can be performed directly without looping through each row, column
  - New columns can be created as the result of operations on existing columns
  - Useful View properties/functions –
    - dtypes - datatype of all columns
    - index - range of values
    - columns - (names),
    - head() – view top n rows, 5 default
    - tail() – view bottom n rows, 5 default
  - Copy() – Create a copy of a dataframe

Chapter 2 – Pandas – View, Slicing, Functions

- **All following functions and operations are applicable to Series and Dataframes**
  - Explanation/examples are given for dataframes as its more commonly used
- Useful **View properties/functions** –
  - axis - 0 means rows or index, 1 means columns
  - dtypes - datatype of all columns
  - index - range of values (rows)
  - columns – column labels (names)
  - head() – view top n rows, 5 default
  - tail() – view bottom n rows, 5 default
  - copy() – Create a copy of the dataframe
- **Selection/Slicing** – Using axis labels, index number, conditions
  - loc[:,'value'] – select by axis label
  - iloc[2,3] – select by index number
  - [2:] – select slices by rows
  - Select a subset of the df using column labels [['column1', 'column2']]
  - Select a subset of a df using conditions
  - Add new (row, column) values using loc or iloc property
- **Functions –** Applied on the entire dataframe at once rather than looping through each row and column
  - Conditional operations
  - Apply() – small local functions
  - Statistics – Sum, mean, variance, etc
  - Use numpy functions
- https://pandas.pydata.org/docs/reference/frame.html

Chapter 2 – Pandas – Missing data, Merge

- **Missing data –** pandas uses numpys **NaN** constant to identify missing data
  - dropna() – drop axis label (row or column) with missing values
  - fillna() – fill missing values with a given value or function result
    - Use method = ‘bfill’ or ‘ffill’ to fill all missing data with next or previous valid value
  - https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.fillna.html
- **Merge** multiple dataframes easily using set or database like JOIN operations
  - **concat()** – concatenates multiple dataframes using union or intersection (outer or inner)
    - ignore_index=True, if index values should not be used and reset instead
  - **merge()** – merges 2 dataframes with database like JOIN (inner, outer, left, right) operations
    - on = column to join on, how = join type
    - Adds _x,_y suffix to overlapping columns in the 2 dataframes
  - https://pandas.pydata.org/docs/reference/api/pandas.merge.html#pandas.merge

Chapter 2 – Pandas – Text data, Iteration, Sorting

- **Text data is** stored as an object or string in pandas
  - Convert it to **string** datatype as it provides most string processing functions accessed via **str** attribute
  - Common functions like - Split(), replace(), count(), len(), strip(), get(), lower(), upper(), cat(), concat(), etc
  - Use to_string() method to convert series or dataframe to string data type
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/text.html
- **Iteration** – Series behave as arrays and dataframes as a dictionary (key, value)
  - Iterating over series gives values, dataframes give column labels(names)
  - items(), iterrows() provide dictionary behaviour
- **Sorting** can be done by index label, column value or both
  - sort_index() – sorts using index value
  - sort_values() – sorts using column value

Chapter 2 – Pandas – Groupby, Casting

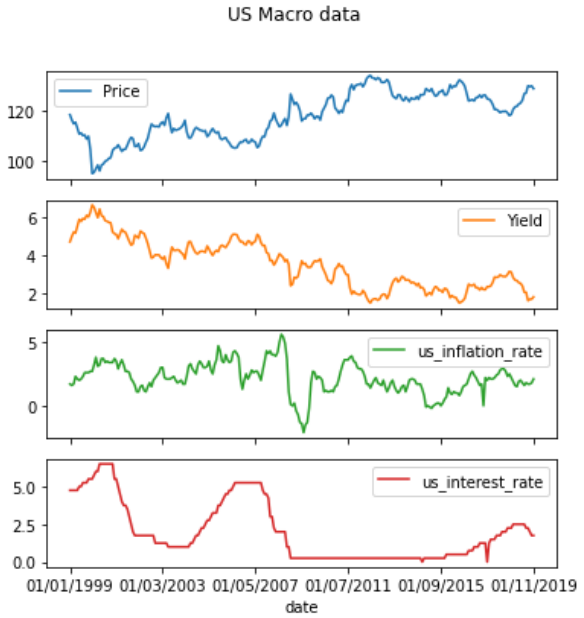
- **Type casting or conversion functions change** dataframe objects to desired datatype
  - Useful for different file types or dataframes containing undefined formats or objects
  - **astype()** – casts datatype to desired type like int, string, datetime
  - **to_datetime(), to_numeric()** - Some flexible functions to convert a dataframe to a specific type
- **Groupby() –** involves splitting the dataframe into different groups based on some criteria, then applying transformation functions on each group and finally combining the results into one dataframe
  - Grouping is done using a list of columns, levels, dict, series, functions or their combination as keys
  - Transformation can be an aggregation operation or any required function
  - **Eg**: Daily Stock Index price for a few days across 2 months is given
    - Get monthly statistics (change timeseries to monthly 'M')
  - **.dt** is used for accessing datetime properties
- **Agg**() – Perform multiple aggregation operations in a single function
  - Can be used with groupby

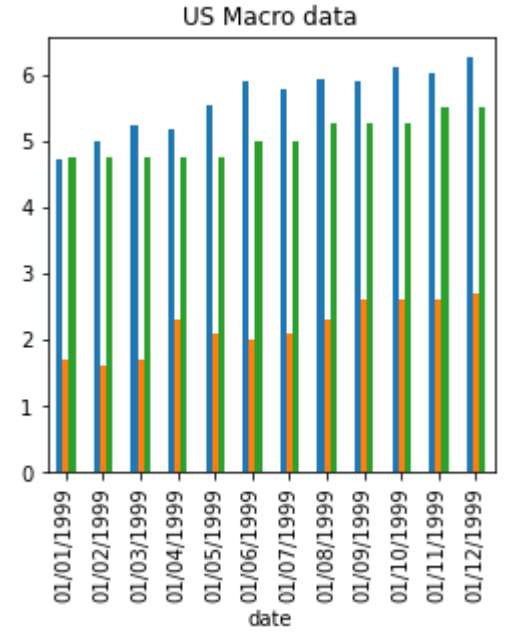
Chapter 2 – Pandas – Time Series, Sampling

- **Time Series** – data indexed by sequential intervals of date and time (d/m/y h:m:s)
  - Extensive range of functions - formatting, converting, resampling, datetime offset, etc
  - Some commonly used functions are explained
    - Date_range(), Timedelta(), offsets(), etc
- **Resample**() - Resamples dataframe time series using a different frequency
  - Requires a Datetime, Timedelta or Period Index
  - Eg: resample daily data to monthly
- **Rolling**() – Get rolling statistics for a dataframe time series (time based window operation)
  - Window represents the time offset (looking backwards) for which to aggregate values
  - Eg: 3 day moving average, window=3

Chapter 2 – Pandas – Read/Write, Plot

- **Read/Write functions** – Pandas I/O API provides reader & writer functions for various file and data formats
  - **Read** - read_csv(), read_json(), read_fwf(), etc
  - **Write** - to_csv(), to_json(), to_pickle(), etc
  - All functions provide multiple arguments to parse different formats, file path, encoding, etc
  - https://pandas.pydata.org/docs/user_guide/io.html
- **Plot** – Pandas provides common visualisation's using its plot() method
  - plot() is a wrapper around matplotlib.pyplot's plot() function (popular python visualisation library)
  - It can easily plot most common and scientific charts – line, bar, pie, scatter, 3-D, subplots, overlaying, etc
  - See the guide for comprehensive examples
  - https://pandas.pydata.org/docs/user_guide/visualization.html#basic-plotting-plot





Chapter 2 – Pandas – SQL Operations

- Pandas can be used to perform most **SQL** type operations
  - Once data is loaded into a dataframe, it can perform most SQL operations:
  - SELECT, FROM, WHERE, GROUPBY, JOIN, LIMIT, HAVING, nested queries, conditions, etc
  - **Eg1**: Calculate and print value for each item. value = price * quantity
    - SELECT item, price * quantity as value FROM df1;
  - **Eg2**: Select items where price>20 and quantity<10
    - SELECT * FROM df1 WHERE price>20 AND quantity<10 ;
  - **Eg3**: Select items with lowest 3 prices
    - SELECT * FROM df1 ORDER BY price ASC LIMIT 3;
  - **Eg4**: Left Join two tables on item.
    - SELECT * FROM df1 LEFT JOIN df2 ON df1.item = df2.item;
  - **Eg5**: Right Join two tables on item.
    - SELECT * FROM df1 RIGHT JOIN df2 ON df1.item = df2.item;
- See the guide for comprehensive examples
  - https://pandas.pydata.org/docs/getting_started/comparison/comparison_with_sql.html

Chapters 3 – Statistics

- Students are expected to have some basic knowledge (School level) of statistics
- This chapter will cover the main statistics concepts required for data science

Useful books and websites

- Beginners book - https://greenteapress.com/thinkstats/thinkstats.pdf
- Comprehensive book - https://web.stanford.edu/~hastie/Papers/ESLII.pdf
- Useful sites, youtube videos
  - https://www.mathsisfun.com/
  - https://mathworld.wolfram.com/
- Lots of online courses (Udemy, MOOC's), tutorials, blogs, etc

Chapter 3 – Statistics - Introduction

- **Statistics** involves data collection (sample) and its analysis using two main techniques – descriptive & inferential
- **Population** is the set of possible values (actual or hypothetical) of the data being considered for analysis
- A **Sample** is a subset of the data selected randomly from the population to estimate its characteristics
- **Descriptive** – summary statistics that use the given sample to describe its distribution using two main properties – central tendency (mean, median, mode) & variability (variance, skew, outliers, quartiles, etc)
- **Inferential statistics** deduces properties of a population from a small sample by building a model using statistical techniques like estimation, hypothesis testing, etc
- Descriptive statistics is concerned with only the observed data while inferential statistics considers the entire population by building a model from a small sample
- **Univariate analysis** describes the distribution of a single variable using descriptive statistics
- **Bivariate analysis** describes the distribution and relationship between two or more variables using descriptive statistics and measures like Correlation, Covariance, Regression, Scatter plot etc
- **Multivariate analysis** involves the observation and analysis of more than one outcome variable. It applies multiple techniques to measure and understand the relationship between multiple variables in a sample. Eg: Clustering, PCA, LDA, Contingency tables, etc
- **Exploratory Data Analysis (EDA)** involves analysing the data sample, primarily visually, by summarising its main characteristics. It involves more descriptive statistics than inferential

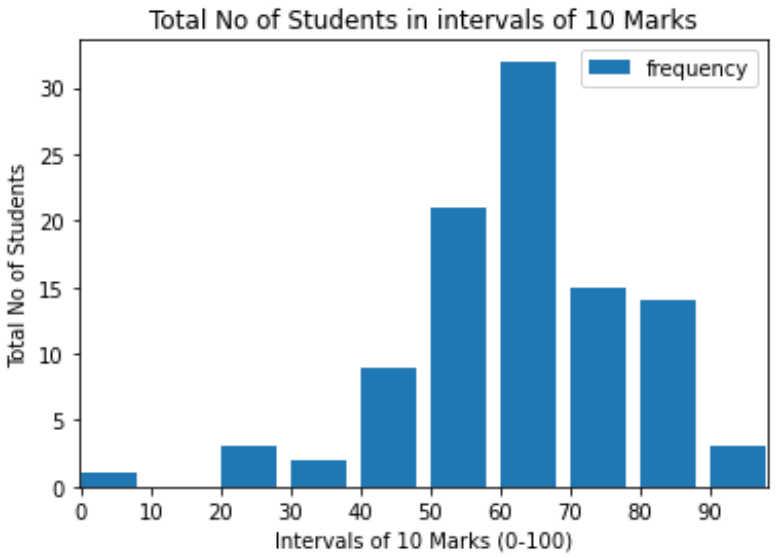
Chapter 3– Statistics– Descriptive– Central tendency, Variability

- Consider **Descriptive statistics (Univariate Analysis)** for a sample of n values x_i, where i = 1 to n
- **Central tendency** – measures the average or typical value of the sample (mean, median, mode)
  - **Mean** is their sum divided by n. $\bar{x} = (\sum_{i=1}^{n} x_i)/n$
  - **Median** is the middle value, after sorting in ascending or descending order
  - **Mode** is the value that occurs most often
- **Variability** – measures dispersion (spread) of the sample from its mean (variance, minimum & maximum values, skewness, kurtosis)
- **Variance** measures how far each each number is from the mean
  - Variance = $\sigma^2 = [\sum_{i=1}^{n}(xi\text{-}\bar{x})^2]/n$
  - It is always positive as it squares the difference
  - The benefit of squaring is that it prevents the +ve and –ve values from cancelling each other
  - It gives higher weight to outliers, as their spread from the mean will be higher along with being squared
- **Standard deviation** is the square root of variance
  - Standard Deviation = σ
  - It has the same unit as the sample data, so its better for comparison
- **Bessel's correction** is the use of n-1 instead of n in the formula for variance and standard deviation
  - It partially corrects the bias in their estimation from the sample
  - It results in a higher MSE (Mean Squared Error)
  - Use it only when you have a sufficiently large sample
- **Examples** with python code are given
  - Numpy functions are also given for ease of calculation

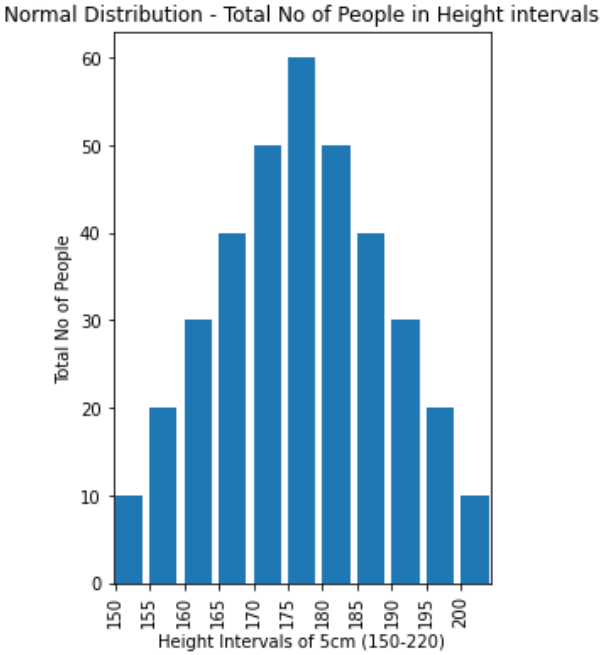
Chapter 3 – Statistics – Descriptive - Distributions

- **Distribution** is a function that describes a variables possible values and how often they occur
  - Used commonly for descriptive statistics
- A **Frequency** distribution displays the frequency of various outcomes of a sample in a table or graph
- A **Histogram** is the most common frequency distribution, used for numeric and categorical data, displayed as a bar chart
  - It splits the entire range of values of the sample into non-overlapping intervals (bins, buckets)
  - It then counts the number of values in each interval
- **Eg**: Frequency distribution of 100 students in a class with marks in statistics between 0 to 100 is given
  - Data is split into intervals of 10 marks and displayed as a histogram
  - Python code is given
- **Normal distribution** is a probability distribution of a real-value random variable symmetric around the mean μ with standard deviation σ
  - 68% of distribution values are within +/- 1σ of μ. 28% values within +/- 2σ of μ. 4% values within +/- 3σ of μ
  - It is also called a bell curve and widely used in various fields
    - Eg: test scores, finance, biology, social sciences (people height), etc
  - A standard normal distribution (μ=0, σ=1) is the most common
  - ***Machine learning requires standardized data for most algorithms*** as large values can skew the results by giving them higher weights

No of Students	Marks
1	9
3	21
2	30
5	40
4	45
12	50
9	55
17	60
15	65
11	70
4	75
10	80
4	85
2	90
1	95
100	



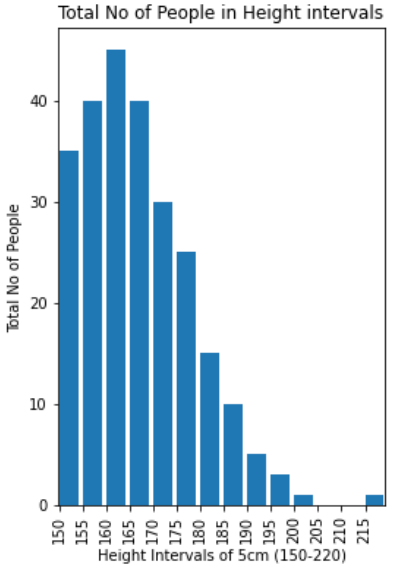
noofpeople	height
10	150
20	155
30	160
40	165
50	170
60	175
50	180
40	185
30	190
20	195
10	200
360	



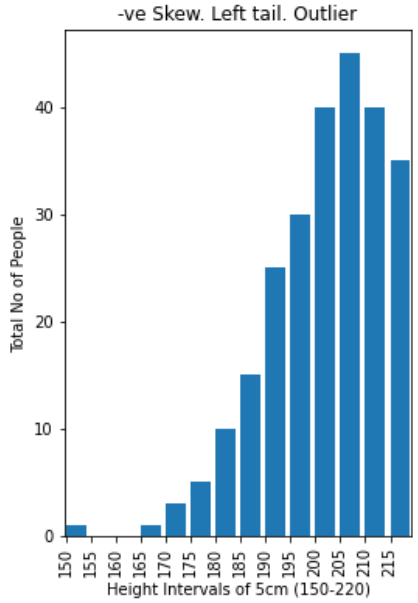
Chapter 3– Statistics– Descriptive– Skewness, Tails, Outliers

- **Skewness** is asymmetry of the distribution about its mean
  - Positively skewed - right tail is longer. Mass is concentrated on left side
  - Negatively skewed - left tail is longer. Mass is concentrated on right side
- **Tail** indicates tapering of the data from the mean (outlier frequency)
  - Can be long or fat
- **Outliers** - extreme values well outside the overall distribution
- **Kurtosis** further describes the shape of the distribution
  - It measures the combined weight of the tails (outliers) relative to the centre of the distribution (tailedness, not peakedness)
  - Higher value means more extreme outliers
  - It affects the outcomes of the analysis
  - It's +/- 3 for a normal distribution
- **Effect Size** measures the strength of the relationship between two variables in a population
  - Effect could be a statistic like correlation coefficient r, coefficient of determination r^2, mean difference, etc
  - **Cohen's d** - Difference between two populations can be described by the difference between their means
  - **d** = $(\bar{x}_1 - \bar{x}_2)/s$. $\bar{x}_1$, $\bar{x}_2$ are the means of the 2 populations. s is the standard deviation of the combined populations

Persons	Height
35	150
40	155
45	160
40	165
30	170
25	175
15	180
10	185
5	190
3	195
1	200
0	205
0	210
1	215
250	



Persons	Height
1	150
0	155
0	160
1	165
3	170
5	175
10	180
15	185
25	190
30	195
40	200
45	205
40	210
35	215
250	



Chapter 3 – Statistics – Descriptive – Percentile, Range

- **Percentile** is the value ***below which*** a percentage of values in the frequency distribution lie
- It is a set of 99 values that split the ordered distribution data into 100 equal groups
- 99^{th} percentile is the maximum possible
- 1^{st} percentile is the minimum. There is no 0^{th} percentile
  - Eg: 45^{th} percentile means the value below which 45% of values in the distribution fall
- **Percentile Rank** of a value is the percentage of values in the frequency distribution that are less than it
  - Eg: 45^{th} percentile means the values percentile rank is 45
- **Eg:** Tests like GRE give your actual score and percentile
  - Python example of a student marks distribution is given

All students marks = [35,40,28,57,78,63,70,85,92,80,90]
Student marks = 70
Percentile Rank of student is: 45
Percentile is: 45^{th} percentile

Value	28	35	40	57	63	70	78	80	85	90	92
Percentile	1	9	18	27	36	45	55	64	73	82	91

- **Range** is the difference between the largest and smallest values in a sample data set
  - Data set might not be in order and it can have repeating numbers

x = [1,3,2,5,4,12,9,17,15]
Range (x) = 17-1 = 16

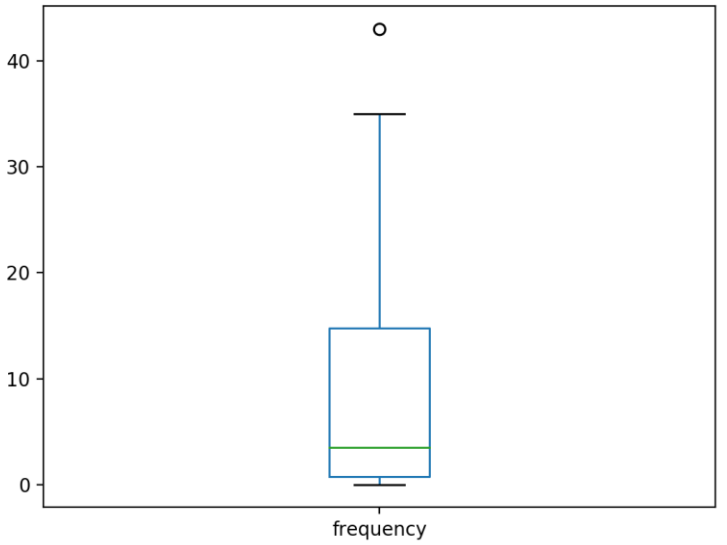
Chapter 3 – Statistics – Descriptive – Quartiles, Boxplot

- **Quartiles** are 3 numbers that divide the sample into 4 equal parts (quarters)
  - It is useful for understanding non-symmetric or skewed data and how its distribution is spread around the centre
  - Data must be ordered for calculating the quartiles
  - Q1, Q2, Q3 are the 1st, 2nd and 3rd quartiles
  - Q1 (25th percentile, lower quartile) represents the data range between 0-25% of the values
  - Q2 (50th percentile, median) represents the data range between 25%-50% of the values
  - Q3 (75th percentile, upper quartile) represents the data range between 50%-75% of the values
  - Remaining data is in the 4th quarter
- **Boxplot or Whisker plot** is used to represent the five number summary of the sample
  - Largest, smallest, Q1, Q2 and Q3 values
  - It has a box and 2 whiskers
  - The box is drawn from Q1 to Q3, with a line in the middle for the sample median
  - Whiskers extend out from the box to lowest and greatest values
  - Any outliers not included in the plot are shown as circles
  - **Interquartile range = Q3-Q1**
- For a given frequency distribution
  - Pandas describe() gives the 3 quartiles
  - Boxplot shows them visually along with the skew and outliers

25%	25%	25%	25%
0-25%	25%-50%	50%-75%	75%-100%
	Q1	Q2	Q3

'frequency': [1,1,0,0,4,7,10,20,24,35,43,29,13,8,3,0,0,0,1,1]

	frequency
count	20.000000
mean	10.000000
std	13.182924
min	0.000000
25%	0.750000
50%	3.500000
75%	14.750000
max	43.000000



Chapter 3 – Statistics – Inferential - Estimation

- **Inferential statistics** consider the entire population by building a model of it from a small sample
- **Estimation** is the process used to infer properties about a population from a sample taken from it
  - It can be done in two main ways – **point** and **interval** estimate
- **Point estimate** – get the best estimate of a single parameter of the population by calculating it from the sample
- **Estimator** is the statistic used. For example, sample mean $\bar{x}$ can be used to estimate population mean μ
- **Error** determines how far off the sample estimated value is from the population actual
  - Error = $\bar{x}$ - μ
- Depending on the population and statistic used we try and minimize or maximise the error by using multiple random samples which might give a wide range of estimated values
- To choose the best value of the estimator we use measures like MSE, RMSE, MLE
- **MSE** – Mean Squared Error is the average of the squares of the errors
  - MSE = $[\sum_{i=1}^{n}(\bar{x}_i - \mu)^2]/n$
  - We want to minimize MSE
  - Closer it is to 0, better the estimate
- **RMSE** – Root Mean Squared Error is the square root of MSE
  - It has the same unit as the data and penalizes large errors
- **MLE** – Maximum Likelihood Estimator will be covered later

Chapter 3 – Statistics – Inferential – Intervals

- **Interval estimate -** get an interval of possible values for a population parameter from the sample estimate
- **Z-score** or standard score is the no of standard deviations by which the raw value of the datapoint (x) is above or below the population mean
  - It is different from standard deviation which measures the variability within the dataset
  - z = (x - μ) / σ. μ, σ are the population mean, std dev
- **Z-test** is used for distributions which can be approximated by a normal distribution
  - It is performed using the z-score and tests the mean of the distribution
- **Confidence Interval** is a type of interval estimate
  - It has a confidence level associated with it indicating % of the interval's estimated values containing the population parameter's actual value (Mean, Standard Deviation SD, etc)
  - The confidence level is assigned before the estimation is done
  - 99%, 95% and 90% confidence levels are the most commonly used
  - For a normal distribution they indicate +/- 1 SD, 2 SD and 3 SD's from the mean
- **CI =** $\bar{x} \pm Z * \frac{\sigma}{\sqrt{n}}$
  - $\bar{x}$ - sample mean, σ - standard deviation of sample or population, **n** – no of sample observations
  - **Z** – value from the z-score table (standard normal distribution table) for the confidence interval
  - The z-score table is easily available online and in statistics books
- Eg: A random sample of 100 students' test scores out of a population of 1000 has $\bar{x}$ = 53 and σ = 31
  - For 95% CI, z-value = 1.96. So CI is 53 $\pm$ 6.
  - It means that there is a 95% chance that the test score of students will lie between 47 and 59

Chapter 3 – Statistics – Inferential – Hypothesis Test

- **Hypothesis testing** applies statistics on a sample to test an assumption about a population parameter
  - It is similar to proof by contradiction
  - To prove a statement is true assume it is false
  - If the assumption is proven to be false, we can conclude that the original statement must be true
- **Hypothesis test** consists of the following steps
  - Formulate the null and alternative hypothesis (assumption)
  - **Null hypothesis (H_0)** – hypothesis that chance is responsible for the observations in the sample
    - False statement
  - **Alternative hypothesis (H_a)** – hypothesis that observations in the sample are due to a real effect
    - It is contrary to the null hypothesis
  - Decide on the **test statistic** (mean, std dev, correlation, etc) to determine if the null hypothesis is true
  - Calculate the **p-value** for the null hypothesis - probability of seeing the apparent effect if the null hypothesis is true
  - Select the **significance level α**, $0 \leq \alpha \leq 1$ the threshold below which the null hypothesis is rejected
    - $\alpha = 0.05$ (5%) is commonly used
  - **Reject the null hypothesis and accept the alternative hypothesis if $p \leq \alpha$**
  - Smaller the p-value, more the evidence that null hypothesis is false
- **Errors** – The null hypothesis can be wrongly rejected depending on the p-value used, resulting in 2 types of errors
  - Type I error – rejecting a true null hypothesis (False Positive)
  - Type II error – accepting a false null hypothesis (False Negative)

Chapter 3 – Statistics – Inferential – Hypothesis Test

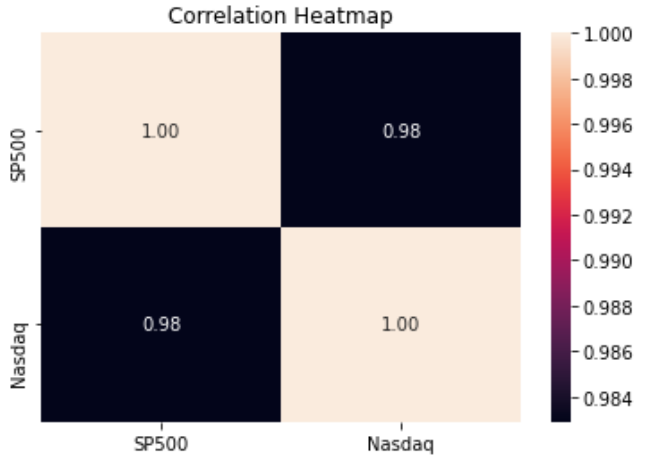
Example – Consider the average annual returns of a multi-asset mutual fund over 20 years

- It claims to make at least 15% average annual returns during the 20 years
- **Null hypothesis (H_0)** – Assume that the mutual funds claim is TRUE, average annual returns $\geq$ 15%
- **Alternative hypothesis (H_a)** – Mutual funds claim is FALSE, average annual returns < 15%
- **Test statistic** – Average (mean) annual returns of the fund
- **Calculate the probability (p-value)** for multiple random samples of 5 years with average annual returns $\geq$ 15%
- **Test if p $\leq$ 0.05** (α = 0.05 or 5% significance level)
- **Reject the Null hypothesis if TRUE and accept the Alternative hypothesis**
- **Calculation logic**
  - Take 10 random samples of 5 years
  - Calculate the average annual return for each sample of 5 years
  - Calculate the total no of samples with average annual return $\geq$ 15%
  - Divide by 10 to get the probability p = total no of true cases / total no of samples
  - Check if p $\leq$ 0.05
- This logic can be extended to evaluate multiple mutual funds across multiple factors like return, risk, portfolio composition, etc
- The results can then be improved by iteratively dropping low importance factors
- Overall objective is to build a model to evaluate fund managers across relevant factors

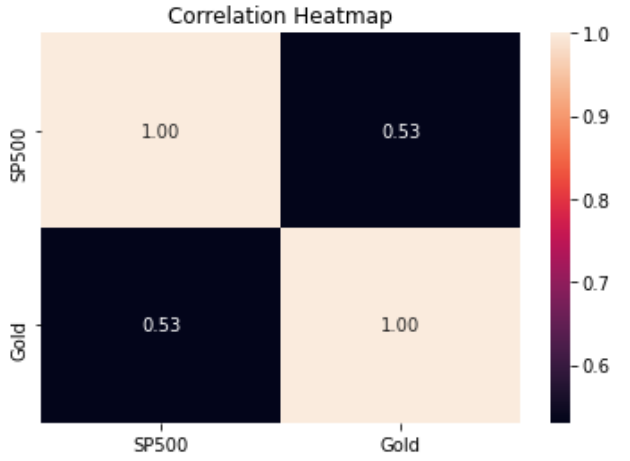
Chapter 3– Statistics– Bivariate Analysis- Correlation, Covariance

- **Bivariate analysis** describes the relationship between two variables using descriptive statistics like Correlation, Covariance, Regression, etc
- **Correlation** indicates a statistical relationship between two variables in a population, not necessarily causal
- **Pearson's correlation coefficient** ρ measures the linear relationship between 2 variables (x,y)
  - $\rho_{x,y} = \sum_{i=1}^{n}[(xi-\bar{x})*(y_i-\bar{y})]/ \sigma_x * \sigma_y$. -1<ρ<1.
  - Higher the value of ρ (rho), stronger the relationship (+ve or –ve relationship)
  - It uses **least squares fit** method to find the best fit line through the sample
  - It minimizes the difference between actual and expected values
  - Eg: Relationship between different asset prices, commodity demand and price, etc
- **Covariance** measures the joint variability of the two variables (x,y)
  - It can be +ve or –ve
  - $Cov_{x,y} = \rho_{x,y} * \sigma_x * \sigma_y = \sum_{i=1}^{n}[(xi-\bar{x})*(y_i-\bar{y})]$
  - It measures the strength of the correlation between the 2 variables
- Python example with correlation and covariance heatmap for different asset price pairs is given
- More detailed example with full dataset is available at https://github.com/datawisdomx/Monthly-Asset-price-relationship-to-Macroeconomic-Data

Correlation between S&P500 and Nasdaq



Correlation between S&P500 and Gold



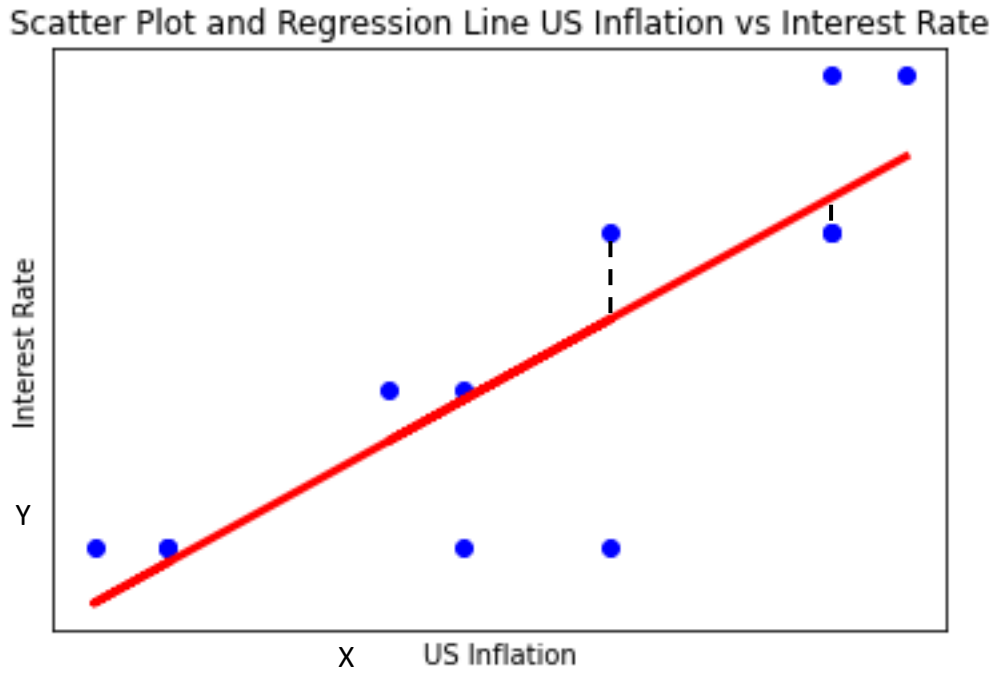
Chapter 3– Statistics– Bivariate Analysis- Regression

- **Regression** is a statistical process for estimating the relationship between a dependent variable Y and one or more independent variables $X_{1..n}$ in a sample
  - It can be simple (one X) or multiple (more than one X) regression
- Regression models Y as a function of X_i and β. Y = $f(X_i, \beta)$
  - $\mathbf{Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \varepsilon}$
  - β represents the unknown parameters
  - ε is the additive error for the deviations of the observed sample values from the population actuals (statistical noise)
- The goal is to estimate a function $\widehat{Y}$ that closely fits the data
  - $\widehat{Y} = \widehat{\beta_0} + \widehat{\beta_1} X_1 + \ldots + \widehat{\beta_n} X_n + \varepsilon$
- Models like **Ordinary Least Squares OLS** are used to estimate β which minimizes the **Residual Sum of Squares (RSS)**
  - **RSS** = $\sum_{i=1}^{n} (Y - \hat{Y})^2$
- Geometrically it is the sum of the squared perpendicular distance of each sample point from the best fit line passing through them
- Closer the points are to the line, better the fit
- **Goodness of fit R^2** measures how well the model fits the sample
  - It is the proportion of variance in the dependent variable that is predictable from the independent variables
  - **$R^2 = 1 - (RSS/TSS)$**
  - TSS = $\sum_{i=1}^{n} (Y - \bar{Y})^2$ is Total Sum of Squares
- Basic example is given. Detailed python examples will be discussed in Chapter 8 Regression

Chapter 3 – Statistics – Bivariate Analysis- Regression

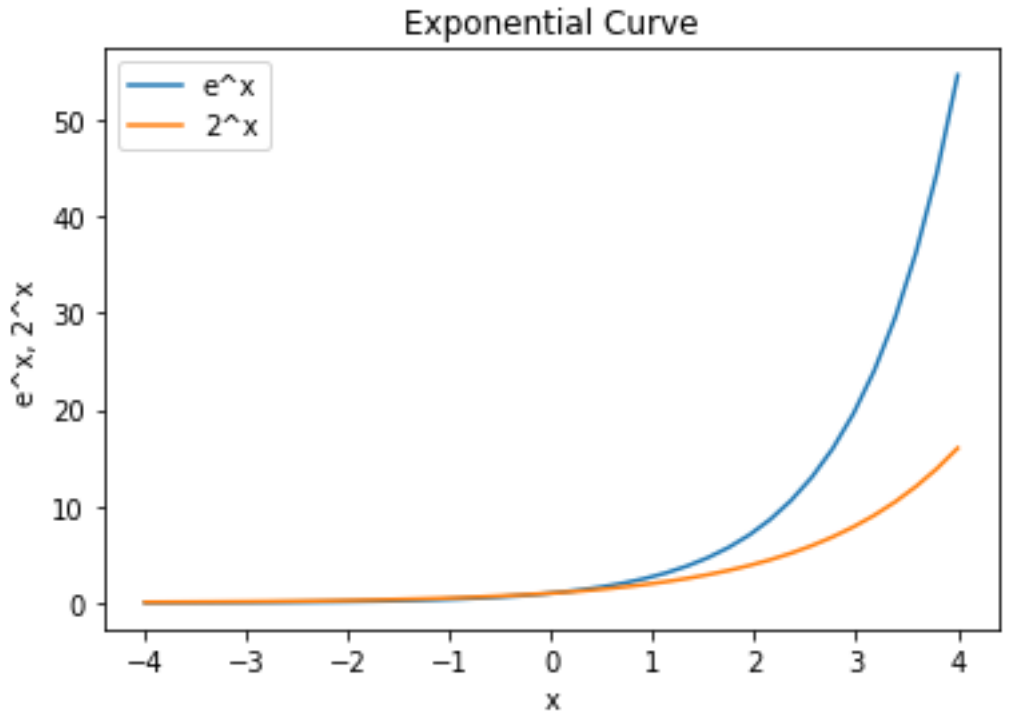
- **Eg:** Consider a simple linear model between **Inflation (X) and Interest rate (Y)**
- US economy monthly data is given for 1999
- We know that Interest rate is normally increased when inflation increases
- Scatter plot is shown of actual values **Y_i (blue dots)**, with the predicted values $\widehat{Y}_i$ on the **best fit regression line (red)** through them
- The model minimises the sum of the vertical distance between all points Y_i (x_i,y_i) and $\widehat{Y}_i$ on the best fit line
- For a simple 2 variable sample, it can be represented as the **equation of a straight line Y = mX + c**
  - m – slope of the line, c – y-intercept (x=0)

X	Y
1.7	4.75
1.6	4.75
1.7	4.75
2.3	4.75
2.1	4.75
2	5
2.1	5
2.3	5.25
2.6	5.25
2.6	5.25
2.6	5.5
2.7	5.5



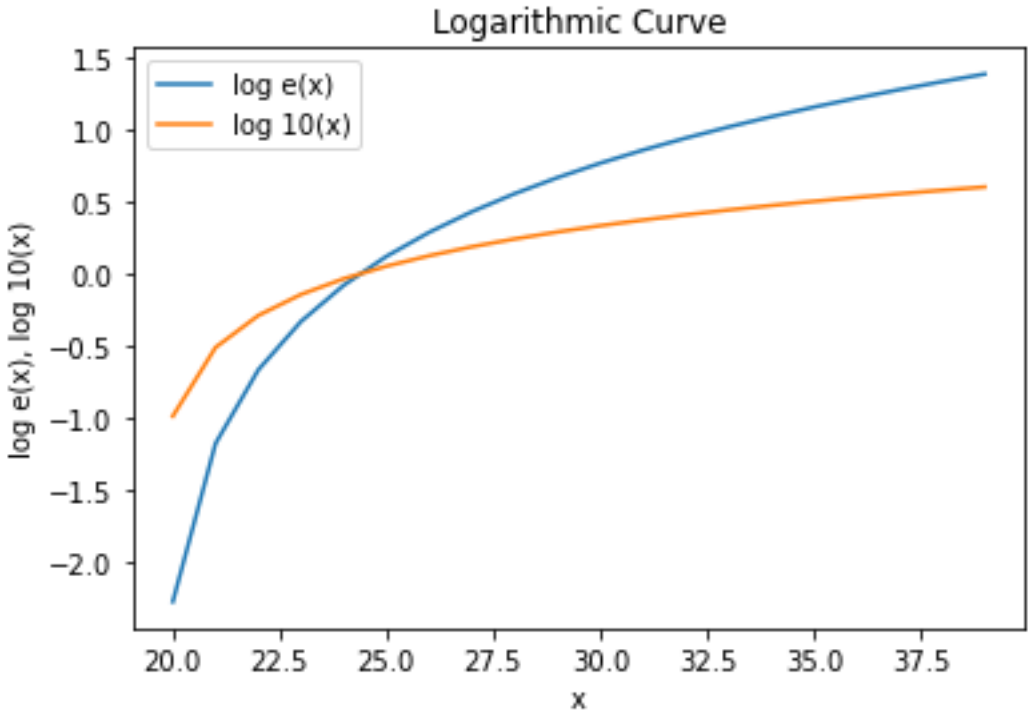
Chapter 3 – Exponentials

- **An Exponential function is of the form $f(x) = a^x$, where a > 0 (a positive real no)**
  - If a>1, it is an increasing (growing) function
  - If 0<a<1, it is a decreasing (decaying) function
  - If a = 1, it is a constant function
- It is useful for calculating quantities that show continuous growth or decay like population/disease, radioactive decay, continuous compound interest, etc
- An important characteristic of exponential functions is that the growth (or decay) rate of the function is directly proportional to its current value
- **Natural exponential function $f(x) = e^x$**
  - Euler's number e = 2.71828…
- Any exponential function can be written as $b^x = e^{x\log_e b}$, where
  - $\frac{d}{dx} e^x = e^x$
  - So derivative of exponential function is equal to itself
- It requires derivatives/differentials (calculus) for better proof and understanding. For now this explanation should be enough for the algorithms to be discussed
- $e^{xy} = e^x * e^y$



Chapter 3 – Logarithms

- **Logarithm is the inverse of an exponential function**
- **$\log_a x = y$**, means that the log of x to the base a = y. a>0, x>0
- The inverse of the log is **$x = a^y$,** which is the exponential function
- **Natural logarithm gives the time needed to reach a certain level of growth**
  - **Natural exponential** gives the amount of continuous growth after a certain amount of time
- Logarithms are used to simplify large calculations
- Log_{10} is called the common logarithm and $\log_e$ is called the natural logarithm
  - Both are easy to calculate using their logarithmic tables and widely used
- Eg: As $100 = 10^2$, then $\log_{10}(100) = 2$. So 10 (base) is multiplied 2 times to get 100
  - However, $\log_e 100 = 4.605$
- They are used to analyse algorithms that solve a problem by dividing it into smaller similar problems. Eg: factorial of a number
- Important logarithmic operations are:
  - Product: $\log_a(xy) = \log_a x + \log_a y$
  - Division: $\log_a(x/y) = \log_a x - \log_a y$
  - Power: $\log_a(x^b) = b \log_a x$
- **Logarithmic scale** is used to express some quantities as logarithms of ratios
  - Eg: decibel – measures the level of sound using logarithm of a power ratio, richter scale – measures earthquake strength using logarithm of the energy emitted



Chapter 4 – Probability

- Students are expected to have some basic knowledge of probability (school level)
- This chapter will cover the main probability concepts required for data science

Useful books, websites, online courses

- https://www.mathsisfun.com/
- https://mathworld.wolfram.com/
- Book - **Probability: Theory and Examples, Rick Durrett**
- Lots of online courses (Udemy, MOOC's, etc) and tutorials on youtube

Chapter 4 – Probability - Basics

- **Sample space** (Set, Ω- omega) – List of all possible outcomes of the experiment (model of a random process)
  - Eg: All possible outcomes of 4 tosses of a coin
  - Each toss can be a Head or Tail, so 2 ways. 4 tosses = 2^4 = 2*2*2*2 = 16 outcomes
- **Probability** indicates the likelihood of an event A (subset of outcomes) occurring in a sample space (total outcomes)
- **P(A) = no of ways event A can happen / total outcomes**
  - Eg: Event A = Getting 2 heads in 4 tosses of a coin
  - No of ways (combinations) of getting 2H in 4 tosses
    - HHTT, HTHT, HTTH, TTHH, THTH, THHT
  - Using combinations equation its calculated as ${}^nC_r = n!/r!(n-r)!$
    - n – no of items to choose from, r – no chosen from n items, without repetition
  - No of ways of getting 2H = ${}^4C_2 = 4!/2!*2! = 6$
  - P(A) = No of ways of getting 2H / Total outcomes = 6/16
- **Axioms** – main rules that apply to all probability models are
  - P(A) >=0. Probability of an event is non-negative
  - 0 <=P(A) <= 1. Probability of any event is always between 0 and 1
  - $P(\Omega) = 1$. Probability of the entire sample space occurring is 1
- Probability of the **complement** (opposite) of an event A, $P(A^c) = 1 – P(A)$
  - Sometimes its easier to calculate the opposite of an event

Chapter 4 – Probability - Basics

- An event can occur on its own (independent) or in relation to another event (dependent)
- 2 events A and B are **Independent** if they occur independently (not affected) of each other
  - **P(A and B) = P(A∩B) = P(A)*P(B)**
  - Eg: Each toss of a coin is independent of each other with a probability of ½
  - Tossing a coin once results in a head, P(H) = ½
  - If next toss results in a head its probability will still be ½, irrespective of previous toss being head or tail
- 2 events A and B are **Mutually Exclusive** if they both cannot occur simultaneously
  - **P(A∩B) = 0**
  - **P(A or B) = P(A∪B) = P(A) + P(B) - P(A∩B) = P(A) + P(B)**
  - Eg: Rolling a 6 sided die, what is the probability of getting a 3 or 5
  - For each roll of a die, only one face (1,2,3,4,5,6) can occur, each with probability = 1/6
  - So P(3 or 5) = P(3) + P(5) = 1/6 + 1/6 = 1/3
- 2 events A and B are **Not Mutually Exclusive** if they can both occur simultaneously
  - **P(A or B) = P(A∪B) = P(A) + P(B) - P(A∩B)**
  - Eg: Drawing 1 card from a deck, what is the probability of getting a heart or 7
  - A deck of 52 cards has 13 hearts and four 7's. There is only one 7 of hearts
  - P(Heart) = 13/52. P(7) = 4/52. P(Heart and 7) = 1/52
  - P(Heart or 7) = 13/52 + 4/52 - 1/52 = 16/52 = 4/13

Chapter 4 – Probability – Conditional, Law of Total

- **Conditional Probability** is used to study dependent events which measures the probability of an event B happening given another event A has already happened
  - It is represented as **P(B|A)** = **Probability of B given A =** $\frac{\mathbf{P(B \cap A)}}{\mathbf{P(A)}}$
  - $P(B \cap A)$ being the probability of event A and B occurring together
  - P(A) is the probability of event A occurring independently
  - Event B's probability is conditional on event A
  - Eg: If 2 cards are drawn from a deck of cards, what is the probability of both being Jacks
  - P(A) = probability of drawing a jack in the first try = 4/52, as there are four Jacks in a deck of 52
  - P(B|A) = Probability of drawing a jack in the second try, when the first try drew a jack
  - As there are now 3 Jacks and 51 cards left in the deck, P(B|A) = 3/51
  - So, $P(B \cap A) = (3/51)*(4/52) = 1/221$
- **Law of Total Probability** states that if the sample space Ω is partitioned into n mutually exclusive events (B_1, B_2, … B_n) that cover the entire Ω, then for any arbitrary event A,
  - **P(A) =** $\sum_{i=1}^{n}$ **P(A|Bi)P(Bi) =** $P(A|B_1)P(B_1) + P(A|B_2)P(B_2) \ldots + P(A|B_n)P(B_n)$, where
  - $P(A|B_i)$ is the conditional probability of A given B_i
  - Effectively we sum up the probability of A that falls in each partition

Chapter 4 – Probability – Bayes Theorem

- **Bayes Theorem** describes the probability of an event occurring depending on the occurrence of prior related events
  - It finds the probability of an event provided probability of other related events is known (conditional probability)
- It uses **Bayesian Inference** which is based on updating the probability of an event (hypothesis) as new relevant data becomes available
  - It computes the posterior (after) probability of a hypothesis using prior (previous) probability and a likelihood function
  - **Prior** probability is the probability of an event before new data is collected
  - **Posterior** probability is the probability of the event after the new data has been incorporated
- It is useful for **dynamic analysis of data** in multiple industries (stock price, disease risk, machine defects, etc) by using prior probabilities updated with new information
  - **For 2 events A and B, Bayes theorem states that** $\mathbf{P(A|B) = \frac{P(B|A) * P(A)}{P(B)}}$, where
  - **P(A|B)** is the **posterior probability,** which is the conditional probability of A given B has occurred
  - **P(B|A)** is the **likelihood,** which is the conditional probability of B given A has occurred
  - P(A) is the **prior probability**, which is the probability of event A occurring
  - P(B) is the **evidence** (**marginal likelihood)**, which is the probability of event B occurring
  - Using law of total probability $P(A|B) = \frac{P(B|A) * P(A)}{\sum_{i=1}^{n} P(B|A_i)P(A_i)}$
- Another way to express Bayes theorem: $\mathbf{Posterior\ probability = \frac{Likelihood * Prior\ probability}{evidence}}$

Chapter 4 – Probability – Bayes Theorem

- Eg: Below table shows a random sample of 10,000 people with or without symptoms who have or don't have cholesterol
- What is the probability of a patient having cholesterol, given they are not showing symptoms

	Cholesterol		
Symptoms	**Yes**	**No**	**Total**
Yes	1,000	3,000	4,000
No	10	5,990	6,000
Total	1,010	8,990	10,000

- **P(Cholesterol | No Symptoms) = $\frac{\textbf{P(No Symptoms|Cholesterol)} * \textbf{P(Cholesterol)}}{\textbf{P(No Symptoms)}}$**
- $= \frac{P(\text{No Symptoms}|\text{Cholesterol}) * P(\text{Cholesterol})}{P(\text{No Symptoms}|\text{Cholesterol}) * P(\text{Cholesterol}) + P(\text{No Symptoms}|\text{No Cholesterol}) * P(\text{No Cholesterol})}$
- P(No Symptoms|Cholesterol) = People with No symptoms out of the total who have Cholesterol = 10/1,010
- P(Cholesterol) = Total people who have Cholesterol in the sample, with and without Symptoms = 1,010/10,000
- P(No Symptoms) = Total people who show No Symptoms in the sample, with and without Cholesterol
  - $= \frac{10}{1{,}010} + \frac{5{,}990}{8{,}990} = \frac{6{,}000}{10{,}000}$
- So, **P(Cholesterol | No Symptoms) = $\frac{(10/1{,}010) * (1{,}010/10{,}000)}{(6{,}000/10{,}000)} = \frac{10}{6{,}000} = 0.167\%$**
- It can be confirmed from the table too: Total people with No Symptoms is 6,000 and out of that only 10 have Cholesterol
- This theorem can be applied to any problem as long as the probabilities on the RHS of the equation can be calculated

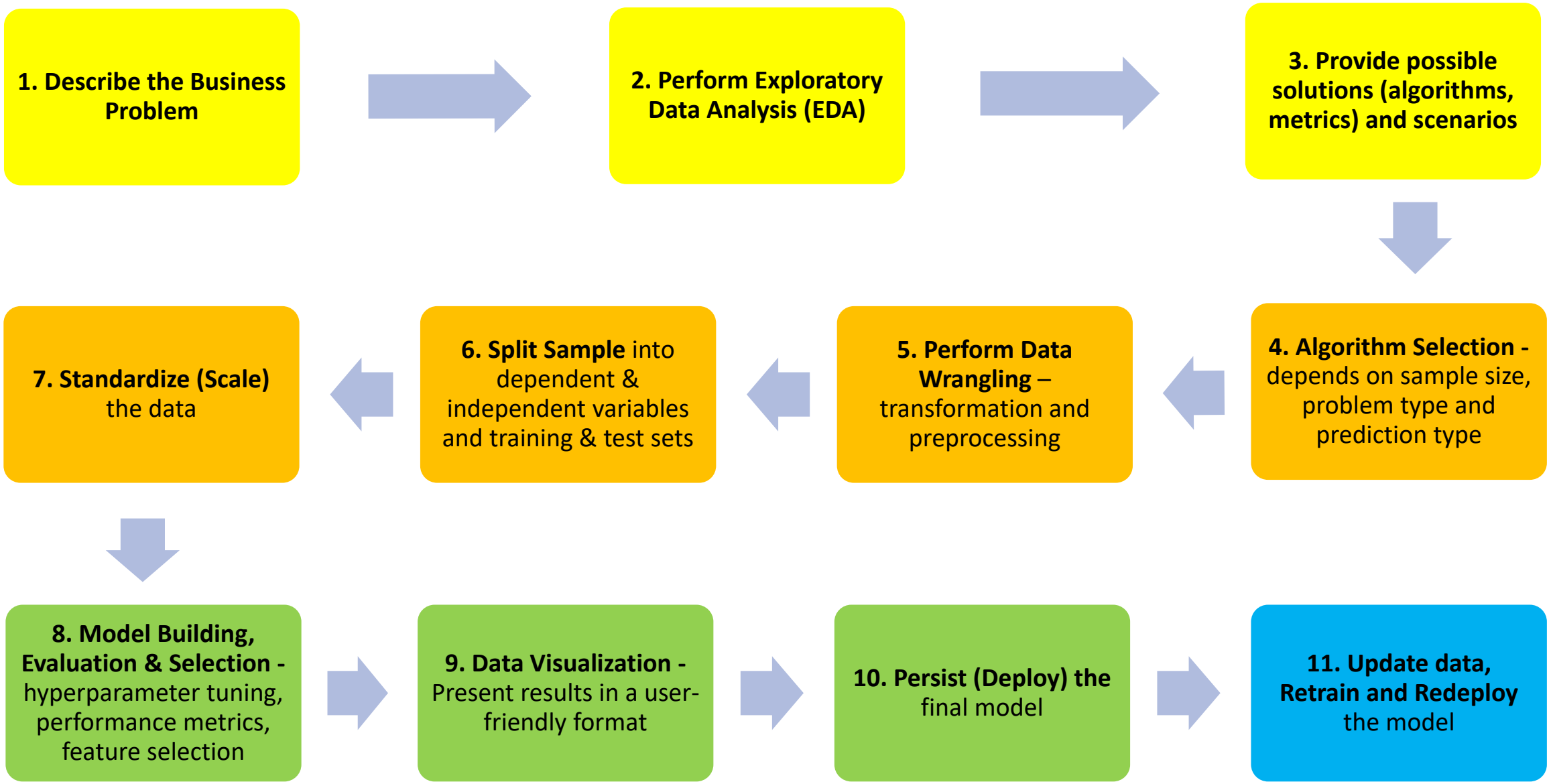
Chapter 4 – Probability – CLT, PMF

- **Central Limit Theorem** (CLT) states that the sum or average of many independent random variables is approximately a normal random variable
  - Say we have *n* X_1, X_2, …, Xn independent random variables (samples) each of size *n* with the same underlying distribution (same population mean **μ** and standard deviation $\boldsymbol{\sigma}$)
  - For each sample n, mean $\bar{X}_n = \frac{X_1 + X_2 + \ldots + X_n}{n}$, $\bar{X}_n$ itself being a random variable
  - **Law of Large Numbers**: As the size of *n* (each sample) increases, the probability that $\bar{X}_n$ is close to μ goes to 1
  - **CLT**: As the number of random variables (samples) *n* increases, the distribution of $\bar{X}_n$ converges to a normal distribution N(μ, σ)
  - CLT approximates the sum or average of independent random variables by a normal random variable, which is useful as its easy to do computations with a normal distribution
  - So for a sufficiently large sample its mean approximates the population mean
  - Eg: Random samples of coin tosses, with increasing sample sizes. Python code given
- **Probability Mass Function (PMF)** – shows the probability distribution of the sample by mapping frequencies of the values to their probabilities by dividing them by the sample size n (normalization)
  - It is similar to a histogram, plotting probabilities instead of frequencies
  - Eg: Student frequency for marks in maths. Convert frequency to probability and plot. Python code given
- ***Probability Density Function (PDF)*** *– will be covered after the calculus chapter*

Chapter 5 – Data Sampling

- ***Note -*** We will not go into much detail here as we'll use the in-built sampling process in the machine learning libraries
  - **k-fold cross-validation** is an efficient and low-error method of random sampling of multiple samples in the data
- **Population** is the set of possible values (actual or hypothetical) of the data being considered for analysis and is often too large to measure entirely
- **Data Sampling** is the selection of a subset of data from the entire population to estimate its characteristics
- **Data Sampling** is a critical function in statistics as it impacts the quality of data used for analysis and is cheaper and less time consuming than measuring the entire population
- **Sample Size** is the number of sample observations selected from the population
- Sampling can be done by two techniques – probability and non-probability
- **Probability Sampling** – every element in the population has a chance of being selected at random in the sample (probability > 0)
- **Non-Probability Sampling** – some elements in the population have no chance of being selected in the sample
  - Selection is based on assumptions regarding the population, making it non-random
  - It results in sampling bias and sampling error
- **Sampling Bias** occurs when some elements in the population have a higher or lower probability of selection (also called as selection bias). Different types are - Self-selection, Pre-screening, exclusion, etc
- **Sampling Error** is caused by the difference between the estimated statistic for the sample and the actual parameter for the population. It can be reduced by using a large no of random samples, preferably of larger size
- **Non-Sampling Error** is the deviation of the sample estimate from the population parameter irrespective of the sample chosen. They can be random or systematic and are much harder to quantify

Chapter 6 – Data Science Process



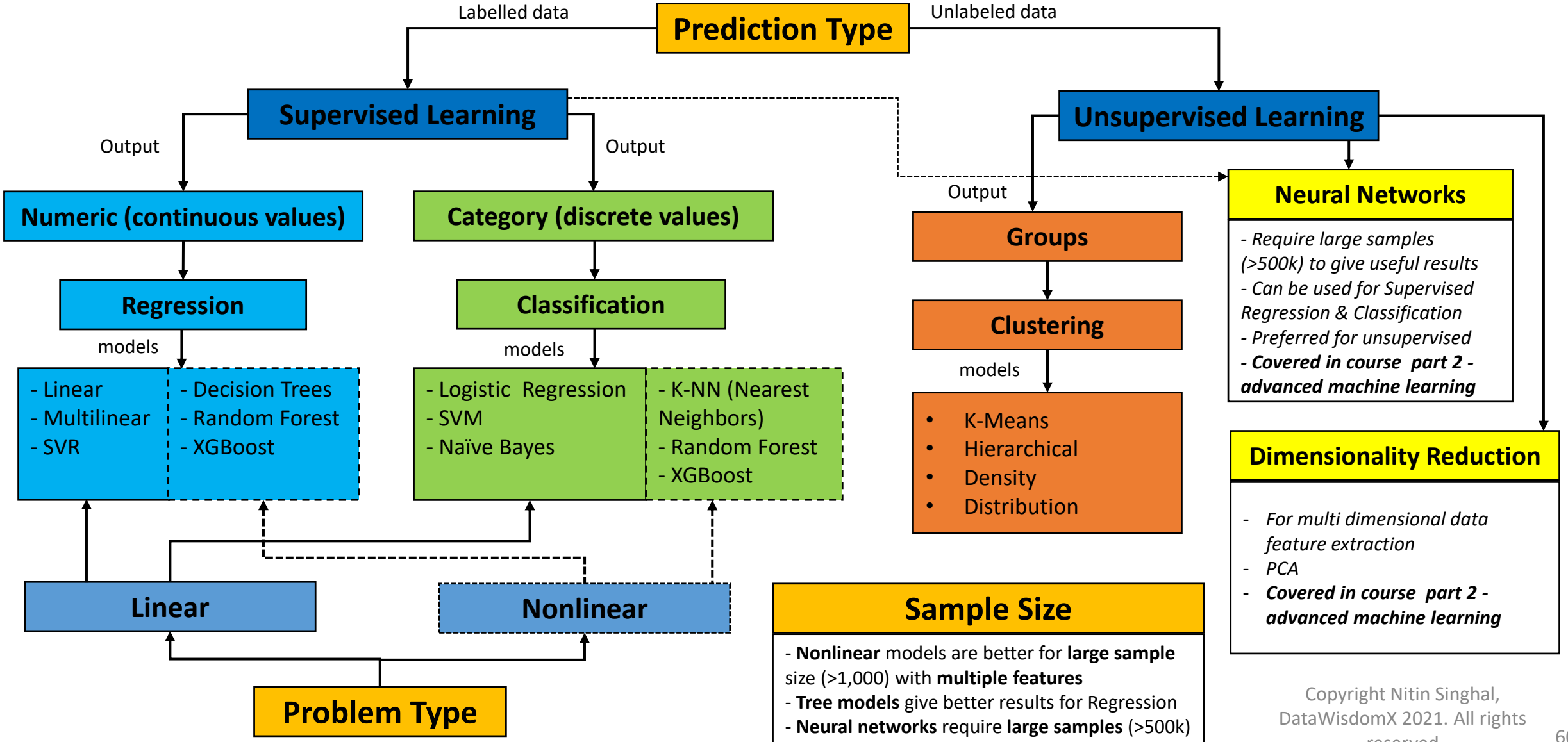
Chapter 6 – Data Science Process

- **Most data science problems can be solved by following a standard process given below**
- **Scikit-learn** provides a very good guide for the entire process - https://scikit-learn.org/stable/data_transforms.html

1. **Describe the business problem** to be solved
    1. Understand the business problem that the users are facing which can be solved using data science
    2. Assess whether they have the necessary data to build the machine learning model
    3. Evaluate the current business process for data sources of the sample data. Identify data quality issues
    4. Identify the features (variables) to be predicted (dependent, Y) using the other features in the sample data (independent, $X_1 \ldots X_n$)
    5. The objective is to build an algorithmic model to predict Y using $X_1 \ldots X_n$, where $Y = f(X_i)$, where i = 1 to n
    6. Define the data science process steps that will be performed to generate the expected outcomes
2. **Perform Exploratory Data Analysis (EDA)** to understand the data - Use python or excel and visualization
    1. Sources - manual, documents, databases, emails, etc
    2. Types/Formats – csv, pdf, txt, json, database, etc
    3. Gaps - input accuracy, update frequency, time gaps, missing data, erroneous data, etc
    4. Identify data wrangling steps (to be performed in detail later)
3. **Provide possible solutions and scenarios** for results and how to interpret them
    1. What are the possible outputs that can be generated from the machine learning model
    2. What type of algorithms can be used – supervised/unsupervised (regression, classification, clustering), deep learning, natural language processing, etc
    3. State what the metrics and results would mean numerically and in business terms

Chapter 6 – Algorithm Selection

4. Algorithm (model) Selection will depend on prediction type, problem type and sample size



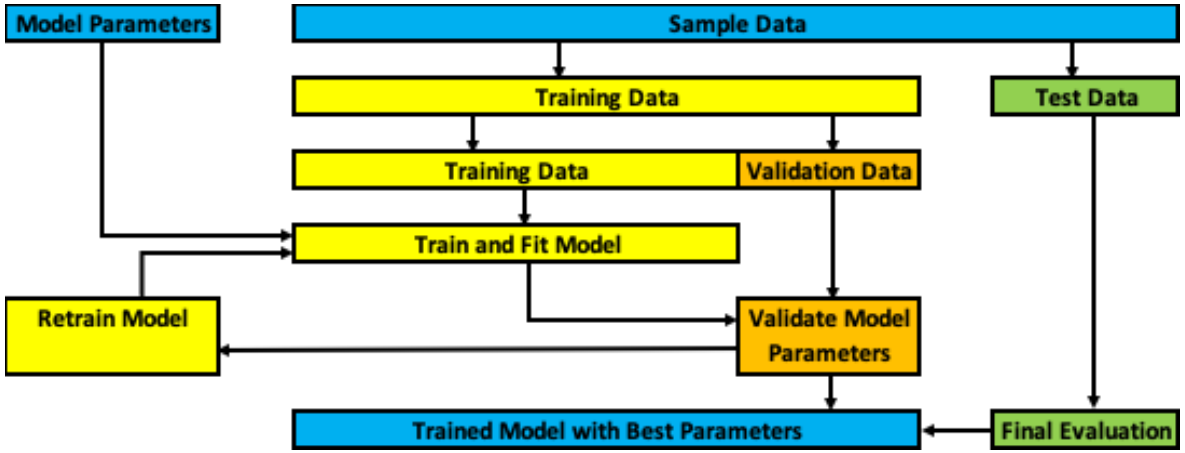
Chapter 6 – Data Science Process

5. Perform Data Wrangling (transformation, preprocessing)

1. Set of **data transformations** performed to convert all sample data into a standard format that can be used by different algorithms
2. Missing values, errors, formatting, aggregation, encoding categorical data, binning, etc
3. *Covered in Chapter 7 – Data Wrangling*

6. Split sample data into

1. **Dependent** (feature Y to be predicted) and **Independent** (features $X_1 \ldots X_n$ that are used for prediction) variables
    1. Depending on the data and the algorithm used there can be one or more independent variables
2. **Training, Validation and Test sets**
    1. **Training data** – large subset of the sample data used to train and fit the model
    2. **Validation data** – *small subset of the training data* used for evaluating the parameters of the trained model
        1. Multiple iterations of training and validating are performed before final evaluation using the test data
    3. **Test data** – small subset of the sample data used for final evaluation of the model parameters using error metrics
    4. **All 3 splits should be independent and not overlap with each other**
    5. Splitting into 3 sets reduces the amount of data available for training
        1. This can be overcome by using k-fold cross-validation process



Reference: *Scikit-learn user guide*

Chapter 6 – Data Science Process

6.3 Train-test split can be done using sklearn's model_selection train_test_split library (if validation set is not required)

- Try different split ratios to get the best result – 0.2, 0.25, 0.33 etc

6.4 Random splitting is done to reduce overfitting/underfitting during model training

6.5 Overfitting/Underfitting – occurs when model is fit too closely to test data or sample size is small. Gives bad predictions with new data

7. Standardizing (**Scaling**) the data is required for most algorithms as they are distance based

1. Data must be centred around mean 0, variance 1 (normalization) or mean with variance of the same order
2. Data with higher order variance (outliers) might get a higher weight in the model, causing bad predictions on new data
3. **Only training data needs to be standardised and not test data**
    1. Otherwise test data will use its own mean/standard deviation leading to a badly fit model
4. Standardizing can be done using sklearn's preprocessing Scale or StandardScaler (which scales and transforms) library
    1. The transformer created for training data can then be applied to test data
5. It is not required for models like (Random Forest, XGBoost) as they are not distance based (not sensitive to feature size)
6. Now that the **data is in a standard format** it can be used by different algorithms (models)

Chapter 6 – Model Building, Evaluation & Selection

8. Model Building, Evaluation & Selection involves

1. **Sample splitting** - Training, testing and validating on different subsets of the sample data
2. **Model evaluation** and **Hyperparameter tuning**
3. Using **error metrics** for measuring model accuracy
4. **Feature (independent) selection iteratively** to remove low importance features, thereby improving model performance without loss of information
5. **Result of the above process will select the best trained model with optimized hyperparameters**
6. The above process is followed during the building of all models and is shown in all python code examples
7. Useful website - https://scikit-learn.org/stable/model_selection.html

- This can become laborious, computationally expensive and prone to overfitting/underfitting
- It can be simplified into an efficient process by using **Pipeline** & **GridSearchCV** libraries together
- **Pipeline** combines data transformations and training multiple models into a sequence of steps
  - **Grid search** across entire hyperparameter grid to select the best combination for each model
  - **Hyperparameters** to be tuned for each model are setup as a predefined parameter array (grid)
  - Running multiple loops of the above using different independent subsets (**K-fold cross-validation**)
  - *Benefit* - It calls fit and predict only once on the sample data for an entire sequence of models
  - *Benefit* - It avoids leaking statistics from test data into the trained model during cross-validation

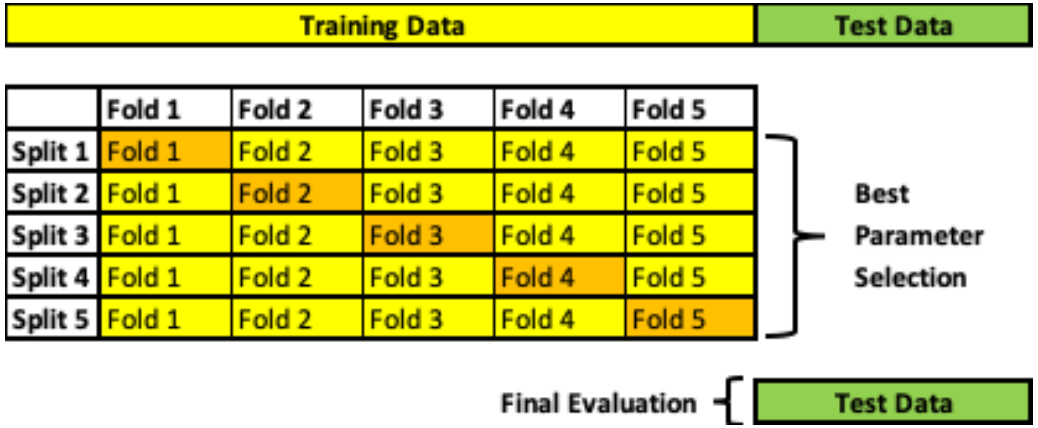
Chapter 6 – Model Building, Evaluation & Selection

- **GridSearchCV** performs **K-fold cross-validation** to train models on multiple variations of the training data of the sample in an iterative loop (k-folds, k-splits) :

  1. Each split repeats steps 2 and 3 below in a loop by changing the fold used for validation, resulting in a new split
  2. Model is trained using k-1 folds as training data
  3. Trained model is validated on the remaining fold of data (using a performance measure like accuracy)
  4. The performance measure calculated by k-fold cross-validation is then averaged across all the iterations

  - **Final evaluation** on the test data (split from the sample and not included in the k-folds)

- **Performance measures used**
  - Error metrics for **regression** (details in Chapter 11)
    - **Minimize** MSE, RMSE, MAE etc as they indicate difference between predicted and actual values
    - **Maximize** R^2 as it indicates goodness of fit of model
  - Evaluation metrics for **classification** (details in chapter 12)
    - **Confusion matrix, etc** to evaluate accuracy of a classifier
    - ROC curve for binary classifiers



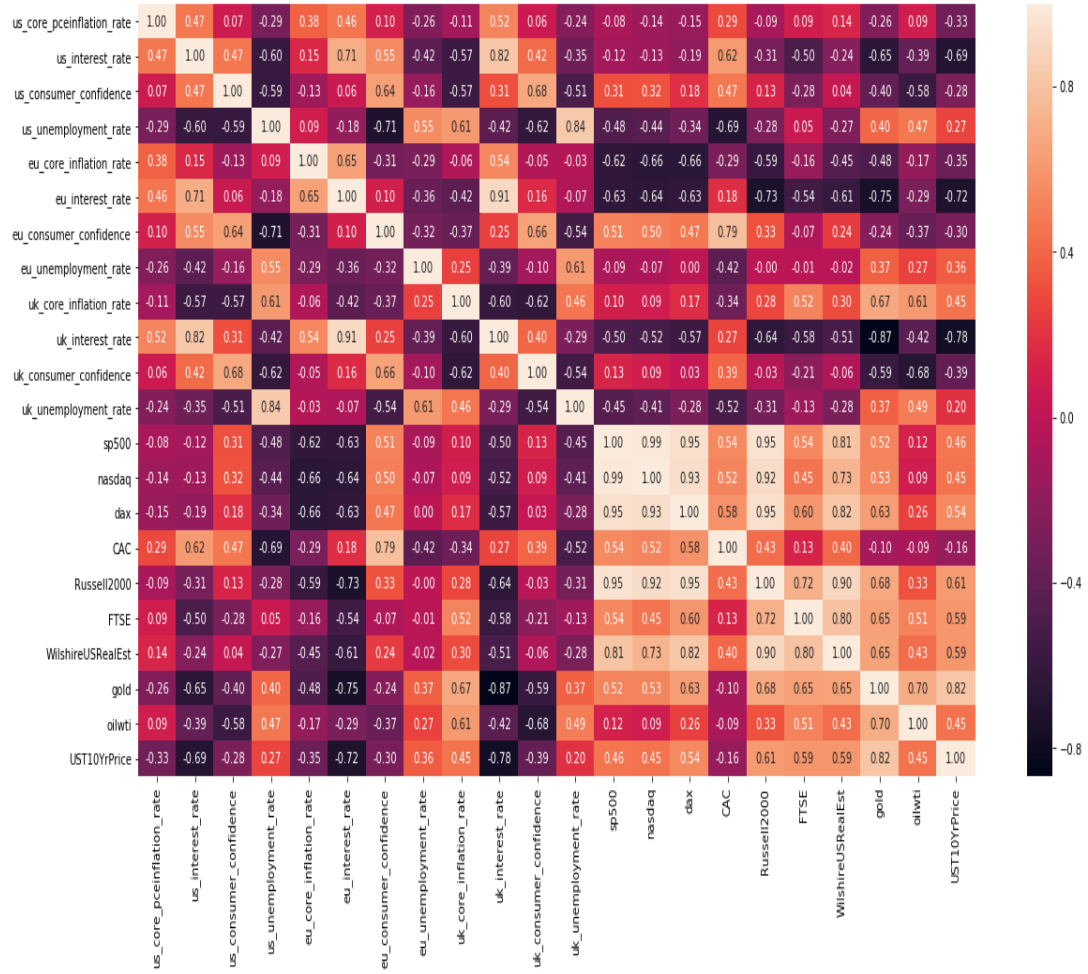
Reference: *Scikit-learn user guide*

Chapter 6 – Data Visualization

9. Data Visualization involves presenting the results in a user-friendly format using different libraries

- **Pandas plot** can be used for basic data visualization
  - It provides a basic plot function for different chart types called directly on dataframes/series
  - Its a wrapper around matplotlib plot function
  - https://pandas.pydata.org/pandas-docs/stable/user_guide/visualization.html
- **Matplotlib** is the most popular and comprehensive python library for data visualization, used for a variety of chart types
  - It integrates with python, pandas and numpy making it easy to plot a variety of data
  - It provides high flexibility in plotting chart types, axes, colours, markers, 3D plots, etc
  - https://matplotlib.org/tutorials/index.htmls
- **Seaborn** is a python library based on matplotlib
  - It provides ease and flexibility for advanced and attractive data visualization
  - Eg: heatmaps, multiplot grids, aesthetic figures, statistics plots (model type), etc
  - https://seaborn.pydata.org/tutorial.html
- All 3 libraries are used in the examples given for the different algorithms to show their functionality
- Use **Excel** for tabular presentation and other data manipulations

Heatmap for Correlation between Core macroeconomic factors and Asset prices



Chapter 6 – Data Science Process

10. Persist (Deploy) the final model to a location where it is integrated into the live environment of the overall analytics solution

1. Generally a repository (directory, endpoint) in a cloud hosted storage (AWS S3 bucket) or a local machine
2. Make API calls to the model to return the results for new data
3. Will be discussed in Chapter 11 – Model Deployment

11. Update data, Retrain and Redeploy the model

1. Once the model is deployed in the live environment, it must be retrained (updated) regularly depending on new data sources or more data becoming available
2. If new data sources are considered, start from step 1 of the data science process
3. Retrained model, if better than previous version, should be redeployed to live
4. This process requires full version control of each live model and the data it used
    1. Necessary for comparing different versions or rolling-back to the best version in case of issues

Sklearn terms, Python code

1. Some terms specific to sklearn are give below (common for all models)
    1. **An *Estimator* is the implementation of the model in a sklearn class**
    2. Estimator class instance is used for calling the required methods
    3. Estimator's **fit()** method fits the estimator to the training data (X and Y) to learn from it and build the model
    4. Estimator's **predict()** method predicts Y values using the X test data
    5. The predicted Y values are then compared to Y test values to measure the accuracy of the estimator (model)
    6. The model is now used for predicting new values of Y using new X values
2. ***Note*** *- By now it is expected that students are comfortable with python coding and related terminology*

Chapter 7 – Data Wrangling

- **Data Wrangling** performs a set of **data transformations** to convert all sample data into a standard format to be used by different algorithms
- **Model results are only as good as the input data** - This is a critical activity taking up to 20%-30% of the total data science process
- Most common data transformations are given below

1. **Cleaning** – uses results of EDA to transform missing/null/erroneous data
    1. Simple strategy would be to remove entire rows/columns or set them to 0, but it can lead to loss of information
    2. Better approach would be to impute missing values using existing data
    3. **Sklearn impute library** – missing values can be imputed using the same feature or all available features
        1. Univariate – strategy using the same feature to impute its missing values by selecting a constant value or statistic like mean, mode, etc
        2. Multivariate – strategy using a function of other features to impute missing values for a feature
            1. Iterative imputer using a regression function, K-Nearest neighbours approach, etc
    4. **Pandas functions** - fill missing data with a specific value (0, mean, etc) using fillna()
        1. Forwards, backwards - ffill(), backfill()
        2. Interpolate missing values using different methods like linear, quadratic, etc - interpolate()
    5. **Drop duplicate rows** as they are erroneous data and should not be used
    6. **Remove irrelevant data** that has no impact on model performance - Constant value for a feature in all rows

Chapter 7 – Data Wrangling

2. Formatting fields – data from various sources (file, database, etc) using different datatypes for the same feature

1. Datetime – has the most widely varying formats across different sources and converting features to a standard format is necessary
2. Numeric as strings – some data sources might store numeric values as strings, which will throw an error during data processing
3. Incorrect datatype – Data might be stored as string or object which is not useful for numeric or datetime related operations

3. Extracting useful information – Splitting a feature into separate features

1. One column into multiple columns or vice-versa (Joining)
2. Eg: Splitting datetime into year, month, day for time series operations or vice versa

4. Encoding Categorical data - text data must be converted to numeric as the algorithms work only with numbers

1. Transforming categorical features into integers is not advisable as the model might interpret them as being ordered and give them inaccurate weightings
2. **One Hot Encoding** is better as it converts them into a binary matrix of **'n'** categories **(dummy variable)**
    1. By assigning 1 to one feature and 0 to the rest
    2. This process is repeated for the remaining categorical features
    3. It represents each categorical feature as a column (dummy variable)
3. **Dummy variable trap -** As independent variables can be highly correlated , remove one dummy variable to overcome it
    1. Eg: Categorical data = [Male, Female]
    2. Requires only 1 dummy variable as male=1 implies female=0

Chapter 7 – Data Wrangling

4.4 sklearn.preprocessing, **OneHotEncoder** library can be used for encoding categorical data (features)

1. Features are encoded using a one-hot (aka 'one-of-K' or 'dummy') encoding scheme
2. Parameter '**drop'** specifies how to drop one of the categories per feature

5. Binning (discretization, quantization, bucketing) partitions continuous data into discrete values (categories, bins)

1. Continuous data can be large, skewed and difficult to analyze
2. Partitioning it into discrete categories makes it easier to understand
3. Sklearn's KBinsDiscretizer can be used for different binning strategies
4. Uniform – uses constant width bins (fixed range of values)
5. Quantile – uses quantiles to have equally populated bins (same % of values, quartiles, etc)

6. Aggregation – For large feature sets its sometimes better to aggregate (mean, std dev, etc) the data

1. Done at dependent variable level to make the model and feature selection more efficient
2. Without losing too much relevant information
3. Reduce random noise

7. Merging/Joining of different datasets of the sample is required to create a single dataset of independent variables which can then be used by different algorithms

8. Pandas is very good for data wrangling

9. Numpy is very good for maths, statistics and array operations

10. Once data wrangling is complete the output is used by the remaining data science process steps

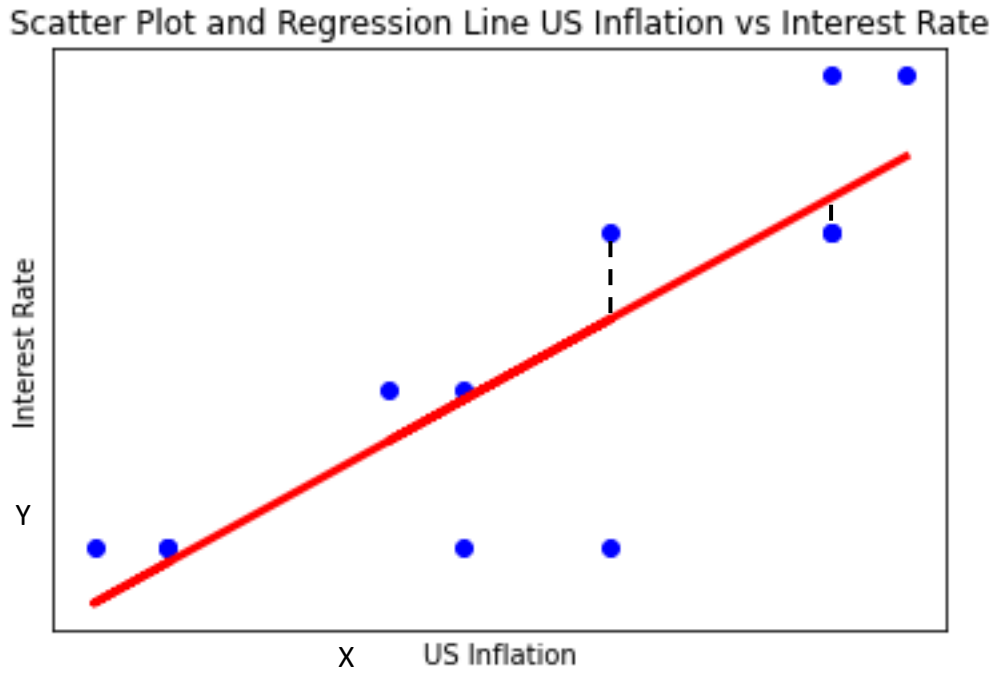
Chapter 8 - Supervised Learning Algorithms - Regression

- **Regression algorithms are used for predicting values on a continuous scale (real numbers, not just 0 or 1)**
- Dependent variable Y values (target function) are predicted using the independent variables $X_{1..n}$ (features)
- Common types are Linear, Multiple linear, Random Forest, XGBoost, *Neural Networks*, etc
- *Regression concepts are already covered in the chapter 3 statistics. Given below briefly:*
  - **Regression** is a statistical process for estimating the relationship between a dependent variable Y and one or more independent variables $X_{1..n}$ in a sample
    - $\mathbf{Y = \beta_0 + \beta_1 X_1 + \ldots + \beta_n X_n + \varepsilon}$.
    - β - unknown parameters (coefficient) for each X. ε - deviation error of sample values from actuals
  - The goal is to estimate a function $\widehat{Y}$ that closely fits the data.
    - $\widehat{Y} = \widehat{\beta_0} + \widehat{\beta_1} X_1 + \ldots + \widehat{\beta_n} X_n + \varepsilon$.
    - $\hat{\beta}$ is the estimate of each parameter
- Models like **Ordinary Least Squares OLS** estimate $\hat{\beta}$ by **minimizing the loss function Residual Sum of Squares (RSS)**
  - **OLS = Min (RSS). RSS** = $\sum_{i=1}^{n} (Y - \hat{Y})^2$
- Effectively, the model tries to minimise the difference between the predicted and actual values (best fit line)
  - The model uses error metrics like MSE, RMSE, MAE, R^2 for minimizing the loss function
- Geometrically it is the sum of the squared perpendicular distance of each sample point from the best fit line passing through them
  - Closer the points are to the line, better the fit
- Useful website - https://scikit-learn.org/stable/modules/linear_model.html

Chapter 8 - Supervised Learning Algorithms - Regression

- **Eg:** Consider a simple linear model between **Inflation (X) and Interest rate (Y)**
- US economy monthly data is given for 1999
- We know that Interest rate is normally increased when inflation increases
- Scatter plot is shown of actual values **Y_i (blue dots)**, with the predicted values $\widehat{Y}_i$ on the **best fit regression line (red)** through them
- The model minimises the sum of the vertical distance between all points Y_i (x_i,y_i) and $\widehat{Y}_i$ on the best fit line
- For a simple 2 variable sample, it can be represented as the **equation of a straight line Y = mX + c**
  - m – slope of the line, c – y-intercept (x=0)

X	Y
1.7	4.75
1.6	4.75
1.7	4.75
2.3	4.75
2.1	4.75
2	5
2.1	5
2.3	5.25
2.6	5.25
2.6	5.25
2.6	5.5
2.7	5.5

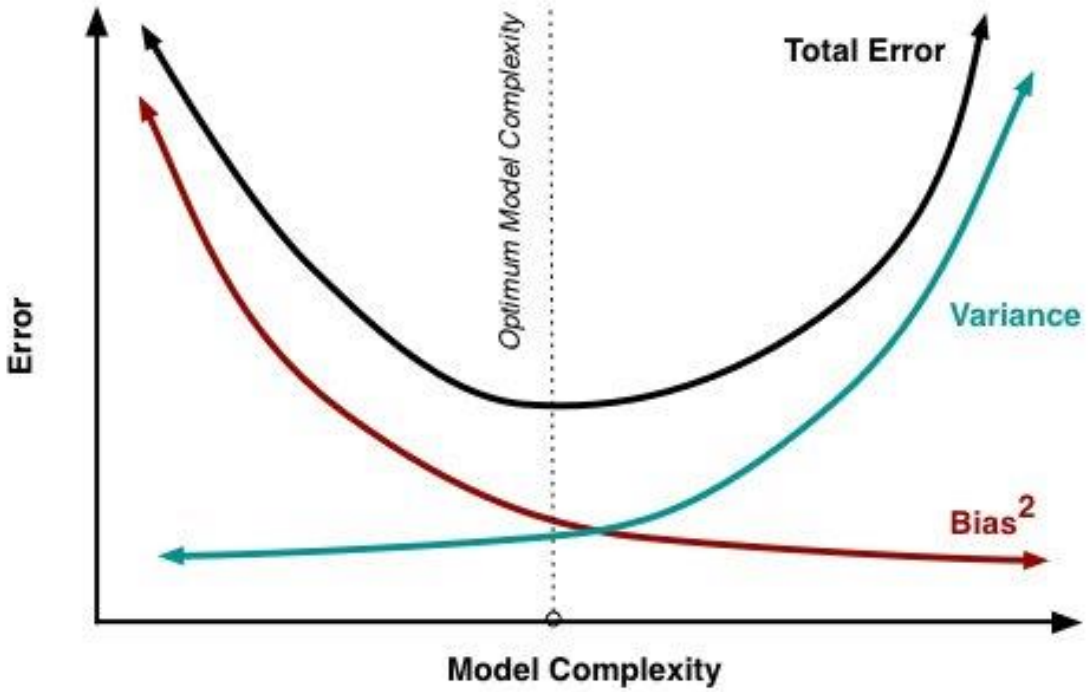


Chapter 8 - Supervised Learning Algorithms - Regression

- As all machine learning models work with numbers and since the data can be numeric (integer or real), ordinal ('true', 'false') and categorical (text like city, group type, etc), it needs to be transformed using the Chapter 7 - Data wrangling steps (encoding, binning, etc)
- Not all regression models are distance based so they might not require standardization (scaling), already covered in Chapter 6 - Data science process
  - Data has to be centred around mean 0, variance 1 (normalization) or mean with variance of the same order
  - Data with higher order variance (outliers) might get a higher weight in the model, causing bad predictions on new data
  - Only training data needs to be standardised and not test data
- Standardization is not required for regression models like (Random Forest, XGBoost) as they are not distance based (sensitivity to feature size)
  - Being decision tree based they consider all features independently based on value
- Supervised learning algorithms suffer from overfitting or underfitting depending on data size and model complexity
  - It is resolved by using the Bias-Variance trade-off
- ***Data source and python code for regression problems***
  - *https://github.com/datawisdomx/DataScienceCourse/tree/main/Part1-MachineLearningandDataAnalytics/*
  - *Data files*
    - *USInflIR10YrYieldPrice.csv*
    - *USMacro10yrPriceYield.csv*

Chapter 8 – Supervised Learning - Bias Variance Tradeoff

- **Overfitting or Underfitting** results in a model that gives good results with training data but bad results with test or new data
  - This is caused by model complexity and training data size
- **Bias (underfitting)** is the error when the estimator makes assumptions about the model from the training data rather than learning from it (underfits to the data)
  - It occurs when the model has **high bias and low variance,** (less data)
  - Low/High bias estimators make less/more assumptions about the model
- **Variance (overfitting)** is the error due to the estimator learning too much from the data it (overfits to the data)
  - It is sensitive to small changes (noise) in the training data
  - It occurs when the model has **high variance and low bias,** (too much data)
  - Low/High variance estimators are less/more sensitive to training data changes
- The **objective** of the data science process is to **build a model that has low bias and low variance**, but that requires a tradeoff
- **Bias – Variance tradeoff** is required due to the inverse relationship between bias and variance
  - Linear models (simple) have high bias but low variance – linear regression, logistic regression, etc
  - Nonlinear models (complex) have high variance but low bias – random forest, k-nearest neighbours, etc
- **The tradeoff can be optimized** by using different models and reducing error by performing k-fold using cross-validation on training data



Source: Scott Fortmann-Roe, Understanding the Bias-Variance Tradeoff

Chapter 8 – Supervised Learning – Regression - Error Metrics

- To choose the best estimator (model) we use error metrics like MSE, RMSE, MAE, R^2
- **MSE** – Mean Squared Error is the average of the sum of squares of the errors (predicted-actual values)
  - MSE = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. y_i – actual value. $\hat{y}_i$ - predicted value
  - We want to minimize MSE. Closer it is to 0, better the estimate
  - It penalizes even the smallest error due to the squaring
- **RMSE** – Root Mean Squared Error is the square root of MSE. It is the preferred metric as it has the same unit as the data
  - RMSE = $\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$
  - We want to minimize RMSE. Closer it is to 0, better the estimate
  - It penalizes large errors
- **MAE** – Mean Absolute Error is the average of the absolute difference between predicted and actual values
  - MAE = $\frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$
  - It is robust to outliers as it weighs all differences equally. Not useful if you want to identify outliers
- **R^2** (Coefficient of Determination or Goodness of Fit) - It explains how well the model fits the sample
  - It is the proportion of variance in the dependent variable that is predictable from the independent variables
  - **R^2 = 1-(RSS/TSS)**
  - Sum of Squares of Residuals, RSS = $\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$. Variance of the model's predictions. Regression minimizes RSS
  - Total Sum of squares, TSS = $\sum_{i=1}^{n}(y_i - \bar{y})^2$. $\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i$. Variance of the dependent variable
  - Closer it is to 1, better the estimate
- Useful website - https://scikit-learn.org/stable/modules/model_evaluation.html#regression-metrics

Chapter 8 – Linear Regression

- The linear regression equation for a single independent variable model is:
  - $\mathbf{Y_i = \beta_0 + \beta_1 X_i + \varepsilon_i}$
- Ordinary Least Squares (OLS) method finds $\hat{\beta}_0$, $\widehat{\beta_1}$ of β_0 , β_1 that minimizes the sum of the squared errors ε **(RSS)**
  - $\mathbf{\varepsilon_i^2 = RSS = \sum_{i=1}^{n} (Y_i - \beta_1 X_i - \beta_0)^2}$
- Resulting in the equations below
  - $\hat{\beta}_0 = \bar{y} - \widehat{\beta_1}\,\bar{x}$
    - $\bar{x}$, $\bar{y}$ are the sample means of x, y
  - $\widehat{\beta_1} = \frac{Cov(x,y)}{Var(x)} = \frac{\sum_{i=1}^{n}[(xi-\bar{x})*(y_i-\bar{y})]}{[\sum_{i=1}^{n}(xi-\bar{x})^2]/n}$
    - *Proof requires calculus, not covered here*
- So, the equation of the best fit regression line will be $\mathbf{\hat{Y}_i = \hat{\beta}_0 + \widehat{\beta_1} X_i}$
  - It is then used to calculate the predicted value $\hat{Y}_i$ for each X_i , to plot the best fit regression line
- **Sklearn.Linear_model, LinearRegression** library is used for building a linear regression model
- Using Bond Prices (Y) and Inflation rate (X) for USA we will determine if there is a linear relationship between them
  - Bond prices are calculated using yield or interest rates, which are dependent on inflation rates
    - There is an inverse relationship between price and yield
    - Price calculation formula is available online and in books
    - It uses face value, time to maturity, periods, coupon rate, yield to maturity (discount rate or interest rate)
  - Data for 20 years, 1999-2019, is used
- The results show that there is a linear relationship between of Bond Prices (Y) and Inflation rate (X) and the model can predict values with reasonable accuracy (RMSE 8.4 and -8.3% validation difference between actual and predicted)
  - It is important to note that problems with strong linear relationship will give better results
- ***Note*** - In the real world most problems involve multiple independent variables and the relationships between them are not strongly linear as will be shown in further models and examples

Chapter 8 – Multiple Linear Regression

- **Multiple Linear regression** uses a linear equation of multiple independent variables ($X_{1...m}$) to estimate the relationship with the dependent variable Y
  - The relationship between Y and each $X_{1...m}$ is linear
- So, Y is now a linear equation of multiple independent X variables ($X_1, X_{2,...,} X_m$)
- For a multiple linear function of X, equation for Y becomes
  - $\mathbf{Y_i = \beta_0 + \beta_1 X_{11} + ... + \beta_m X_{nm} + \varepsilon}$
  - **i** = 1 … n is the no of rows of sample data, **j** = 1 … m is the no of independent X variables
- Ordinary Least Squares method finds $\hat{\beta}_0, \hat{\beta}_1, \ldots, \widehat{\beta_m}$ of $\beta_0, \beta_1, \ldots, \beta_m$ that minimizes the sum of the squared errors ε **(RSS)**
  - $\mathbf{\varepsilon_i^2 = RSS = \sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{m} \beta_j X_{ij})^2}$
- Equation of the best fit regression line will be $\mathbf{\hat{Y}_i = \hat{\beta}_0 + \sum_{j=1}^{m} \hat{\beta}_j X_{ij}}$
- **Sklearn.Linear_model, LinearRegression** library is used for building a multiple linear regression model
- The sample of Bond Prices (Y) and Inflation rate (X), used in the linear model is further enriched by adding other data
  - US macroeconomic factors that impact bond prices like consumer confidence, retail sales, industrial production and unemployment rate
  - This is true for most real-world problems, where the output variable's value depends on multiple input variables
- The multiple linear regression results further improve on the results of the single independent variable linear model for predicting Bond Prices (Y) , indicating that adding more independent variables data improves the model
  - This occurs as with more variables and data the estimator can learn more about the model resulting in better predictions
- However, the results are still not accurate enough (RMSE 5.5 and -3% validation difference between actual and predicted value)
  - We will look at other regression techniques to further improve the results
- ***Note*** - It is important to note that just getting better metrics does not mean better predictions, as error terms like MSE and RMSE can degrade due to a few outlier values
  - This issue will be discussed later in how to select the best model for the given data

Chapter 8 – Polynomial Regression

- **Polynomial regression uses linear regression to fit polynomials to the data**
- It uses linear algebraic equations of the **n^{th} degree** to estimate the unknown parameters $\hat{\beta}_i$
- The relationship between Y and X is non-linear, Y changes with higher order changes in X
- However, the regression model used to estimate $\hat{\beta}_i$ is still linear
- For polynomial function of X, equation for Y becomes
  - $\mathbf{Y_i = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \ldots + \beta_n X_i^n + \varepsilon_i}$
- Ordinary Least Squares method finds $\hat{\beta}_0, \hat{\beta}_1, \ldots, \hat{\beta}_n$ of $\beta_0, \beta_1, \ldots, \beta_n$ that minimizes the sum of the squared errors **ε (RSS)**
  - $\mathbf{\varepsilon_i^2 = RSS = \sum_{i=1}^{m} (Y_i - (\beta_0 + \beta_1 X_{i+} \ldots + \beta_n X_i^n))^2}$
  - **i =** 1 … m is the no of rows of data, **n -** is the degree of the polynomial equation
- Equations for $\hat{\beta}_i$ will incorporate higher degrees of X_i
- Equation of the best fit regression line will be $\mathbf{\hat{Y}_i = \hat{\beta}_0 + \hat{\beta}_1 X_i + \hat{\beta}_2 X_i^2 + \ldots + \hat{\beta}_n X_i^n}$
- **Sklearn.preprocessing, PolynomialFeatures** library is used for building a polynomial regression model
  - https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html
- Using the same linear example of Bond Prices (Y) and Inflation rate (X)
  - We first transform inflation (X) into a polynomial equation of the 3rd degree (X, X^2, X^3)
  - Then apply the linear regression model to it to predict Bond Price (Y)
- The results show no improvement over linear regression results and are still not accurate enough (RMSE 7.35 and -8.5% validation difference between actual and predicted value)
- This indicates that polynomial regression is not the best model for such linear relationships
- Generally, it is better for non-linear problems like signal processing

Chapter 8 – Regularization Regression

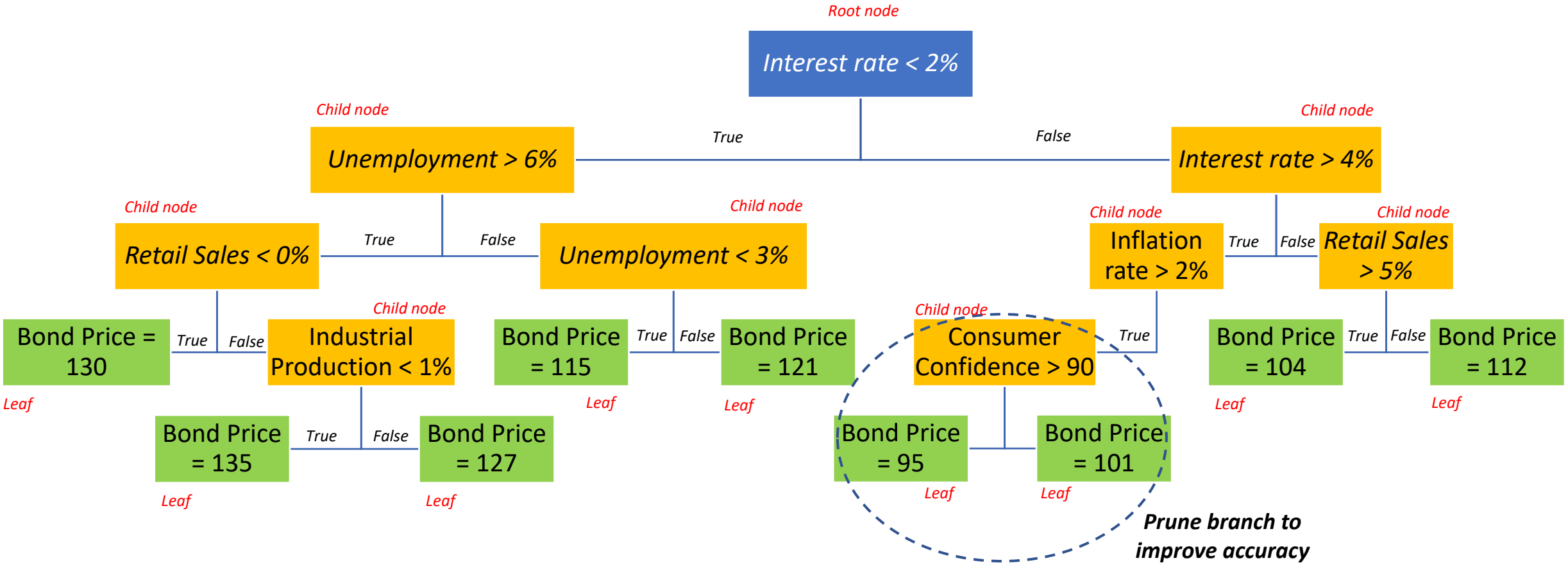
- Regularization regression is the process of adding a small amount of bias to a linear regression model to induce smoothness in order to prevent overfitting (complexity)
- This is done by adding a penalty term λ to the loss function RSS which penalizes increase in the unknown parameters β
- **Ridge regression or L2 regularization** modifies the loss function **RSS** by adding the **penalty term λ** which is a square function of the unknown parameters β
  - **RSS =** $\sum_{i=1}^{n} (Y_i - \beta_0 - \sum_{j=1}^{m} \beta_j X_{ij})^2$
  - **Ridge regression loss function = RSS +** $\lambda \sum_{j=1}^{m} \beta_j^2$
  - λ penalizes increase in β parameters in order to minimize the modified loss function
  - λ = 0 makes it a linear model. As λ -> ∞, β -> 0, increasing the penalty
  - This is how Ridge regression keeps the β parameters small
  - However, as β parameters are never equal to 0 all features will be selected, even insignificant one's
- **Lasso regression or L1 regularization** is similar to ridge regression except that the penalty term λ is a modulus function of the unknown parameters β
  - **Lasso regression loss function = RSS +** $\lambda \sum_{j=1}^{m} |\beta_j|$
  - For a large λ some of the parameters can be 0, so it also enables feature selection, improving interpretability
- Regularization can reduce the variance without significantly increasing bias, but only up to a limit
  - So, the right value of λ has to be selected
- We'll use the same data for predicting US 10 year govt Bond Prices (Y) using multiple US macroeconomic variables ($X_{1...n}$)
- The results are similar to multiple linear regression but they are still not accurate enough (RMSE 5.2 and -2.5% validation difference between actual and predicted value)

Chapter 8 – Decision Tree Regression

- **Decision trees** use a hierarchical tree-based structure with **parent and child nodes** connected by **branches** (conditions)
- They split the sample dataset into different subsets (branches) based on the result of conditions, starting with the **root node** at the top level
- Each node, depending on the **result of the conditions**, is further split into child nodes at the next lower level
- The algorithm traverses up and down different branches of the tree till only **leaf nodes** (no more splitting) remain by using recursion
- It is similar to human decision making
- The **conditions for splitting** are determined by the independent variables ($X_{1...n}$) at each level to estimate the dependent variable (Y) value at the leaf nodes
- Each complete traversal along one branch from root to leaf node results in one decision (Y value), using conditions on X values
- **Stopping criteria** is used for controlling tree growth by setting parameters like tree depth, no of samples for node split, etc
- **Metrics** used at each node to determine the direction to move in the tree (down, left or right Or stop) are
  - **Entropy, Information Gain and Gini Impurity**
- **Decision tree builds a model Y = f($X_{1...n}$)** that **minimises Information Gain and optimizes Gini Impurity to 1**
  - Y - Dependent variable, $X_{1...n}$ - Independent variables
  - It predicts output variable $\widehat{\boldsymbol{Y}}$ which is a real number
- **Entropy (H)** is a measure of information in a single random variable X
  - **H(X) = -** $\sum_{i=1}^{n}(\boldsymbol{pi} * \mathbf{log}\,\boldsymbol{pi})$. p_i is the probability of item with label i being chosen
  - Label can be a discrete class or group of similar values (real number)
  - H(X) is 0 at X = 0,1 and determines leaf nodes. Other values determine node splitting
  - Higher entropy means higher randomness in the information, which makes estimation difficult
- Useful website - https://scikit-learn.org/stable/modules/tree.html

Chapter 8 – Decision Tree Regression

Sample Decision Tree with multiple branches, nodes and leaves



Chapter 8 – Decision Tree Regression

- **Information Gain (IG)** is the amount information gained (reduction in entropy) about a random variable X by observing another random variable Z = z at the next state (using conditional probability)
  - IG(X,z) = H(X) - H(X|z) = Entropy(parent) – Sum of Entropy(children)
  - ***We want to increase information gain and reduce entropy***
- **Gini Impurity (GI)** measures the probability of a randomly chosen element from the set being incorrectly labelled due to the chosen subset (wrong class or group)
  - GI(p) = 1 - $\sum_{i=1}^{n} p_i^2$
  - For a set of items with m subsets, $i \in (1,2\ldots,n)$
  - p_i is fraction of items labelled with subset *i*
  - ***Closer the Gini Impurity is to 1 the better***
- Decision trees can be used for regression (continuous) and classification (discrete) problems
- They are easy to build and interpret, but can become complex and are sensitive to minor data changes and overfitting which then requires tree pruning
- **Pruning** involves removing branches of the tree that improve accuracy
- **Sklearn.tree, DecisionTreeRegressor** library is used for building the models
  - Some parameters for controlling tree growth are max_depth, min_samples_split, min_samples_leaf
- **The decision tree algorithm significantly improves on the results of multiple linear regression** for predicting Bond Prices (Y) using multiple macroeconomic variables data ($X_{1\ldots n}$)
  - RMSE 3.3 and -0.4% validation difference between actual and predicted value
- ***Note*** *- Decision trees due to their limitations are not widely used and have been replaced with Random Forest and XGBoost*

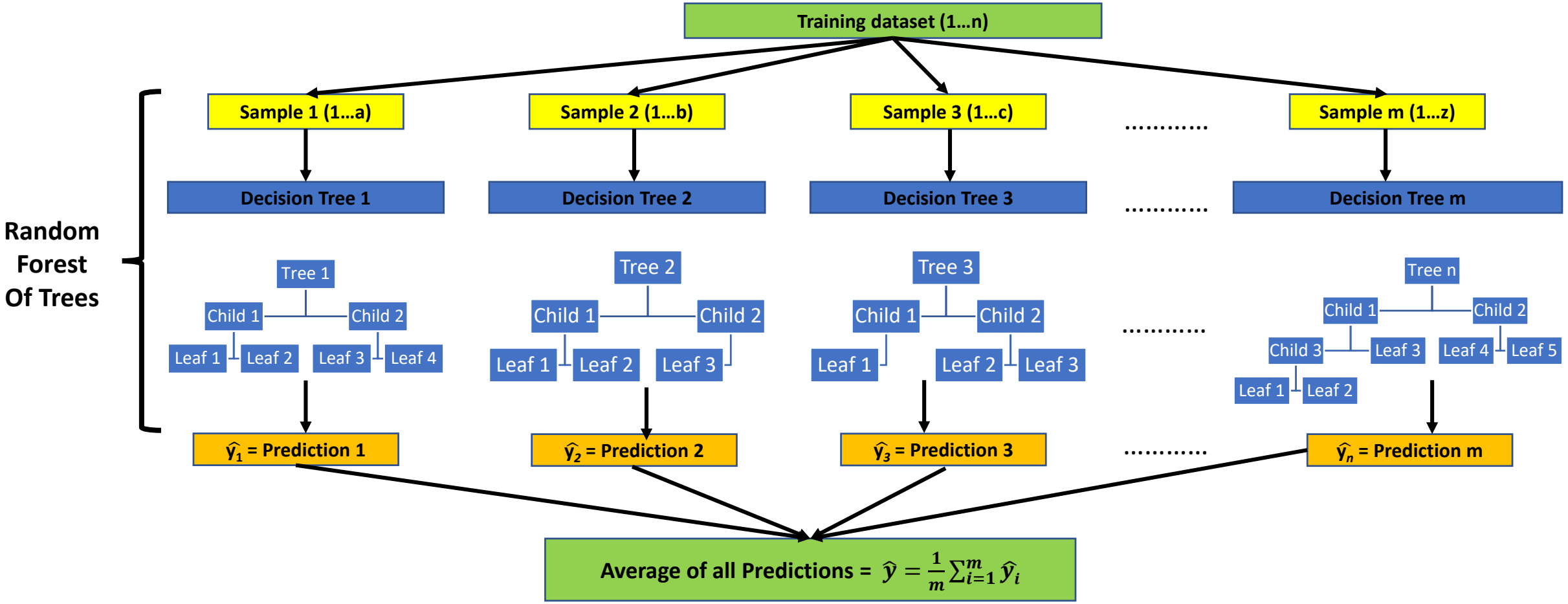
Chapter 8 – Ensemble Algorithms

- **Ensemble algorithms** combine the predictions of several estimators (***weak learners***), generally the same algorithm, to give a result better than using a single estimator
- The ensemble converts a group of weak learners into a ***strong learner***
- Two main ensemble methods are – bagging and boosting
- **Bagging** (Bootstrap aggregating) applies the same estimator to independent subsets of the sample, training them in parallel and takes the average of their predictions to build an ensemble that give better results with **reduced variance**
  - **Random Forest** – Uses decision tree as the estimator and averages the predictions of all trees
    - Most popular bagging technique
- **Boosting** builds the ensemble sequentially by applying the same estimator iteratively to the entire dataset, each time reducing the errors of the previous iteration and adding the new predictions to the previous iteration. It gives results with **reduced bias**
  - **Adaboost** (Adaptive Boosting) – increases/decreases the weight in the next iteration of each wrongly/correctly predicted observation from the previous iteration. Each observation is given the same weight at the beginning. The process makes the estimator focus more on the difficult to predict (wrong) observations than the easy ones in each iteration
  - **GradientBoost** – It applies the estimator iteratively to minimize the residual errors (predicted - actual) in the next iteration and combines the predictions with the previous iteration, like a gradient descent algorithm
  - **XGBoost** (Extreme Gradient Boosting) – It uses gradient boosted decision trees, adding a new tree in each iteration to optimize the loss function (like MSE). It uses regularization to reduce overfitting
    - Most popular boosting technique
- Random Forest and XGBoost are discussed in detail in the next slides
- Useful websites - https://scikit-learn.org/stable/modules/ensemble.html

Chapter 8 – Random Forest Regression

- Random Forest is an **ensemble of decision tree estimators that trains multiple estimators in parallel** on subsets of the dataset and then averages their predictions to reduce errors and get the best results
- Each tree is built from a **random sample drawn with replacement** from the training set (bootstrap sample)
  - Training set – 1…n, Decision Trees – 1…m. m < n
  - Sample 1 – 1…a, where a < n. Similarly, Samples 2,3 … m
- It is a random forest of decision trees, with each decision tree built using the process already described (tree nodes, splitting, children, leaves)
- **Random forest builds a model Y = f($X_{1…n}$)** that minimises Information Gain and optimizes Gini Impurity to 1
  - Y - Dependent variable, $X_{1…n}$ - Independent variables
  - $\widehat{\boldsymbol{y}} = \frac{\mathbf{1}}{\boldsymbol{m}} \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}} \widehat{\boldsymbol{y}}_{\boldsymbol{i}}$, $\hat{y}$ - is the average of all predictions, $\hat{y}_i$ is the prediction of each decision tree *i*
  - Output $\widehat{\boldsymbol{y}}$ is a real number
- Randomness is introduced in the ensemble by the sampling technique and other parameters like number of features used for splitting and tree depth
- Random forest model reduces the estimator variance by averaging the results of multiple decision tree estimators
- Use sklearn's GridSearchCV K-fold cross validation to train and validate each estimator on random independent subsets of the training data
  - Then averaging the estimates to reduce sampling error and improve accuracy

Chapter 8 – Random Forest Regression



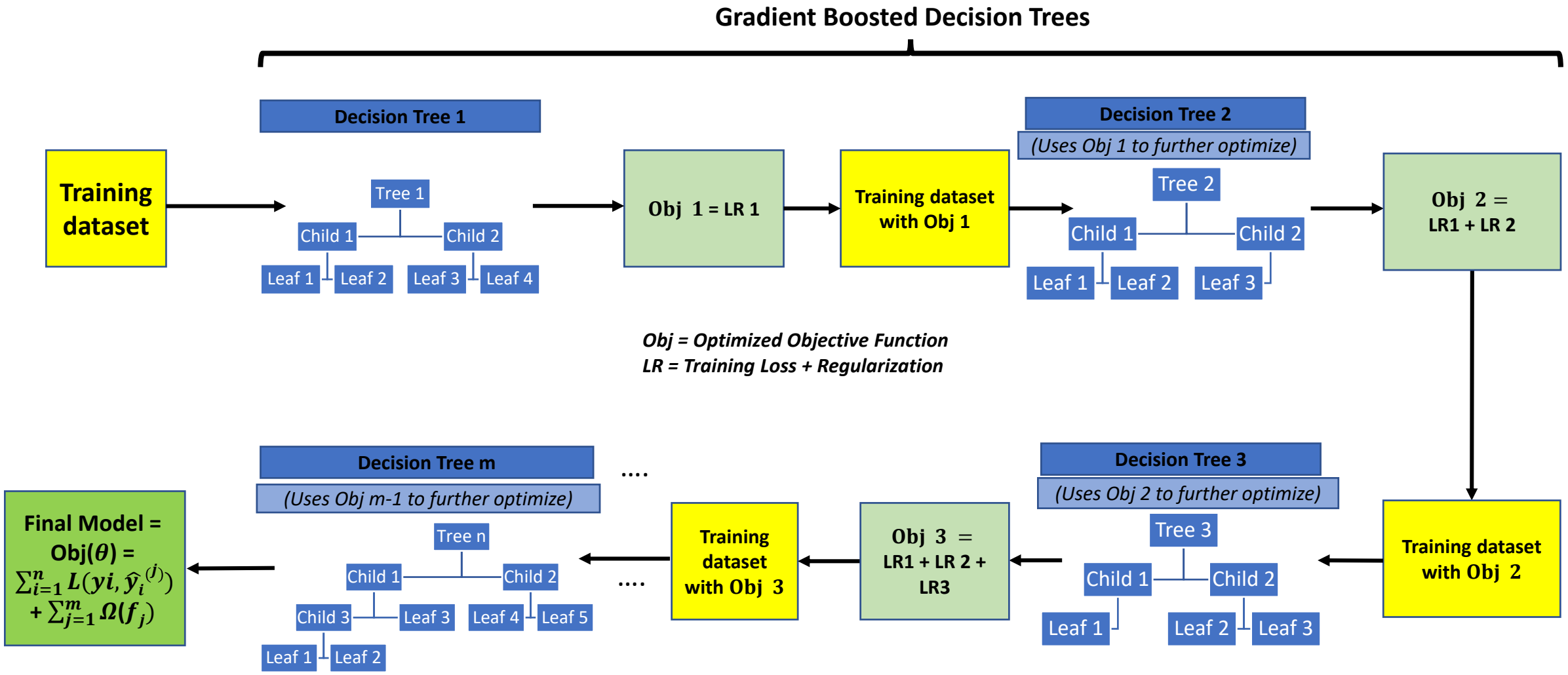
Chapter 8 – Random Forest Regression

- Sklearn.ensemble, RandomForestRegressor library is used for building the model
  - https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html#sklearn.ensemble.RandomForestRegressor
- **Parameters** can be set to get the best prediction and control tree growth. Useful one's are:
  - n_estimators = number of trees in the forest. Default is 100. More can be better but optimal value depends on data size and parameter tuning
  - max_features = number of features to consider for best split at each node (no of features, sqrt of no of features, etc)
  - n_jobs = used for parallel construction of trees. Value of -1 uses all available cores on the machine
  - max_depth = maximum depth of the tree
  - min_samples_split = minimum number of samples required to split a node
  - min_samples_leaf = minimum number of samples required at leaf node
- RandomForest model is good for **interpretation** as it has a **feature_importances_** attribute that gives a relative importance to each feature (X) with values between 0 and 1, total of all values being 1
  - Higher value indicates more importance of the feature in determining Y
- **The random forest algorithm further improves on the results of decision tree regression** for predicting Bond Prices (Y) using multiple macroeconomic variables data
  - RMSE 2.5 and -1.5% validation difference between actual and predicted value

Chapter 8 – XGBoost Regression

- XGBoost (Extreme Gradient Boosting) uses **gradient boosted decision trees**, adding a new tree sequentially in each iteration to optimize the **Objective function**
  - It consists of a Training Loss function L(θ) and a Regularization term $\Omega(\theta)$
  - **Obj(θ) = L(θ) + $\Omega(\theta)$**
  - θ – parameters of the model we learn from the data
- **Training Loss function L(θ)** is a prediction accuracy measure (predicted-actual) like MSE
  - L(θ) = MSE = $\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$
- **Regularization term $\Omega(\theta)$** controls model complexity by reducing overfitting to the data (using bias-variance tradeoff)
- XGBoost uses an additive strategy by fixing the previous iteration's errors in the next iteration by adding a new tree
  - It does not use a traditional gradient descent algorithm by calculating the gradient (convex curve, where loss function goes down/up the slope to minimize the loss)
- **XGBoost builds a model Y = f($X_{1...n}$)** that optimizes the objective function Obj(θ)
  - Y - Dependent variable, $X_{1...n}$ - Independent variables
  - Training set – 1…n, Decision Trees – 1…m. m < n.
  - y_i - actual values, $\hat{y}_i$ - predicted values, $\hat{y}_j$ - prediction of each decision tree
  - $\hat{y}_i = \sum_{j=1}^{m} f_j(xi)$. Training data *i = 1…n*, No of trees *j = 1…m.* m < n.
  - **Obj(θ) = $\sum_{i=1}^{n} \boldsymbol{L}(y_i, \hat{y}_i^{(j)}) + \sum_{j=1}^{m} \boldsymbol{\Omega}(f_j)$**
  - $\hat{y}_i^{(j)}$ – is the prediction value for tree *j*
- It is the most popular implementation of gradient boosting for both accuracy and speed
  - It does not train multiple trees in parallel like random forest
  - It does parallel processing within each tree by building each branch in parallel
  - It performs in memory sorting and splitting of all features once in parallel, which are then stored and reused again
- Useful website - https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Chapter 8 – XGBoost Regression



Chapter 8 – XGBoost Regression

- You can use XGBoost's cv method (n-fold cross validation) to train and validate each tree on multiple independent subsets of the training data and averaging the estimates to reduce sampling error and improve accuracy
  - We will use sklearn GridSearchCV library for k-fold cross validation, making xgboost part of the model pipeline
- **Xgboost, XGBRegressor** library is used for building the model
- Three types of parameters can be set to get the best predictions. Useful one's are:
  - **General** - which boosting model to use - tree or linear
    - booster – gbtree, gblinear or dart
  - **Booster** - depends on booster model
    - Lot of booster model related parameters
    - max_depth – maximum depth of a tree. Higher values increases complexity/overfitting
  - **Learning task –** to specify the learning objective
    - objective – reg:squarederror(regression with squared loss), reg:squaredlogerror (regression with squared log loss), etc
    - eval_metric – evaluation metric for validation data, default according to objective. rmse for regression
- XGboost model is good for **interpretation** as it has a **feature_importances_** attribute that gives a relative importance to each feature (X) with values between 0 and 1, total of all values being 1
  - Higher value indicates more importance of the feature in determining Y
- **XGboost regression results are similar to the results of random forest regression** for predicting Bond Prices (Y) using multiple macroeconomic variables data
  - RMSE 3.0 and -1.5% validation difference between actual and predicted value

Chapter 8 – Regression Pipeline and GridSearchCV

- Use **Pipeline and GridSearchCV** libraries to build, optimize and evaluate multiple regression algorithms at once and then select the best one
- **Pipeline** combines data transformations and training multiple models into a sequence of steps
  - **Grid search** across entire hyperparameter grid to select the best combination for each model
  - **Hyperparameters** to be tuned for each model are setup as a predefined parameter array (grid)
- **GridSearchCV** performs **k-fold cross-validation** to train models on multiple variations of the training data of the sample in an iterative loop (k-folds, k-splits)
- The results of applying **GridSearchCV** algorithm on a **pipeline** of **different regression models** to predict US bond prices using US macroeconomic data are shown
- The **regression algorithms considered** are
  - Multiple Linear Regression
  - Regularization – L1 Lasso and L2 Ridge
  - Random Forest Regression
  - XGBoost Regression
- A **10-fold cross-validation** splitting strategy is used to ensure multiple variations of the training data are used (better data sampling)
- Sklearn_metrics are used to evaluate and measure the accuracy of the results
- The results are **better for all models**
  - The hyper-tuned parameters can now be used for each model
- **Random Forest and XGBoost still give the best results** compared to all other models
- Useful websites
  - https://scikit-learn.org/stable/modules/cross_validation.html
  - https://scikit-learn.org/stable/modules/grid_search.html
  - https://scikit-learn.org/stable/modules/compose.html

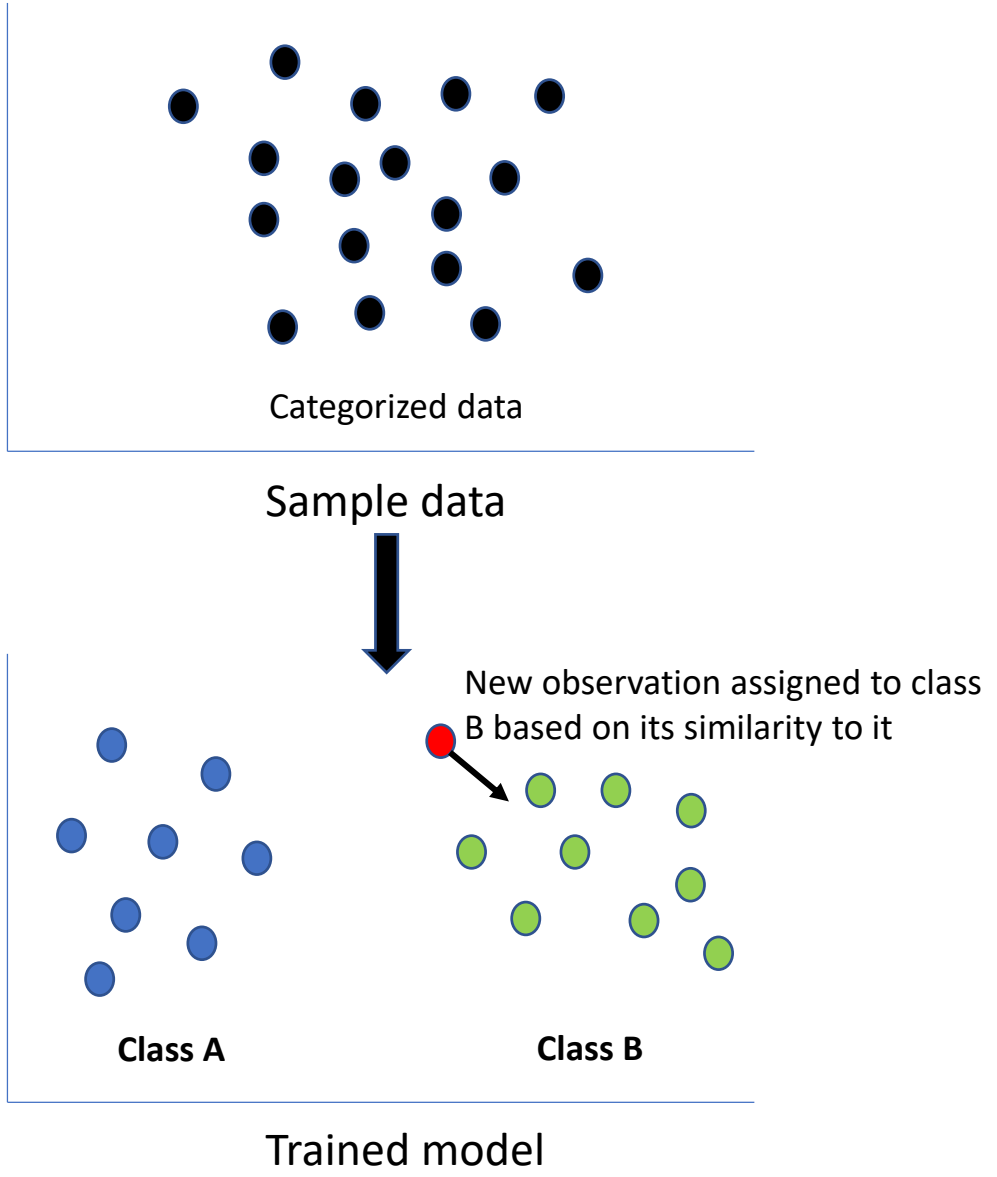
Optimal parameter grid:

RandomForestRegressor - Best grid params:
{'rgr__criterion': 'mse', 'rgr__max_depth': 10,
'rgr__max_features': 'sqrt',
'rgr__min_samples_leaf': 1,
'rgr__min_samples_split': 2,
'rgr__n_estimators': 200,
'rgr__random_state': 42}
'RandomForestRegressor_rmse': 2.45
'RandomForestRegressor_r2': 0.93

XGBoostRegressor - Best grid params:
{'rgr__learning_rate': 0.1, 'rgr__max_depth':
5, 'rgr__objective': 'reg:squarederror',
'rgr__seed': 1}
'XGBoostRegressor_rmse': 2.39,
'XGBoostRegressor_r2': 0.93

Chapter 9 - Supervised Learning Algorithms - Classification

- **Classification algorithms are used for predicting a class or category for an observation based on the training from a sample dataset**
  - **The algorithm is called as a classifier (or model)**
  - It is used for predicting discrete values or classes (0,1,2 or 'True', 'False' etc) instead of continuous values
  - Y is the predicted class (dependent variable) and $X_{1...n}$ is the input feature set (independent variables or vector)
- **Training set** - The classifier requires labeled training data containing class labels (Y) for each observation (feature set $X_{1...n}$) to learn from
  - The training data is already categorized (correct class)
- **Test set** - Contains only input features ($X_{1...n}$) and is used by the classifier for predicting the class labels (Y)
  - Test set is used for testing the accuracy of the trained classifier
- The objective is to build a classifier that can assign a new observation to the correct class or category
- The classifier does this by using **similarity measures** like Euclidean distance, cosine of angle, etc between the new observation and the different classes
  - An observations feature set is converted into a vector of points whose distance or angle can now be measured from the class



Chapter 9 - Supervised Learning Algorithms - Classification

- Classification is generally of 2 types
  - **Binary –** Data is categorized into one of 2 classes (0 or 1, ‘True’ or ‘False’, etc)
  - **Multiclass -** Data is categorized into on of the many classes (0,1,2,3; ‘City1’, ‘City2’, ‘City3’, etc)
- Some common examples for classification are:
  - Assign 0 or 1 to each observation based on a condition being true or false
  - Assigning each email to ‘spam’, ‘not spam’ category by using prior spam email data
  - Detecting the ‘presence’ or ‘absence’ of a disease by using prior disease symptom data
- **Popular classifiers** are Logistic regression, K-Nearest Neighbours (K-NN), Support Vector Machines (SVM), Naïve Bayes, Decision Trees, Neural Networks, etc
- Classifiers use **error metrics** like **confusion matrix** (True/False Positives/Negatives), **ROC curve**, etc
- As all machine learning classifiers work with numbers and since the data can be numeric (integer or real), ordinal (‘true’, ‘false’) and categorical (text like city, group type, etc), it needs to be transformed using the Chapter 7 - Data wrangling steps (encoding, binning, etc)
- As classifiers are similarity or distance based, they generally require standardization (scaling), already covered in Chapter 6 - Data science process
  - Data has to be centred around mean 0, variance 1 (normalization) or mean with variance of the same order
  - Data with higher order variance (outliers) might get a higher weight in the model, causing bad predictions on new data
  - Only training data needs to be standardised and not test data

Chapter 9 - Classification – Error Metrics

- A **Confusion matrix** is used to evaluate the accuracy of a classifier, as we need to know how many values are predicted correctly for each class label
  - Rather than a single measure of accuracy or error
- A **NxN table** is used to summarize the results of the confusion matrix. N is the no of classes in the classifier
  - Each row represents values in the actual class
  - Each column represents values in the predicted class
  - The table cells contain the count of class labels that are predicted correctly or incorrectly
- For example, a binary classifier has a 2x2 table matrix
  - True, False representing 1, 0 class labels for Y
- Using the predicted and actual class values, the row/column intersection represents the count of values that are called
  - **True Positives (TP)** – Predicted True, Actual True
  - **True Negatives (TN)** – Predicted False, Actual False
  - **False Positives (FP)** - Predicted True but Actual False – Type 1 error
  - **False Negatives (FN)** - Predicted False but Actual True – Type 2 error
  - Type 1, Type 2 error importance depends on the problem (disease, fraud, etc)
- **Other** measures of accuracy are
  - **Accuracy rate** = Correct predictions (TP+TN) / Total predictions
  - **Error rate** = 1 - Accuracy Rate
  - **Precision** = TP / (TP+FP). What ratio of correctly predicted values are true
  - **Sensitivity or Recall or True Positive Rate TPR** = TP / (TP+FN). What ratio of actual true values are correctly predicted
  - **False Negative Rate FNR** = FN / (TP+FN). What ratio of actual true values are incorrectly predicted
  - A high TPR and low FNR means more correctly predicted true values

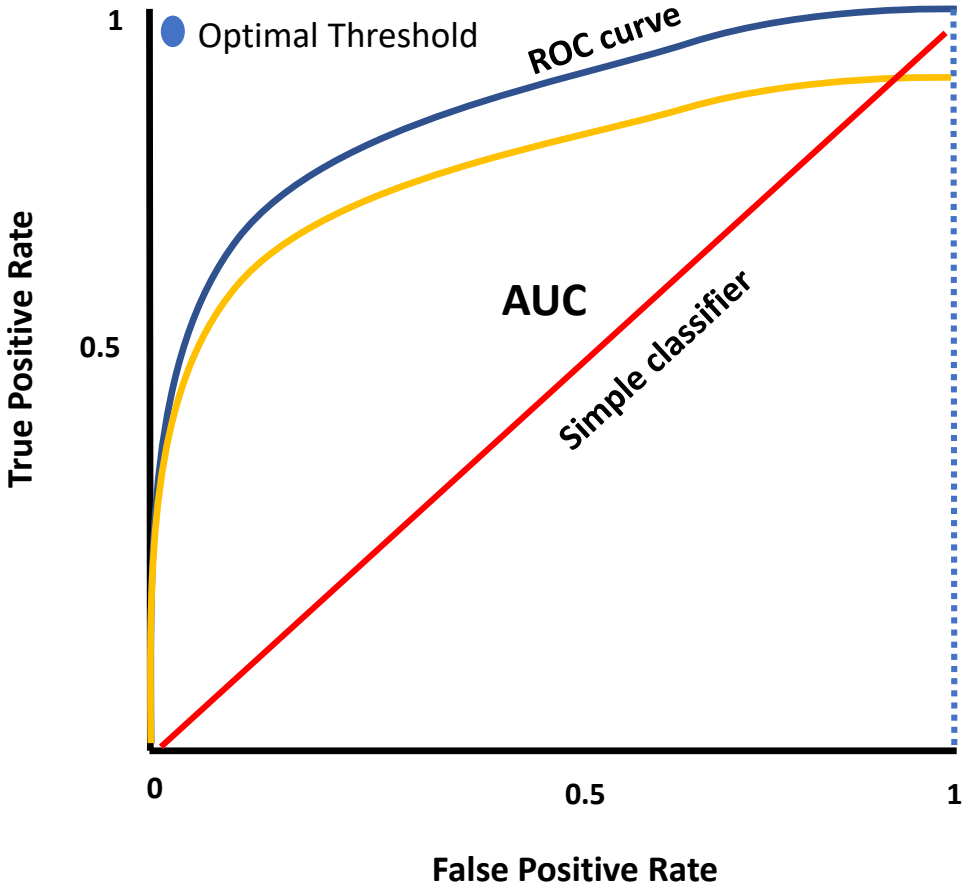
		Predicted Values	
		TRUE	FALSE
Acual Values	TRUE	TP	FN
	FALSE	FP	TN

Confusion Matrix

Chapter 9 - Classification – Error Metrics

- **Specificity or True Negative Rate TNR** = TN / (TN+FP). What ratio of actual false values are correctly predicted
- **False Positive Rate FPR** = FP / (TN+FP) = 1-Specifity. What ratio of actual false values are incorrectly predicted
- A high TNR and low FPR means more correctly predicted false values

- **ROC (Receiver Operating Characteristic) curve** is used for evaluating binary classifiers
  - It is a probability plot of **True Positive Rate** against **False Positive Rate** for different threshold values
  - It is a plot of sensitivity vs 1-specifity, indicating how sensitive the classifier is to false positives
  - It is used for **selecting the optimal model** by varying the decision threshold value used by the classifier
  - It is used for cost-benefit analysis when selecting the optimal decision threshold
  - The steepness of a ROC curve is increased to maximize the true positive rate while minimizing the false positive rate
  - The top left corner of the plot is the 'optimal threshold' where TPR = 1 and FPR = 0
  - This means all true and false values are correctly classified (no false positives)
- **AUC curve** – is the Area Under the ROC Curve
  - It is the probability that the classifier ranks true values higher than false values. It is a summary of the ROC curve
  - AUC varies from 0 to 1. The larger the area under the curve the better the classifier
- Useful website - https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics



Chapter 9 – Classification – Sample data for problems

Source
https://data.world/exercises/logistic-regression-exercise-1

Description
Dataset for practicing classification -use NBA rookie stats to predict if player will last 5 years in league

Summary
Classification Exercise: Predict 5-Year Career Longevity for NBA Rookies
y = 0 if career years played < 5
y = 1 if career years played >= 5

Data Dictionary
Given

Data Policy
Shared with Everyone. https://data.world/policy/terms/

	Description
Name	Name
GP	Games Played
MIN	MinutesPlayed
PTS	PointsPerGame
FGM	FieldGoalsMade
FGA	FieldGoalAttempts
FG%	FieldGoalPercent
3P Made	3PointMade
3PA	3PointAttempts
3P%	3PointAttempts
FTM	FreeThrowMade
FTA	FreeThrowAttempts
FT%	FreeThrowPercent
OREB	OffensiveRebounds
DREB	DefensiveRebounds
REB	Rebounds
AST	Assists
STL	Steals
BLK	Blocks
TOV	Turnovers
TARGET_5Yrs	Outcome: 1 if career length >= 5 yrs, 0 if < 5...

Chapter 9 – Classification – Multiclass Sample data

Source
https://www.kaggle.com/scarecrow2020/tech-students-profile-prediction

Description
- Tortuga Code is an online educational platform, oriented over programming and technology
- Tortuga Code is looking for a recommender system that assigns students to a specific category depending on their developer profile.

Summary
Multiclass Classification: Predict student developer category

Data Dictionary
Given

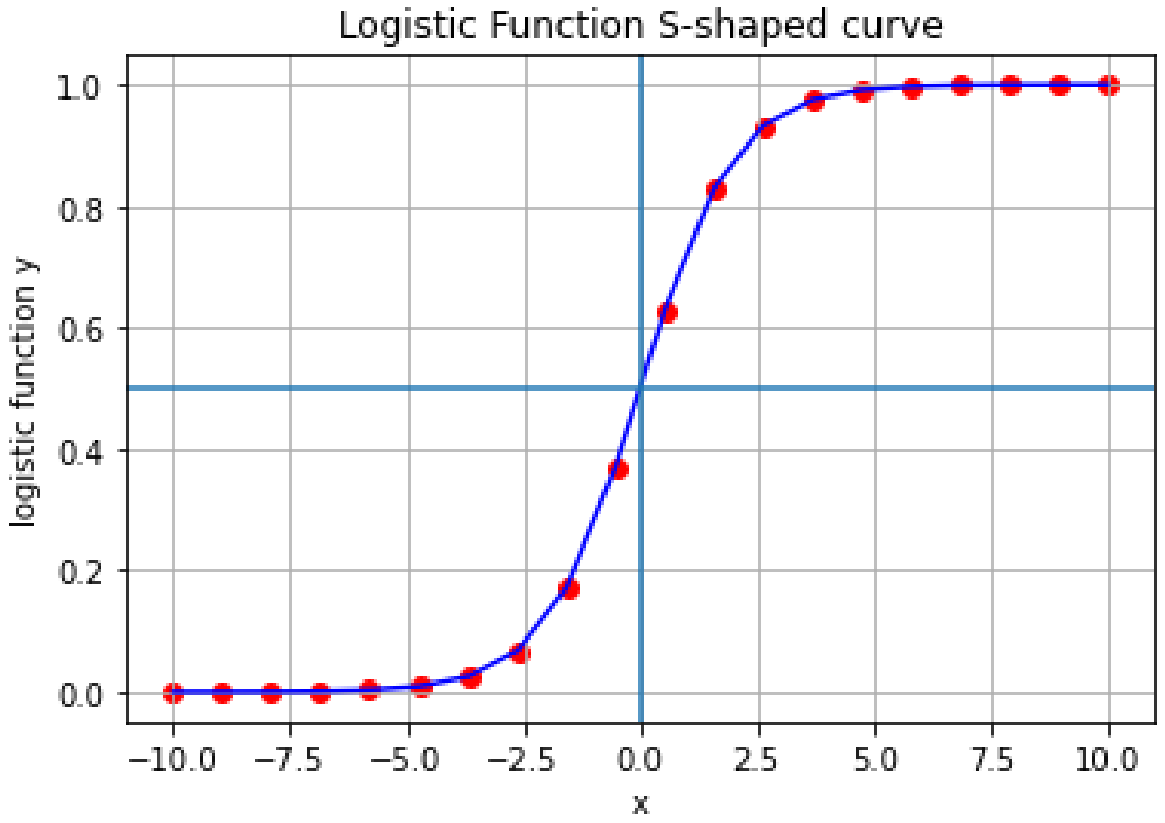
Data Policy
This Kaggle data is from IBM Maratona, distributed using Apache License 2.0
CC0: Public Domain

Features (X)
Unnamed: 0 - Useless column
NAME - Name of the student
USER*ID - ID for each student*
*HOURS*DATASCIENCE - Numbers of hours studied data science courses
HOURS*BACKEND - Numbers of hours studied web (Back-End)*
*HOURS*FRONTEND - Numbers of hours studied web (Front-End)
NUM*COURSES*BEGINNER*DATASCIENCE - Numbers of beginner courses of Data Science completed by the student*
*NUM*COURSES*BEGINNER*BACKEND - Numbers of beginner courses of Web (Back-End) completed by the student
NUM*COURSES*BEGINNER*FRONTEND - Numbers of beginner courses of Web (Front-End) completed by the student*
*NUM*COURSES*ADVANCED*DATASCIENCE -Numbers of advanced courses of Data Science completed by the student
NUM*COURSES*ADVANCED*BACKEND - Numbers of advanced courses of Web (Back-End) completed by the student*
*NUM*COURSES*ADVANCED*FRONTEND - Numbers of advanced courses of Web (Front-End) completed by the student
AVG*SCORE*DATASCIENCE - Average score in Data Science completed by the student
AVG*SCORE*BACKEND - Average score in Web (Back-End) completed by the student
AVG*SCORE*FRONTEND - Average score in Web (Front-End) completed by the student

Target (Y)
PROFILE - Tech profile of the students
- beginner_front_end - advanced_front_end - beginner_back_end - advanced_back_end - beginner_data_science - advanced_data_science

Chapter 9 – Classification – Logistic Regression

- **It is a linear model for classification problems**
  - *Not a regression model despite its name*
- **It builds a logistic model that assumes a linear relationship between the *probability of the log-odds*** of the dependent variable Y (output class label) and the Independent variables $X_{1...n}$ (input)
  - The class labels are distinct values - 0,1 for binary or 0,1,2…n for multiclass classification
- **The logistic model estimates the class label of Y by**
  - **First using a logistic function** to calculate the *probability of the log-odds of* Y varying between (0,1)
  - **Then using a decision threshold to** convert the probability between (0,1) into a distinct class label (0,1,2, etc)
- A **logistic function (sigmoid function)** is an S-shaped curve, with real number inputs between (-∞, +∞) and output probability between (0,1)
  - The equation for a **standard logistic function f(y) =** $\frac{1}{1+e^{-y}}$, ***e*** *is the exponential function (growth/decay)*
- A plot of a simple logistic function f(y) for x between (-10, 10) is shown
- Useful website - https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression

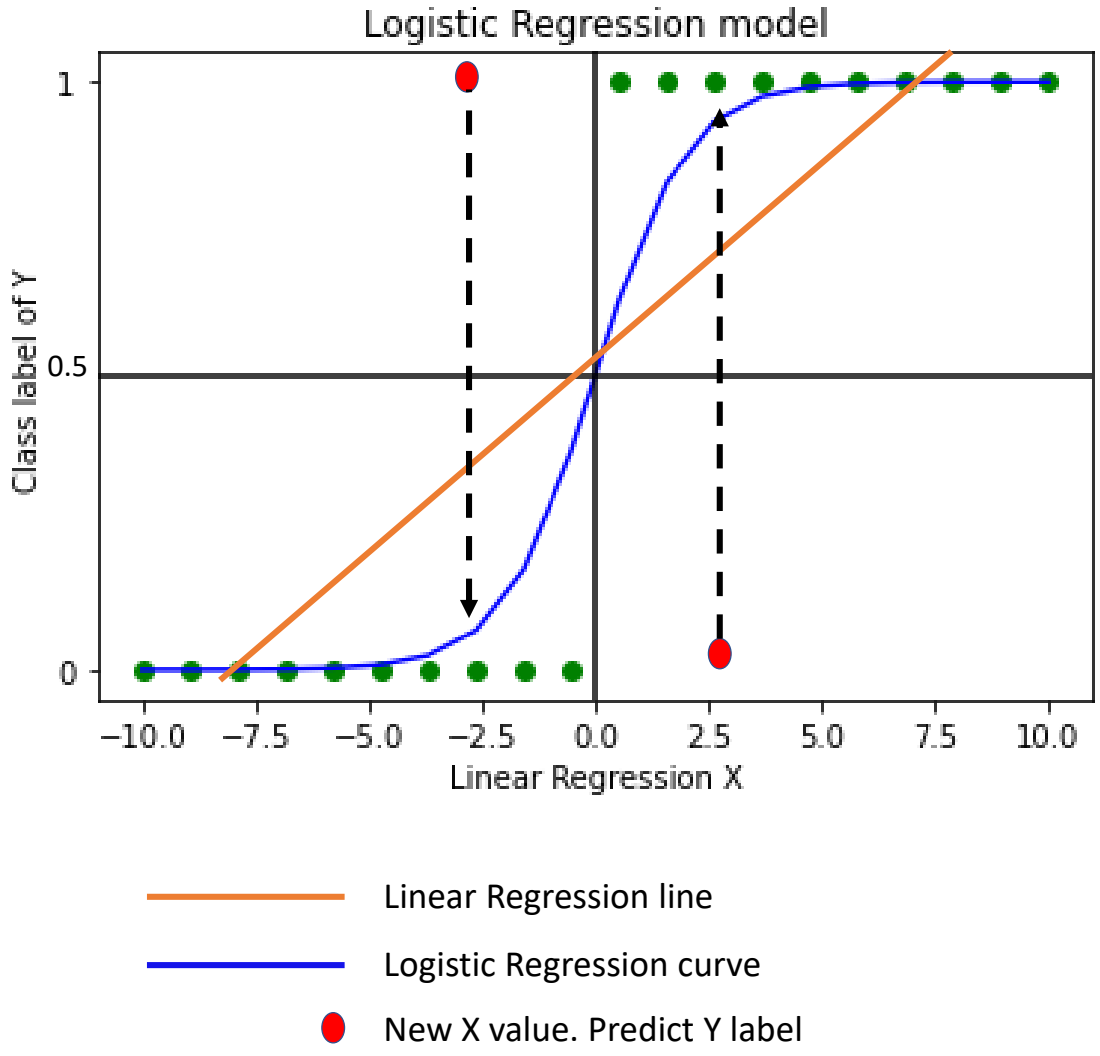


Chapter 9 – Classification - Logistic Regression

- To understand this, consider a binary classification problem where Y is either 0 or 1
- $Y = \beta_0 + \sum_{i=1}^{n} \beta_i X_i$ is a linear regression equation for Y dependent on multiple independent variables $X_{1...n}$
- **P(Y=1|X) = p =** probability of the dependent variable Y=1 occurring, given X independent variables
- A **logit function,** which is the inverse of a logistic function, is used to derive the logistic regression equation
  - It is used to calculate the log-odds of the probability by taking the logarithm of the odds of the probability
  - Given p, **Odds =** $\frac{p}{1-p}$ **=** $\frac{\text{event happening } (Y=1)}{\text{event not happening } (Y=0)}$
  - So, **Logit(p) =** $\log_e(\frac{p}{1-p})$
    - Normally natural logarithm e is used but it can be any base (2, 10, etc)
- As the log-odds of the probability is assumed to be equal to a linear function, $\log_e(\frac{p}{1-p}) = \beta_0 + \sum_{i=1}^{n} \beta_i X_i$
  - By using exponential function, we can transform logit(p) to get $\frac{p}{1-p} = e^{(\beta_0 + \sum_{i=1}^{n} \beta_i X_i)}$
- Applying algebra, we get $\mathbf{p = \frac{1}{1 + e^{-(\beta_0 + \sum_{i=1}^{n} \beta_i X_i)}}}$ **= f(y), a logistic function of a regression equation**
  - p is the probability varying between (0,1)
  - β - the unknown regression coefficients will be estimated using maximum likelihood estimation (covered later)
- ***Conclusion*** - This shows that while the linear regression equation (input) can vary from (-∞, +∞), probability p (output) varies between (0,1) being the result of a logistic function of a linear regression equation

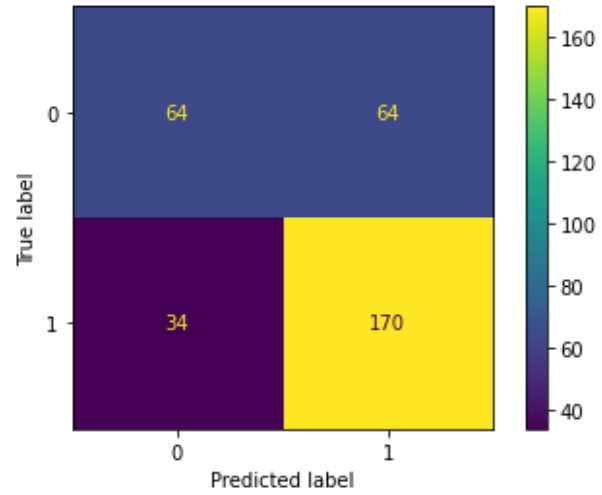
Chapter 9 – Classification - Logistic Regression

- Now, using a **decision threshold (cut-off)** on the probability p the logistic regression model assigns a discrete value of 0 or 1 as a class label for Y
  - Eg: If p >= 0.5, class label = 1. Else class label = 0
  - 0.5 is the default threshold for binary classification
  - However, the threshold value can be optimized to get the best threshold for class labels
- Logistic regression model graph for X (-10,10) is shown
- Comparing logistic regression to linear regression, the model converts a non-fitting straight line into a best fit S-shaped line
- We can now predict the class label for new X values depending on where they fall on the best fit logistic model curve
- The **benefit of using a logistic regression model** is that it improves interpretability by transforming large changes in X into a smaller range for Y
  - While the odds increase exponentially (for a unit increase in X and β, the odds increase by $\boldsymbol{e^{\beta X}}$)
    - As Odds = $e^{(\beta_0 + \sum_{i=1}^{n} \beta_i X_i)}$
  - The log-odds increase at a constant rate (logarithmic function)
  - Causing only a small increase in probability *p*



Chapter 9 – Classification - Logistic Regression

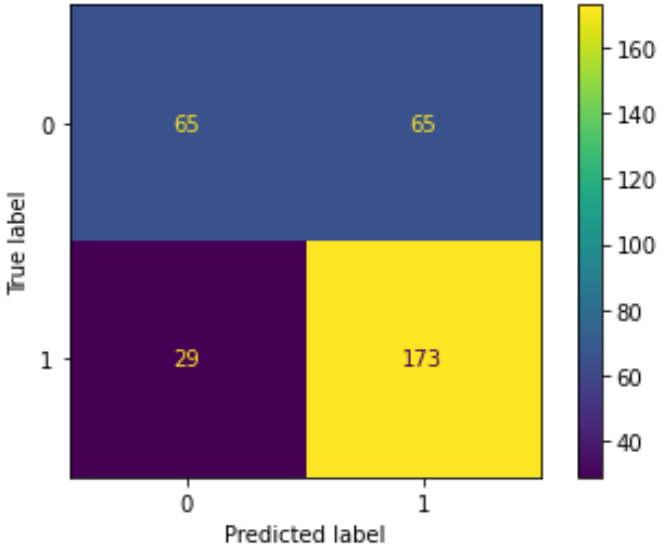
- Logistic Regression can be used for **multiclass classification by using the 'one-vs-rest' or 'one-vs-all' strategy**
  - **'one-vs-rest'** trains a single classifier per class, with the samples of that class labelled as 1 and all other samples as 0
  - It is preferred for multiclass classification
- **Sklearn.linear_model, LogisticRegression** library is used for building the model
- It implements regularized logistic regression using different solvers
- **Parameters** can be set to get the best predictions. Useful one's are:
  - **Penalty** - **norm** used in the regularization penalty ('l1', 'l2', etc)
    - Norm – vector norm (p) or magnitude of n dimensional vector x. $|\mathbf{x}|_p = (\sum_{i=1}^{n} |xi|^p)^{1/p}$
  - **C** – Inverse of regularization strength, smaller values specify stronger regularization
  - **Solver** – Algorithm to use in the optimization problem ('lbfgs', 'liblinear', 'sag', etc)
- **Sklearn_metrics - confusion_matrix, accuracy_score and classification_report** are used to evaluate and measure the accuracy of the results
  - The confusion matrix (TP,TN,FP,FN numbers) in a grid is a good visual way to evaluate accuracy
  - The accuracy rate and precision are generally good measures
  - The precision results show different values for class labels (0,1)
- The results of the algorithm on the NBA rookie stats to predict if a player will last 5 years in league **show good results (~ 72%)**
- The **results can be improved** by using other algorithms and **GridSearchCV** for better data sampling and model training using hyperparameter tuning
- LogisticRegression model is good for **interpretation**

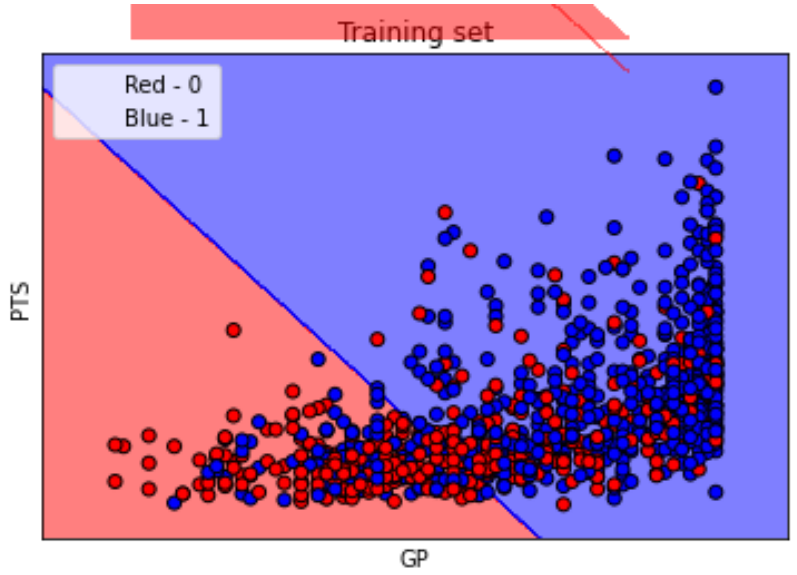


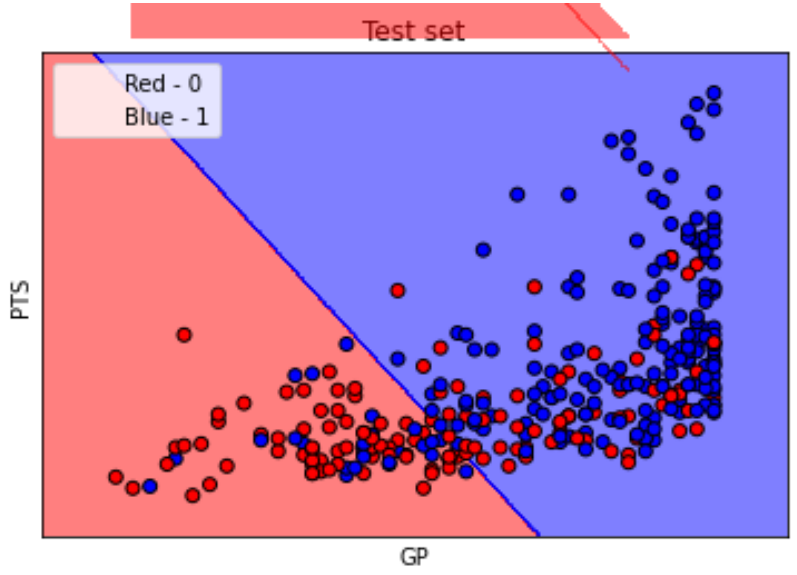
	precision	recall	f1-score	support
0	0.65	0.5	0.57	128
1	0.73	0.83	0.78	204

Chapter 9 – Classification - Logistic Regression

- Another way to visualize the **scatter plot of the results with a best fit decision boundary splitting** them using a numpy **meshgrid** is given
  - Meshgrid converts the 1-D array into
  - Being 2 dimensional we can use **only 2 features (X independent** variables)
  - Results are still quite good, despite using only using 2 X independent variables
  - Indicating that they are the most important features
  - Separate python code is given for it

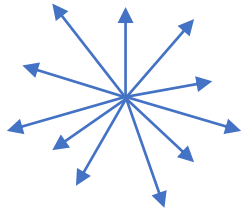




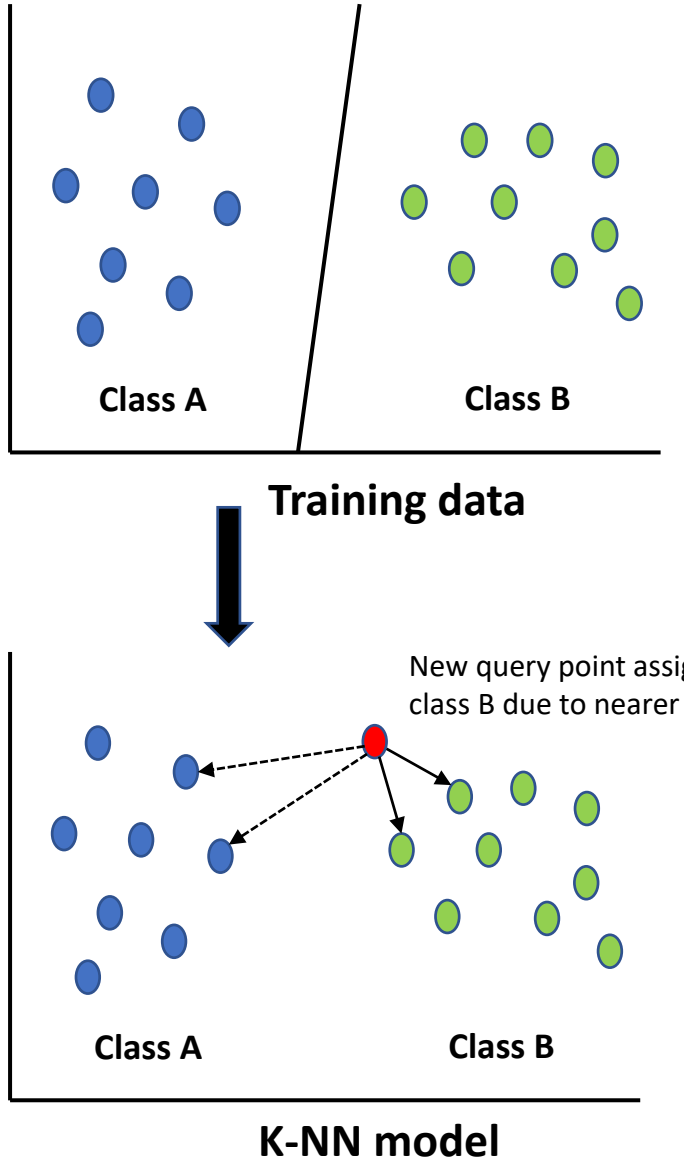


Chapter 9 – Classification – K-Nearest Neighbors

- **K-Nearest Neighbors (K-NN)** is an instance-based non-parametric classifier
  - It does not involve estimation of parameters to build a model using training data
  - It does not make any assumptions about the model function **$Y = f(X_{1...n})$**, between the dependent variable Y (output class label) and the independent variables $X_{1...n}$ (inputs)
  - It is a non-linear classification model
- **It stores in memory instances of all training data** as n-dimensional vectors in a multidimensional feature space, with labelled classes
  - Each observation is represented as a vector of independent variables ($X_{1...n}$)
  - It uses fast indexing structures like KDTree or Ball tree to overcome speed and memory issues due to large instances of the training data
- **A Class label is assigned** to an **unlabeled observation (query point) based on a simple majority vote of the nearest neighbors** (most common class in k neighbors nearest to it)
- It **uses Euclidean distance to calculate nearness of the query point to its k neighbors**
  - **k** is a user defined integer. Optimized using hyperparameter tuning
  - Higher k value reduces the impact of noise
  - Normally same weight is assigned to each neighbor of the query point
  - Sometimes higher weight is assigned to nearer neighbors by using the inverse of distance from the query point to get a better fit

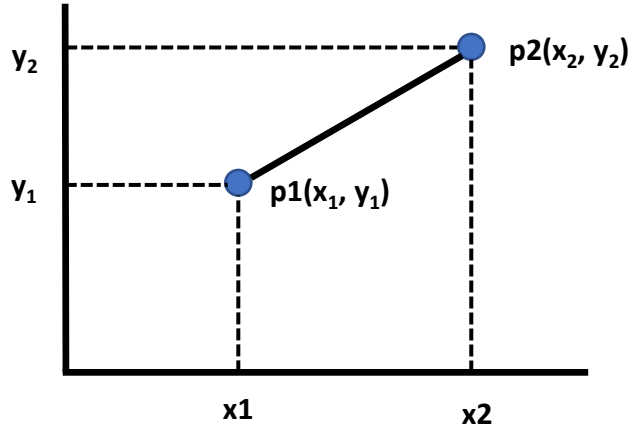


N-dimensional vector space

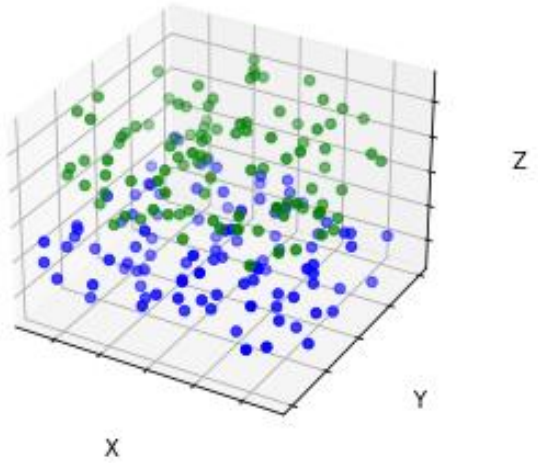


Chapter 9 – Classification - K Nearest Neighbors

- Being distance based it requires standardization (scaling) of data to prevent giving higher weight to independent variables with larger scales
- It is useful for problems where the decision boundary is irregular (not separable by a linear hyperplane)
- Algorithms used to calculate the distance are:
- **Brute force** - **Euclidean** distance is calculated between all pairs of points. Inefficient for large sample sizes
- **KD Tree (**K dimensional tree) – It reduces number of distance calculations by encoding aggregate distance information
  - By using the following logic, without explicit distance calculations, If point A is far from point B and B is near C, then A is far from C
  - It is a binary tree that recursively partitions the data along cartesian axes, as nested orthotropic regions (3 mutually perpendicular planes)
- **Ball Tree** – Better for higher dimensions. Recursively partitions the data into a series of nested hyper-spheres
  - Each hyper-sphere, defined by radius r and centroid C, contains multiple nodes each containing points within the hyper-sphere
  - It uses triangle inequality to reduce the number of neighbor searches |x + y| <= |x| + |y|
    - Sum of the length of 2 sides of a triangle must be <= length of 3rd side
- Useful website:
  - https://scikit-learn.org/stable/modules/neighbors.html



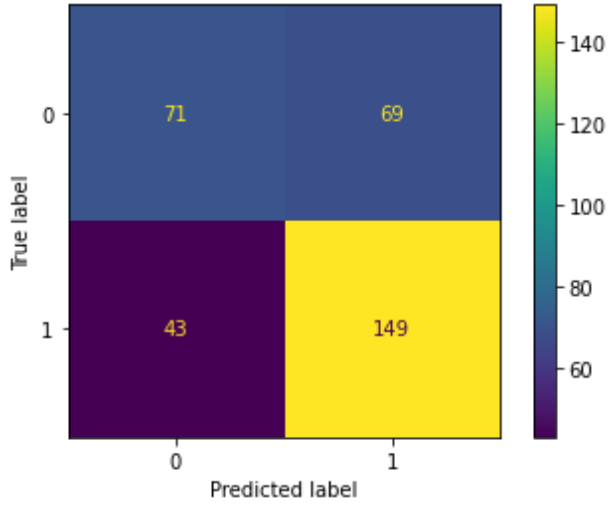
Euclidean distance between 2 points p1 and p2 = $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$



https://www.freepik.com/free-vector/3d-background-with-black-spheres_948648.htm

Chapter 9 – Classification - K Nearest Neighbors

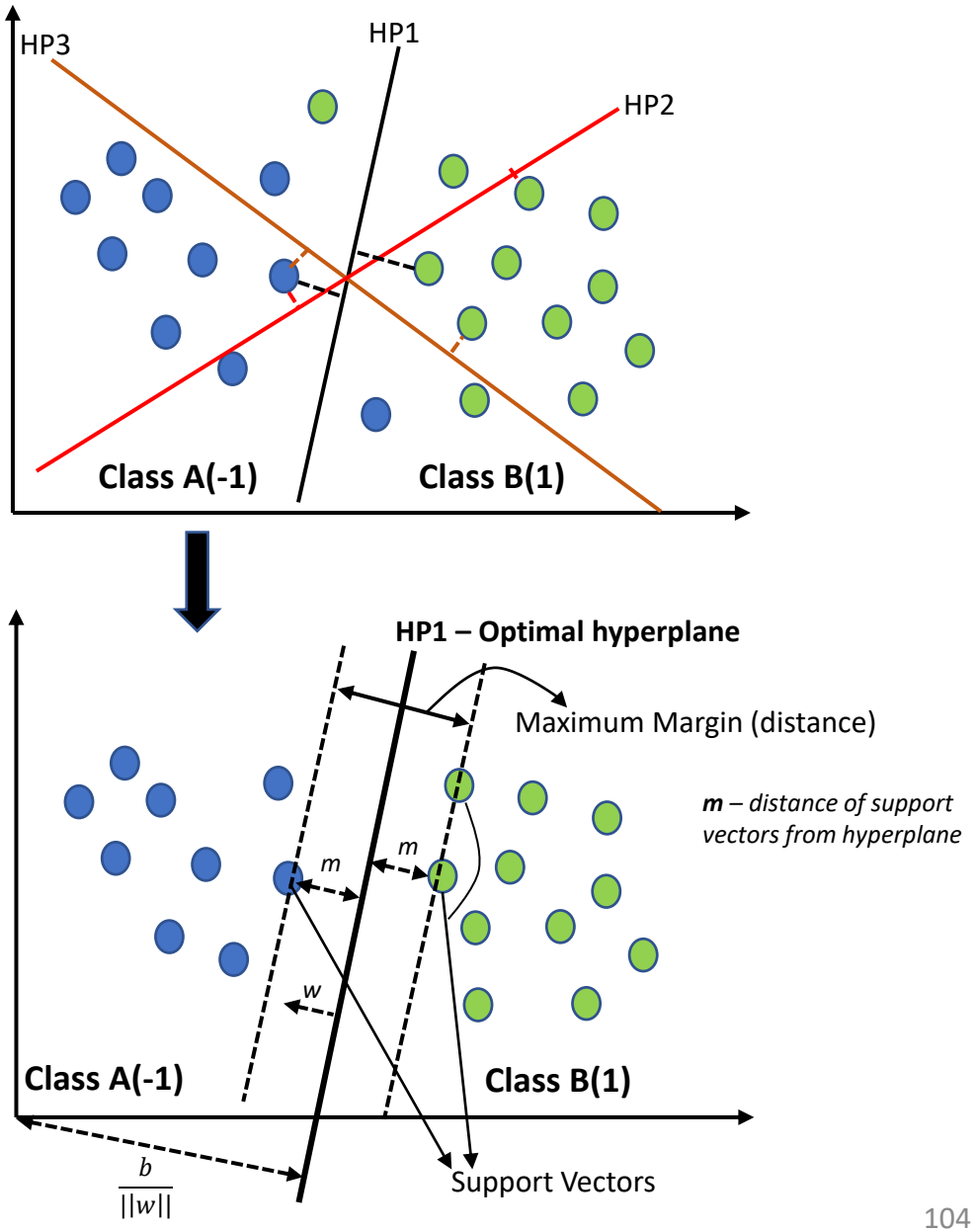
- **sklearn.neighbors, KNeighborsClassifier** is used for building the model
- Some useful **parameters** that can be set to optimize the classification are:
  - **n_neighbors** - no of neighbors to use. Default = 5
  - weights – weight function to use for each point
    - 'uniform' - same weight to each point
    - 'distance' - inverse of distance from each point as weight
  - algorithm – to compute distance ('auto', 'ball_tree', 'kd_tree', 'brute')
  - metric – distance metric for the tree. Default = 'minkowski' (4D space)
  - p – power parameter for minkowski metric
    - 1 = Manhattan distance
    - 2 = Euclidean distance
- **Sklearn_metrics** - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The confusion matrix (TP,TN,FP,FN numbers) in a grid is a good visual way to evaluate accuracy
  - The accuracy rate and precision are generally good measures
  - The precision results show different values for class labels (0,1)
- The results of the algorithm on the NBA rookie stats to predict if a player will last 5 years in league **are worse than logistic regression**(~ 66%)
- The results can be improved by using other algorithms and GridSearchCV for better data sampling and model training using hyperparameter tuning
- K-NN can also be used for regression problems, but the results are not as good compared to the algorithms covered in regression chapter



	precision	recall	f1-score	support
0	0.62	0.51	0.56	140
1	0.68	0.78	0.73	192

Chapter 9 – Classification - Support Vector Machines (SVM)

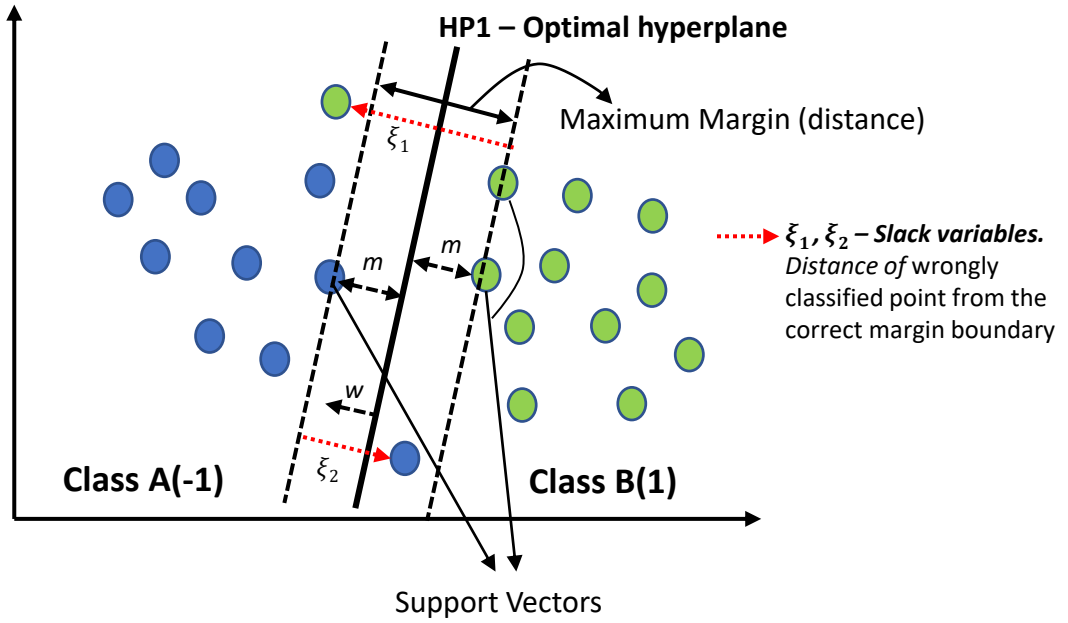
- **Support Vector Machines (SVM)** are used for classification by **building a *hyperplane* that best separates the labelled classes in the training data**
- For labelled training data with n-features (independent variables ($X_{1...n}$)), **each observation is represented as a vector of n-dimensions in the feature space**
  - Each observation is a point with n-dimensions (value of each feature)
- A **new observation** can be classified into the correct class using the hyperplane
  - There can be multiple hyperplanes that separate the classes, but the model selects the best one using different optimization techniques
  - Eg: For binary classification with 2 classes, the hyperplane will be a single line
- The **SVM classifier (SVC)** finds the **best fit (optimal) hyperplane** by calculating the **largest distance (*maximum margin*) between points of the different classes nearest to the hyperplane**
- These points are called **support vectors (margin boundaries),** being most unlike the class they are in
- Larger the margin, farther away are the boundary points from the hyperplane
  - The more these points are like the class they are in and less like the other class
- Consider a binary classification problem with 2 classes (A,B) labelled (1,-1) that can be separated by many hyperplanes (HP1, HP2, HP3, etc)
  - For HP2 and HP3, the support vectors are very close to the hyperplane resulting in a small maximum margin
  - HP1 has the largest maximum margin, resulting in the optimal hyperplane
- A **Linear SVC hyperplane** can be defined as $\mathbf{y_i = f(x_i) = (w^T x_i - b)}$
  - (x_i, y_i) is the set of points of training data. x_i - n-dimensional vector. $y_i \in (1,-1)$
  - **w** – normal vector from the hyperplane to point x_i. **||w||** is its length or magnitude
  - **b** – bias term, so the hyperplane doesn't pass through origin for all hyperplanes
  - $\frac{b}{||w||}$ is the offset of the hyperplane from the origin



Chapter 9 – Classification - Support Vector Machines

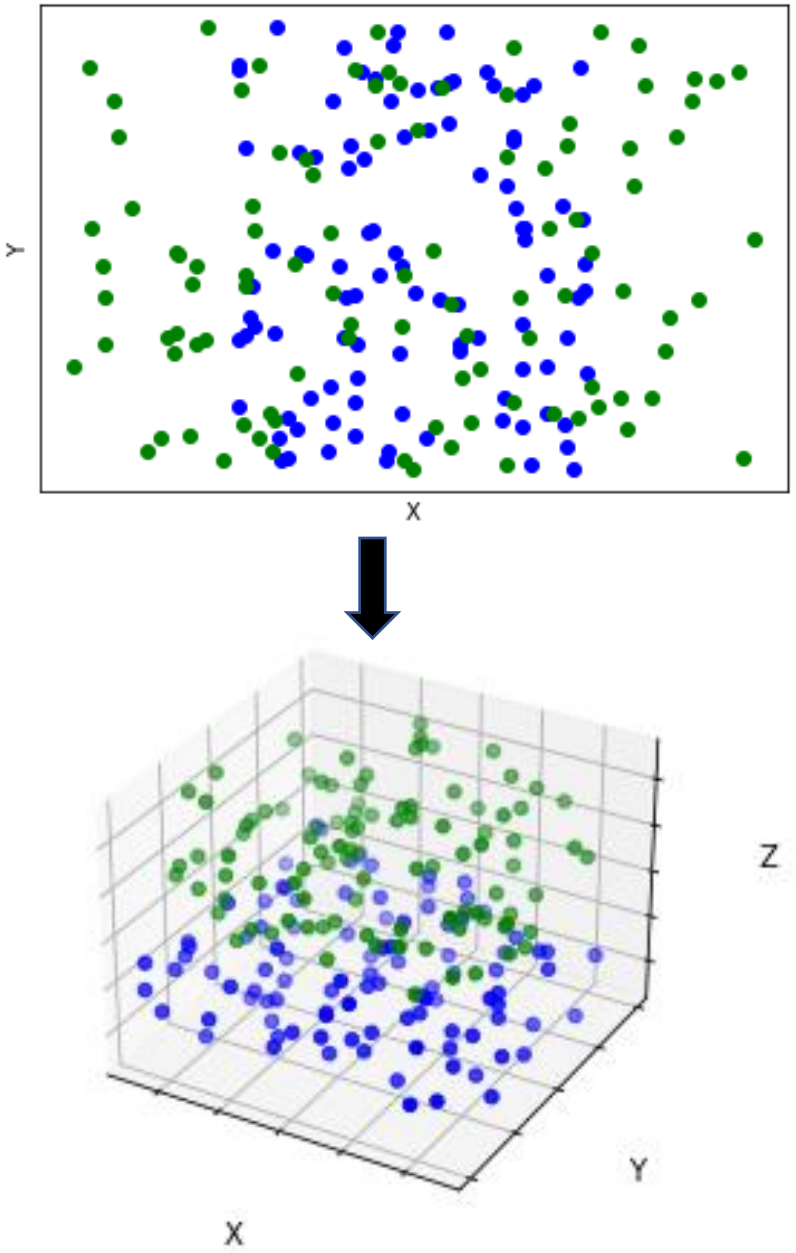
- $w^T x_i - b \geq 1$ for $y_i = 1$ and $w^T x_i - b \leq -1$ for $y_i = -1$
- Which is $\mathbf{y_i(w^T x_i - b) \geq 1}$. y_i being the ***sign function.*** $1 \leq i \leq n$

- Consider 2 parallel hyperplanes, with maximum distance between them, that separate the 2 classes
  - The distance between these 2 hyperplanes is called the margin (2m units)
  - These 2 hyperplanes define the margin boundaries
  - The maximum margin hyperplane is in their middle (m units from both hyperplanes)
  - Geometrically distance between the 2 hyperplanes = $\frac{2}{||w||}$
  - To maximise the margin, we minimize ||w||
- The **linear SVC** becomes an **optimization problem** which **minimizes ||w||** subject to
  - $\mathbf{y_i(w^T x_i - b) \geq 1}$. $1 \leq i \leq n$
  - w, b are determined by solving this equation
- This equation is used if the 2 classes are **linearly separable** (all class points are on the right side of the margin)
- If the 2 classes are **not linearly separable** (some class points are on the wrong side of the margin), the classifier uses slack variables to adjust for these errors
  - **Slack variable (ξ)** - is the distance of a wrongly classified point from the correct margin boundary
- The linear SVC now minimizes **||w||** subject to
  - $\mathbf{y_i(w^T x_i - b) \geq 1 - \xi_i}$. $1 \leq i \leq n$
  - $\xi_i \geq 0, \sum \xi_i \leq$ constant
- The **linear SVC** solves the below **primal problem**
  - $\mathbf{min}_{w, b, \xi}(\frac{1}{2} \mathbf{w^T w + C} \sum_{i=1}^{n} \xi_i)$
  - subject to $\mathbf{y_i(w^T x_i - b) \geq 1 - \xi_i}$. $1 \leq i \leq n$. $\xi_i \geq 0$
  - **C** is a penalty term which controls the impact of the slack variables, replacing the constant



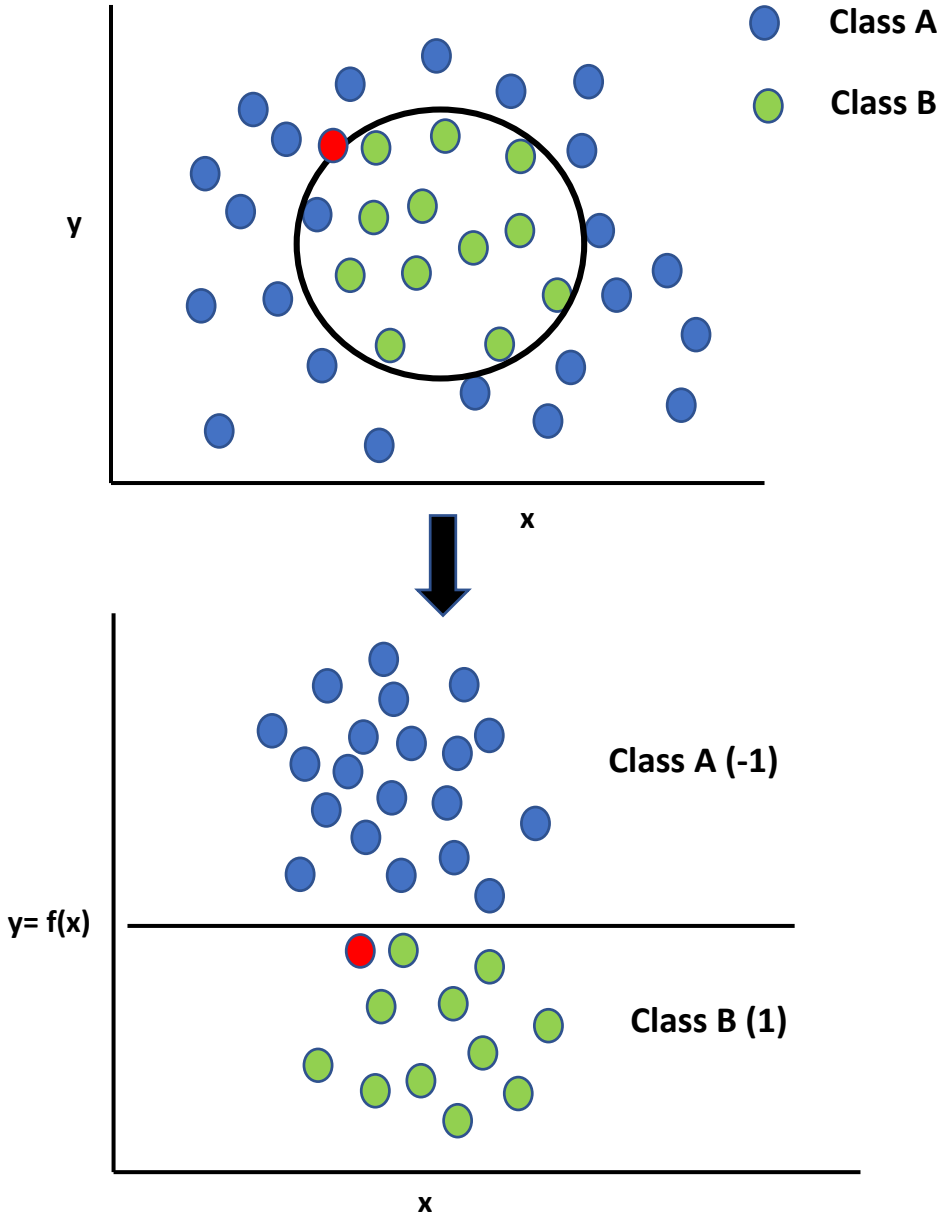
Chapter 9 – Classification - Support Vector Machines

- A **nonlinear SVC** is used when the training data cannot be linearly separated
  - The classifier cannot be expressed as a linear function of independent variables
- It uses a nonlinear function, generally a polynomial equation with degree ≥ 2
  - **Eg: $y_i = f(x_i) = ax_i^3 + bx_i^2 + cx_i + d$**
  - A nonlinear function is one where the output change nonlinearly with the input
- **It uses the 'kernel trick'** or **kernel function** to build the maximum-margin hyperplane
  - A kernel function transforms the data to a higher dimension feature space (Eg: 2D to 3D), to make it linearly separable by a hyperplane
  - Which is then transformed backed to the lower dimension with the linear separating hyperplane
  - Kernel functions can be Polynomial, Radial Basis Function, Gaussian, etc
  - **kernel functions** are generally distance weighted similarity functions (weighted sum, integral) which assign weights to the training data (x_i,y_i)
  - They don't explicitly calculate the dimension coordinates of the vectors (data points) in the feature space
- Instead, **they calculate the inner product between the vectors (x, x') by using a kernel function**
  - x – labeled training data vector. x' – unlabeled input vector (image of x)
  - *$x = [x_1, x_2, \ldots, x_n]$, $x' = [x'_1, x'_2, \ldots, x'_n]$.*
  - *n – no of dimensions of each vector*
  - The inner product converts the 2 vectors into a single number (scalar) **x.x' =** $\sum_{j=1}^{m}(\boldsymbol{xj}, \boldsymbol{x'}_j) = x_1 * x'_1 + x_2 * x'_2 + \ldots + x_m * x'_m$
- Useful website - https://scikit-learn.org/stable/modules/svm.html



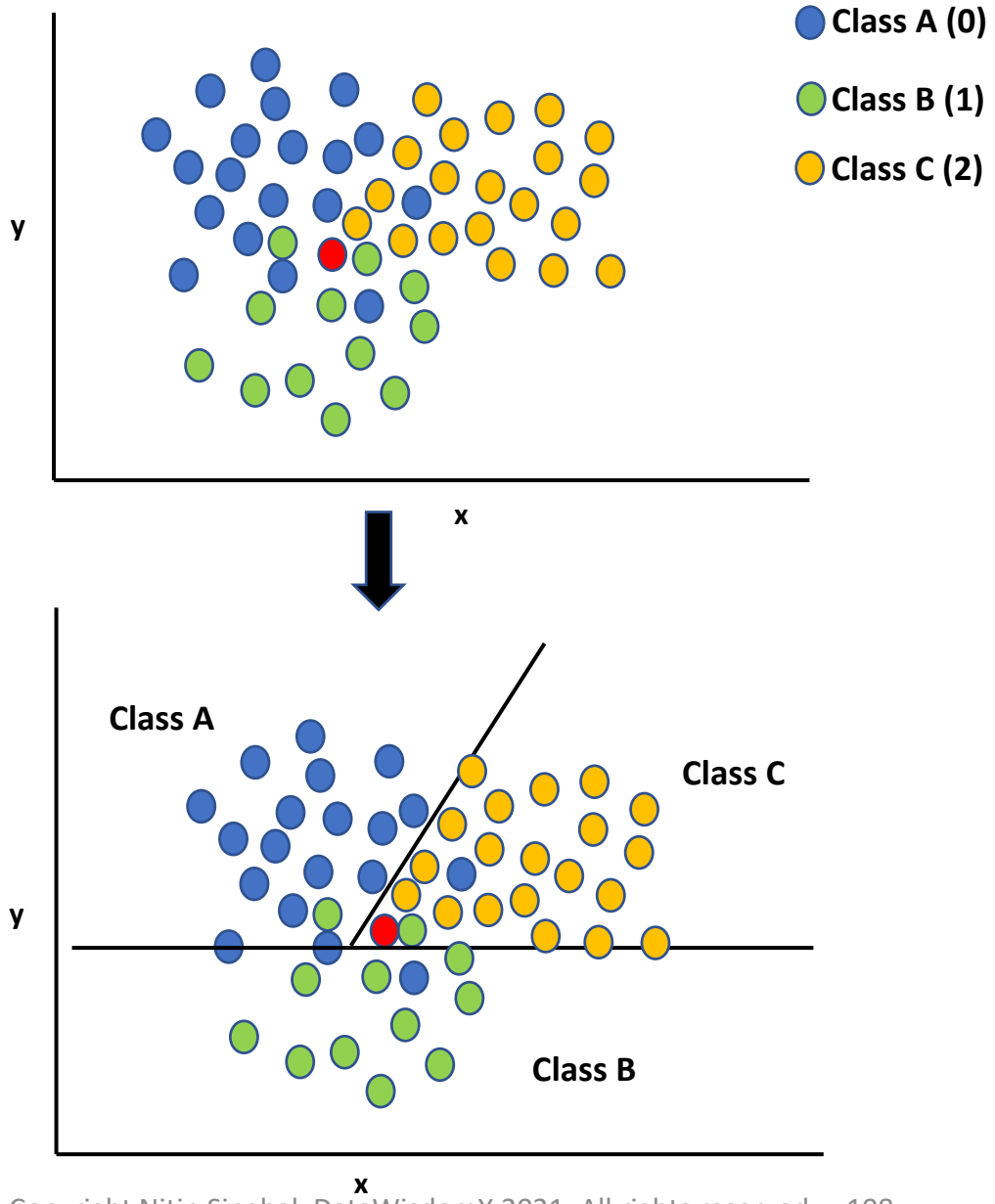
Chapter 9 – Classification - Support Vector Machines

- The **classifier learns from the training data (x_i, y_i) pairs and assigns weights w_i to them**
- It then **applies a similarity function (k) between x' and each x_i, to calculate the weighted sum of similarities to predict $\hat{y}$**
  - $\hat{y}$ **= sign(** $\sum_{i=1}^{n} w_i y_i k(xi, x')$). $\hat{y} \in (1,-1)$
  - $\hat{y}$ – the predicted label for x' (image of x)
- The **nonlinear SVC** solves the **dual problem** below
  - $\min_{\alpha,} (\frac{1}{2} \alpha_i^T Q_{ij} \alpha_i - e^T \alpha)$,
  - Subject to $y^T_i \alpha_i = 0$ and $0 \leq \alpha_i \leq C$.
  - α_i are called the dual coefficients. **C** is the penalty term
  - *i = 1, 2, …n. j = 1,2,…m. m – no of unlabelled inputs*
  - $Q_{ij} = y_i y_j k(x_i, x_j)$, a matrix
  - **$k(x_i, x_j)$ is the kernel function** that transforms the training data into a higher dimension space
- For e.g., The unlabeled data point (red) cannot be correctly labelled by a linear SCV due to the feature space (x,y)
  - The 2D space cannot be linearly separated
- A nonlinear SVC transforms the 2D feature space (x,y) for classes (A,B) into a 3D feature space (x,y,z)
  - The transformed 3D space can be separated by a maximum margin hyperplane
  - This 3D space can then be mapped back to a 2D space separated by a maximum margin hyperplane
- The unlabeled data point (red) can now be correctly assigned to class B



Chapter 9 – Classification - Support Vector Machines

- A **multiclass SVM classifier is used when the training data is to be separated into more than 2 classes (0,1,2,..,n)**
- It transforms the multiclass classification problem into multiple binary classification problems
- It uses different strategies for building the multiclass classifier
  - **'one-vs-one'** – trains one classifier per pair of classes from the training data
    - The class with the most votes gets selected by the combined classifier
  - **'one-vs-rest'** – trains one classifier per class, with the samples of that class labelled as +1 and all other samples as -1
    - For each classifier its class is fitted against all other classes
    - This is the preferred strategy for multi-class classifiers
- Eg: In the given plot of training data with 3 classes (A,B,C or 0,1,2), assigning a class to the unlabeled data point (red dot) is not easy due to similarity to all 3
  - By applying multiclass SVC the training data can be separated into distinct classes by hyperplanes
  - The unlabeled data point (red) can now be correctly labelled with class C (2), being inside its boundary

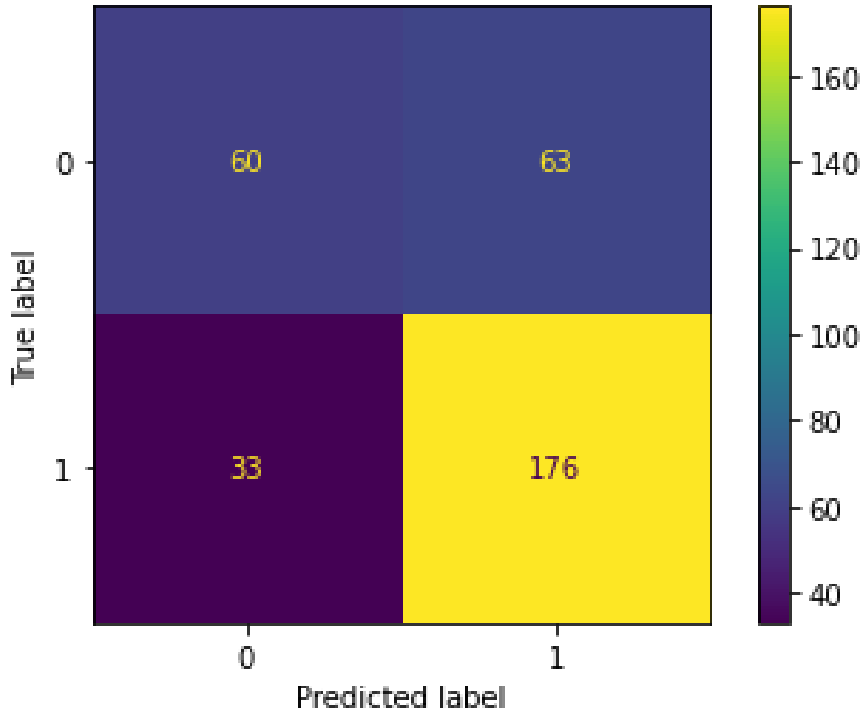


Chapter 9 – Classification - Support Vector Machines

- **sklearn.svm, SVC** library is used for building SVM classifiers (SVC)
  - The **kernel** parameter is set depending on the classification problem being solved – linear or nonlinear
  - Kernel='linear' is generally used for building linear SVC
  - Kernel='rbf', 'gaussian' is generally used for building nonlinear SVC
- For linear problems we can also use Sklearn.svm, LinearSVC library
- Some **parameters** in Sklearn.svm, SVC that can be set for optimization are:
  - **C** – Regularization parameter (penalty term). It is inversely proportional to C.
  - **Kernel** – kernel to be used for the algorithm
    - (linear', 'poly', 'rbf', 'sigmoid', 'precomputed')
    - Default = 'rbf' – radial basis function
  - **Gamma** – kernel coefficient for 'poly', 'rbf', 'sigmoid'
    - ('scale', 'auto')
    - It determines no of features to be used
  - **decision_function_shape** – decision function to be used for multiclass classification
    - ('ovo', 'ovr'). Default = 'ovr' – one-vs-rest
    - Ignored for binary classification
- Python example is given to show the results of binary classification using SVC

Chapter 9 – Classification - Support Vector Machines

- **Sklearn_metrics** - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The confusion matrix (TP,TN,FP,FN numbers) in a grid is a good visual way to evaluate accuracy
  - The accuracy rate and precision are generally good measures
  - The precision results show different values for class labels (0,1)
- The results of the algorithm on the NBA rookie stats to predict if a player will last 5 years in league **are similar to logistic regression** (~ 72%)
- The results can be improved by using other algorithms and GridSearchCV for better data sampling and model training using hyperparameter tuning



	precision	recall	f1-score	support
0	0.64	0.56	0.59	122
1	0.76	0.81	0.79	210

Chapter 9 – Classification – Naïve Bayes

- **Naïve Bayes classifiers use conditional probability models like Bayes theorem**
- Bayes theorem has already been explained in *Chapter 4 – Probability*
- It is the probability of an event occurring depending on the occurrence of prior related events
- For **2 events x and y**, it states that $\mathbf{P(y|x) = \frac{P(x|y)*P(y)}{P(x)}}$, where
  - y - class variable (dependent), x – feature space (independent variables)
- It is also expressed as: $\mathbf{Posterior\ probability = \frac{Likelihood * Prior\ probability}{Evidence}}$
- If the feature space is partitioned into n mutually exclusive samples (x_1, x_2,…, x_n), then for any y, using law of total probability
- $\mathbf{P(y|x_i) = \frac{P(x_i|y)*P(y)}{\sum_{i=1}^{n} P(y|xi)}}$, where x_1,..., x_n is the independent variable feature space
- $\sum_{i=1}^{n} P(y|xi)$ is constant being the input data (evidence)
- So, Bayes theorem becomes: $\mathbf{P(y|x_i) \propto P(x_i|y)*P(y)}$
- **Using joint probability and chain rule**
- $\mathbf{P(x_i|y)*P(y) = P(x_1,..,x_n, y)} = P(x_1|x_2,..,x_n, y)*P(x_2|x_1, x_3,..,x_n, y)*...* P(x_n|x_{n-1}, y)*P(x_n|y)*P(y)$

Chapter 9 – Classification – Naïve Bayes

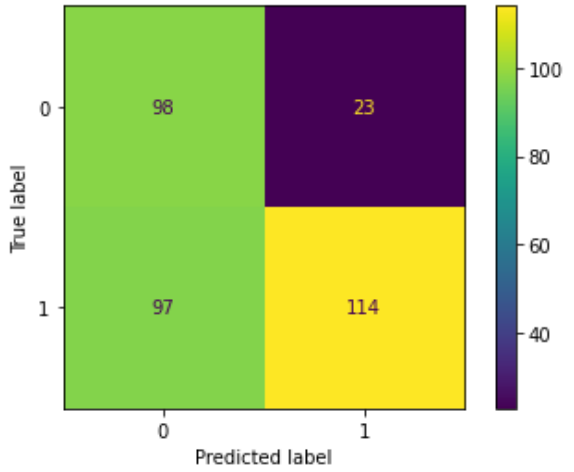
- **Naïve Bayes** makes the **"naïve" assumption of conditional independence (no relationship)** between every pair of features for a given a class variable (label)
  - Normally there is some relationship (correlation) between different features in the training data
- Using the **naïve assumption** of conditional independence between $x_{1\ldots n}$, the equation becomes
- **$P(y|x_i) \propto P(y) * \prod_{i=1}^{n} P(xi|y)$**
- The classifier then uses **Maximum a Posteriori Probability (MAP) estimation** to estimate the class label $\hat{y}$ **for unlabelled data** using the above equation
  - MAP estimates the relative frequency of class y in the training set and selects the most likely value
- **The classifier** solves the equation: **$\hat{y} = \text{argmax}_y P(y) * \prod_{i=1}^{n} P(xi|y)$**
  - **argmax** (**arguments of the maxima**) is the subset of x for which y = f(x) is maximized
- Despite its oversimplified assumption, naïve bayes classifiers work quite well in many real-world applications like document classification and spam filtering
- They require a small amount of training data to estimate the necessary parameters
- They can be extremely fast compared to more sophisticated methods
- Useful website - https://scikit-learn.org/stable/modules/naive_bayes.html

Chapter 9 – Classification - Naive Bayes

- Different types of naïve bayes classifiers are dependent on the type of distribution they assume for **P(xi|y)** – Gaussian, Multinomial, Bernoulli
- **Gaussian Naive Bayes**
  - Used for continuously distributed training data, as a Gaussian distribution, associated with each class
  - It estimates y using a Gaussian function for P(xi|y)
  - sklearn.naive_bayes.GaussianNB library is used
- **Multinomial Naive Bayes**
  - Used for multinomially distributed data, feature count of discrete features, like word vector counts for text classification
  - It estimates y using a relative frequency counting function for P(xi|y)
  - sklearn.naive_bayes.MultinomialNB library is used
- **Bernoulli Naive Bayes**
  - Used for multivariate Bernoulli distributions, multiple binary valued features
  - It uses the occurrence count of binary features and used for short document classification
  - It estimates y using a binary function for P(xi|y)
  - sklearn.naive_bayes.BernoulliNB library is used
- Different problems are solved in the python examples to show the results of different Naïve Bayes classifiers

Chapter 9 – Classification - Naive Bayes

- **Sklearn_metrics** - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The confusion matrix (TP,TN,FP,FN numbers) in a grid is a good visual way to evaluate accuracy
  - The accuracy rate and precision are generally good measures
  - The precision results show different values for class labels (0,1)
- The results of Gaussian Naïve Bayes algorithm on the NBA rookie stats to predict if a player will last 5 years in league **are worse than logistic regression**(~ 64%)
- The results can be improved by using other algorithms and GridSearchCV for better data sampling and model training using hyperparameter tuning



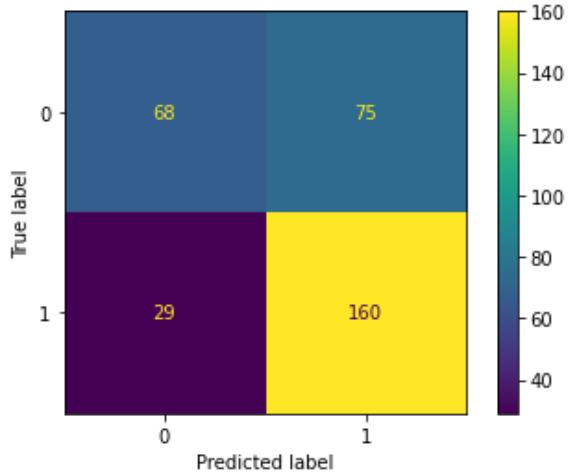
	precision	recall	f1-score	support
0	0.5	0.81	0.62	121
1	0.83	0.54	0.66	211

Chapter 9 – Classification – Decision Trees, Ensembles

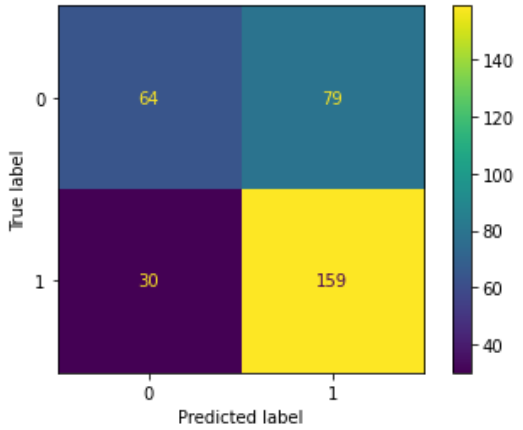
- **Decision trees and Ensemble algorithms are already explained in Chapter 8 - Regression**
- Decision Trees and Ensemble algorithms (Random Forest, XGBoost) are built using the same logic as in regression
- Main difference is that for classification the algorithms generate discrete outputs (class labels)
- **The classifier builds a model Y = f($X_{1...n}$), with output $\hat{y}$ being discrete values (class labels 0,1,2,…,n)**
- **Decision Tree**
  - Use a hierarchical tree-based structure with parent and child nodes connected by branches (conditions)
  - Metrics used at each node to determine the direction to move in the tree (down, left or right Or stop) are
    - Entropy, Information Gain and Gini Impurity
  - **It minimises Information Gain and optimizes Gini Impurity to 1**
  - It has limitations, is an old algorithm and has been replaced with random forest
  - sklearn, tree.DecisionTreesClassifier library is used
- **Ensemble - Random Forest**
  - It is an **ensemble of decision tree estimators that trains multiple estimators in parallel** on subsets of the dataset and then averages their predictions to reduce errors and get the best results
  - $\hat{y} = \frac{1}{m}\sum_{i=1}^{m} \hat{y}_i$,
  - $\hat{y}$ - is the average of all predictions, $\hat{y}_i$ is the prediction of each decision tree *i*
  - sklearn.ensemble, RandomForestClassifier is used
- **Ensemble – XGBoost**
  - Extreme Gradient Boosting uses **gradient boosted decision trees**, adding a new tree sequentially in each iteration to optimize the **Objective function**
  - **Obj(θ) = L(θ) + Ω(θ)**
  - xgboost, XGBClassifier library is used. Learning parameters contain Classification related values
- Different python examples are given to show the results of decision tree and ensemble classifiers

Chapter 9 – Classification - Decision Trees, Ensembles

- **Sklearn_metrics** - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The confusion matrix (TP,TN,FP,FN numbers) in a grid is a good visual way to evaluate accuracy
  - The accuracy rate and precision are generally good measures
  - The precision results show different values for class labels (0,1)
- The results of Decision Tree, Random Forest and XGBoost algorithms on the NBA rookie stats to predict if a player will last 5 years in league **are worse than logistic regression**(~ 68%)
  - Results are shown for random Forest and XGBoost
  - Python examples are given for all 3 models
- The results show that tree and ensemble algorithms don't give good results for classification problems
- The results can be improved by using other algorithms and GridSearchCV for better data sampling and model training using hyperparameter tuning



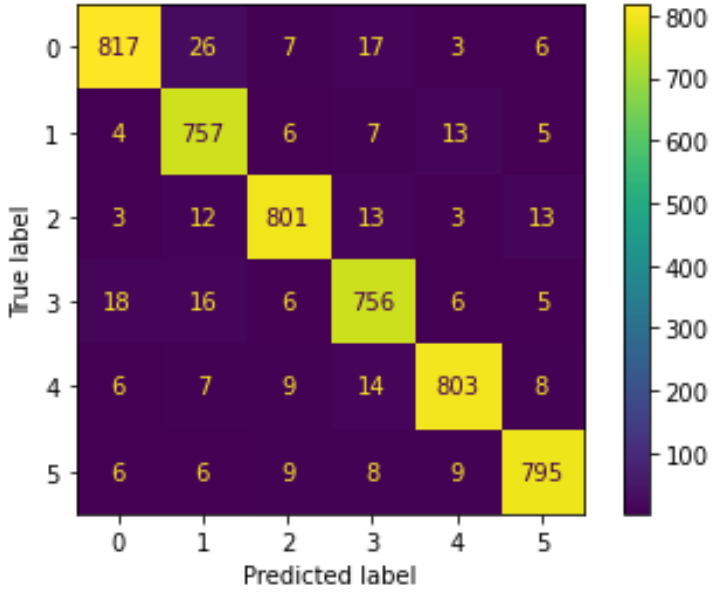
	precision	recall	f1-score	support
0	0.7	0.48	0.57	143
1	0.68	0.85	0.75	189



	precision	recall	f1-score	support
0	0.68	0.45	0.54	143
1	0.67	0.84	0.74	189

Chapter 9 – Multiclass Classification using SVC

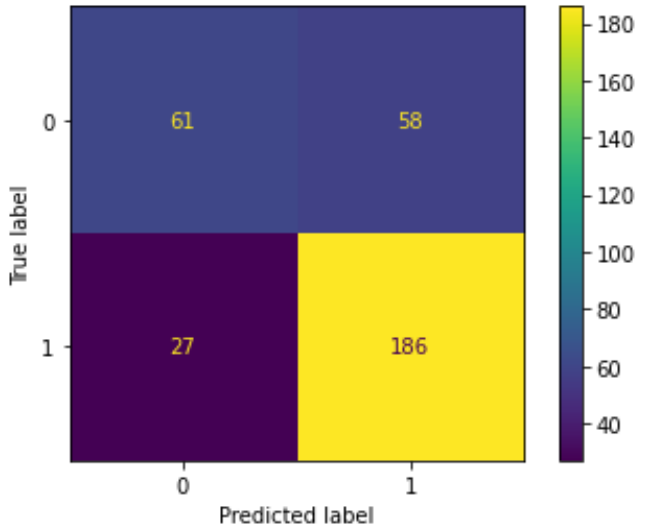
- We are going to solve a **multiclass classification problem using SVC algorithm**
  - You can use other algorithms too – Logistic regression, etc
- We will not discuss the details of the algorithms as they have already been covered in the classification lecture series
- **Main difference** here is that we are now **predicting more than 2 output class labels** from the training data (0,1,2,..,n)
- The model is built using Kaggle Tortuga Code dataset to **predict different students' developer category** depending on their developer profile information
  - The dataset has already been explained in the introduction lecture
- The results are **very good with accuracy of (~95%)**
- **Sklearn_metrics** - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The results show different values for the class labels (different categories) - (0,1,2,3,4,5)
- The results can be improved by using other algorithms and GridSearchCV for better data sampling and model training using hyperparameter tuning
- Useful website - https://scikit-learn.org/stable/modules/multiclass.html



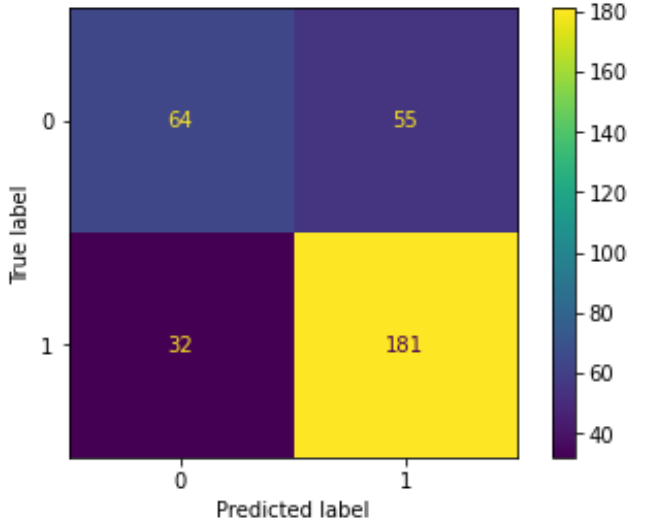
	precision	recall	f1-score	support
advanced_backend	0.96	0.93	0.94	876
advanced_data_science	0.92	0.96	0.94	792
advanced_front_end	0.96	0.95	0.95	845
beginner_backend	0.93	0.94	0.93	807
beginner_data_science	0.96	0.95	0.95	847
beginner_front_end	0.96	0.95	0.95	833

Chapter 9 – Classification Pipeline and GridSearchCV

- The results of applying **GridSearchCV** algorithm on a **Pipeline** of **different classification models** using the NBA rookie stats are shown
- **Pipeline** combines data transformations and training multiple models into a sequence of steps
  - **Grid search** across entire hyperparameter grid to select the best combination for each model
  - **Hyperparameters** to be tuned for each model are setup as a predefined parameter array (grid)
- **GridSearchCV** performs **k-fold cross-validation** to train models on multiple variations of the training data of the sample in an iterative loop (k-folds, k-splits)
- The classification algorithms considered are
  - LogisticRegression
  - SVC
  - GaussianNB
  - KNeighborsClassifier
  - RandomForestClassifier
  - XGBClassifier
- A **10-fold cross-validation** splitting strategy is used to ensure multiple variations of the training data are used (better data sampling)
- Sklearn_metrics - confusion_matrix, accuracy_score and classification_report are used to evaluate and measure the accuracy of the results
  - The results show different values for the class labels (different categories) - (0,1,2,3,4,5)
- The results are **better for all models**
  - The hyper-tuned parameters can now be used for each model
- **Logistic regression and SVC still give the best results** compared to other models
- Useful websites
  - https://scikit-learn.org/stable/modules/cross_validation.html
  - https://scikit-learn.org/stable/modules/grid_search.html
  - https://scikit-learn.org/stable/modules/compose.html



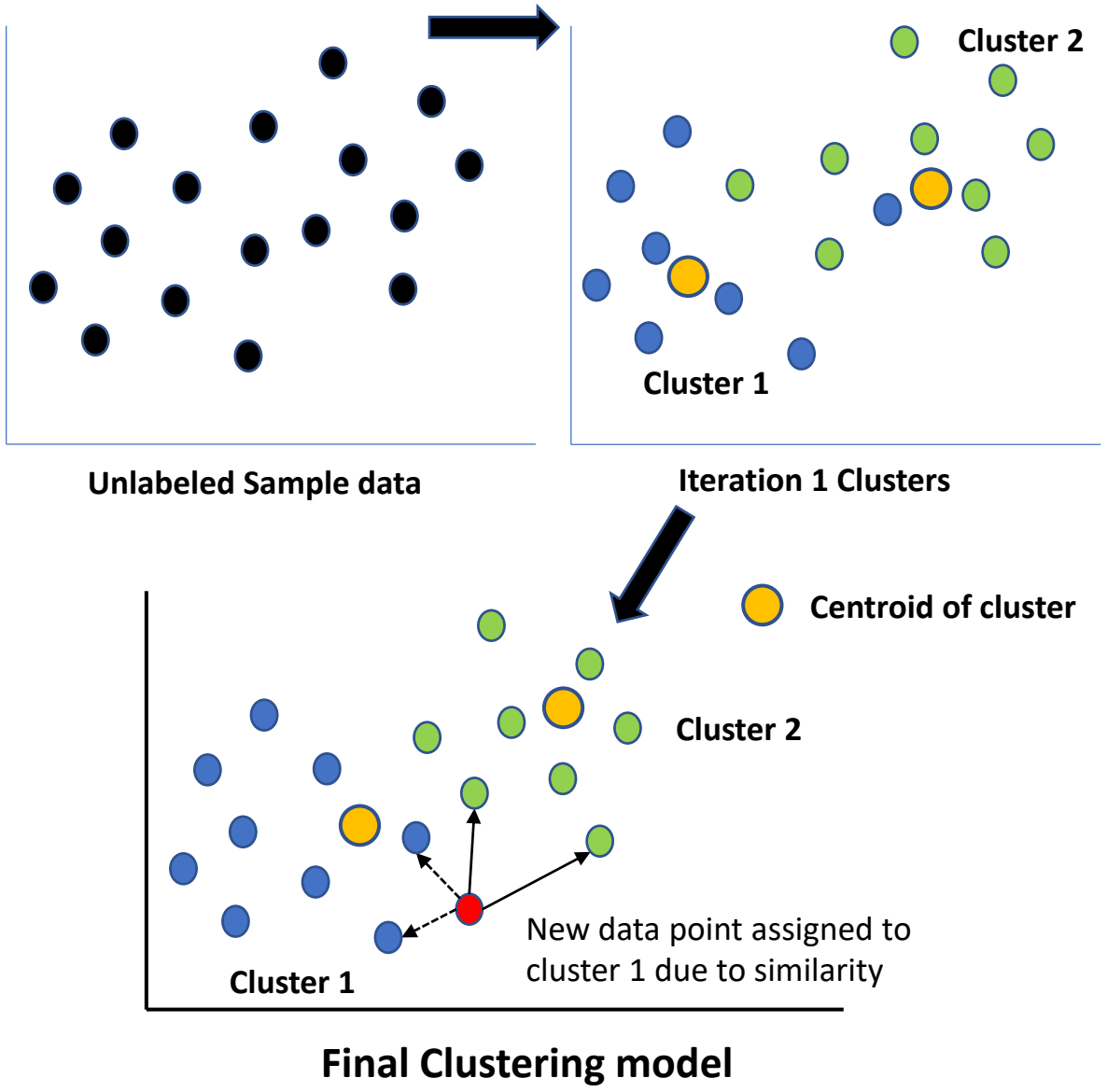
Logistic Regression



SVC

Chapter 10 - Unsupervised Learning Algorithms - Clustering

- **Clustering is a type of unsupervised learning algorithm which uses unlabeled training data X to separate it into y clusters (classes) using some similarity measure (distance, density, etc)**
  - Data points similar (close) to each other are assigned to one cluster
  - It is unlike supervised algorithms that require labeled data to learn from in order to build models to make predictions
- It is a **multi-objective optimization algorithm** that uses an **iterative process** to find the best fit clusters for the training data
- A new unlabeled observation can then be assigned to the right cluster based on the distance metric used
- In the figure shown, the clustering algorithm starts with the unlabeled sample data and performs multiple iterations of
  - Splitting the data into target number of clusters and calculating their centroid
  - In the next iteration it reassigns the data to clusters using the previous iterations centroids and calculates the new centroid, repeating this till an optimal value is reached
  - The splitting, no of iterations, centroid calculation, etc depend on the algorithm used and its defined parameters
  - No of clusters are generally user defined or estimated by the algorithm



Chapter 10 - Unsupervised Learning Algorithms - Clustering

- **Some Metrics used**:
  - **Silhouette Coefficient** – defines how well defined and dense the clusters are. Useful when actual y values are unknown. Between (-1,1). Closer to 1 better
  - **Adjusted Rand index** - proportional to the number of sample pairs with same labels for predicted and actual y labels. Between (0,1). Closer to 1 better
  - **Homogeneity -** each cluster contains only members of a single class. Between (0,1). Closer to 1 better
  - **Completeness -** all members of a given class are assigned to the same cluster. Between (0,1). Closer to 1 better
- Different clustering algorithms depend on the type of distance metric used. Some common one's are discussed
- Useful website - https://scikit-learn.org/stable/modules/clustering.html

Chapter 10 - A comparison of the clustering algorithms in scikit-learn

Method name	Parameters	Scalability	Usecase	Geometry (metric used)
K-Means	number of clusters	Very large n_samples, medium n_clusters with MiniBatch code	General-purpose, even cluster size, flat geometry, not too many clusters	Distances between points
Affinity propagation	damping, sample preference	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Mean-shift	bandwidth	Not scalable with n_samples	Many clusters, uneven cluster size, non-flat geometry	Distances between points
Spectral clustering	number of clusters	Medium n_samples, small n_clusters	Few clusters, even cluster size, non-flat geometry	Graph distance (e.g. nearest-neighbor graph)
Ward hierarchical clustering	number of clusters or distance threshold	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints	Distances between points
Agglomerative clustering	number of clusters or distance threshold, linkage type, distance	Large n_samples and n_clusters	Many clusters, possibly connectivity constraints, non Euclidean distances	Any pairwise distance
DBSCAN	neighborhood size	Very large n_samples, medium n_clusters	Non-flat geometry, uneven cluster sizes	Distances between nearest points
OPTICS	minimum cluster membership	Very large n_samples, large n_clusters	Non-flat geometry, uneven cluster sizes, variable cluster density	Distances between points
Gaussian mixtures	many	Not scalable	Flat geometry, good for density estimation	Mahalanobis distances to centers
Birch	branching factor, threshold, optional global clusterer.	Large n_clusters and n_samples	Large dataset, outlier removal, data reduction.	Euclidean distance between points

Source – Scikit-learn Clustering online guide

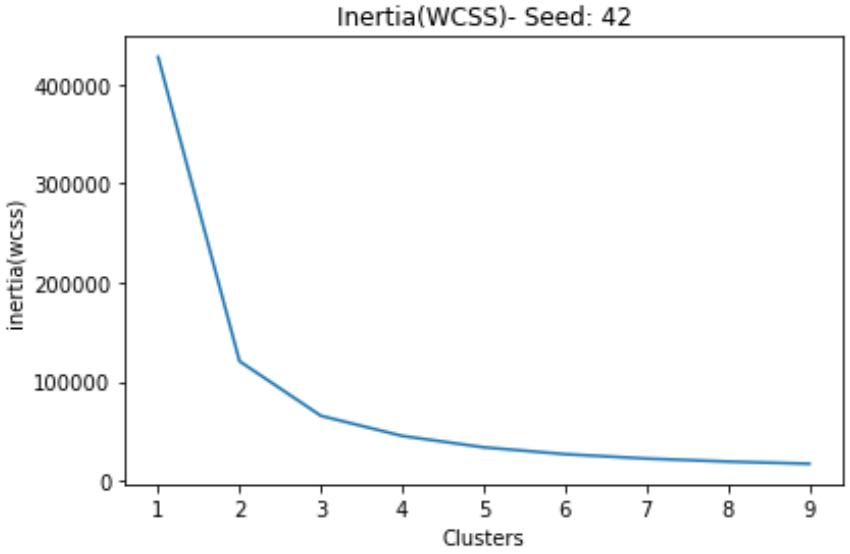
Chapter 10 - Clustering Algorithms - K-Means

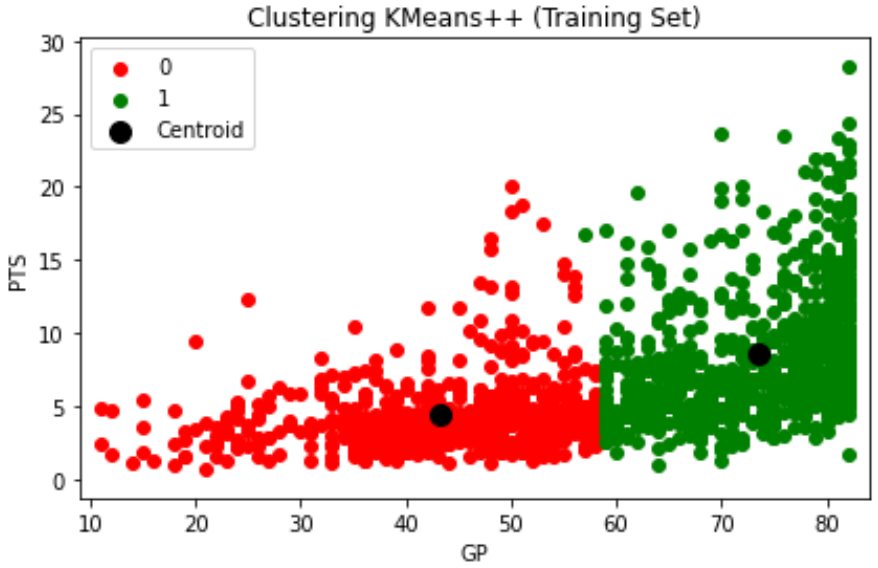
Centroid based K-Means clustering algorithm separates the n samples of X into **k clusters by minimizing the squared distance of the data point from the cluster mean C (centroid)**

- Centroids are selected so that the **within-cluster-sum-of-squares WCSS** (inertia) is minimized
- The equation to be solved is $\sum_{i=0}^{n} \boldsymbol{min}(||\boldsymbol{xi} - \boldsymbol{\mu_j}||^2)$. $\mu_j \in$ C, is the mean of each cluster (j = 1 to k)
- The algorithm works by first randomly initializing ***k***, then repeats the below steps till the threshold is reached
  - Assigns each sample x_i to the nearest centroid μ_j
  - It then creates new centroids by taking the mean of samples assigned to each previous centroid
  - Difference between the mean of the old and new centroids is compared to the threshold
  - These 3 steps are repeated till the threshold is reached
- As ***k*** is randomly fixed at the start, it is a drawback as the algorithm has to be repeated for different values
- To over come this random initialization issue the **Elbow method** and **K-Means++** initialization are used
- The **Elbow method** gives a better initial value for k, by plotting the explained variance as a function of the no of clusters
  - The 'elbow' or 'knee' of the curve acts as the cut-off point for k for diminishing returns
  - The point where adding more clusters does not improve the model, cost vs benefit)
- **K-means++ initialization** further improves the results by initializing the centroids to be distant from each other
- K means algorithms assume that the clusters are convex shaped (outward curve, all parts pointing inwards. Eg: rugby ball)
- **Useful website - https://scikit-learn.org/stable/modules/clustering.html#k-means**

Chapter 10 - Clustering Algorithms - K-Means

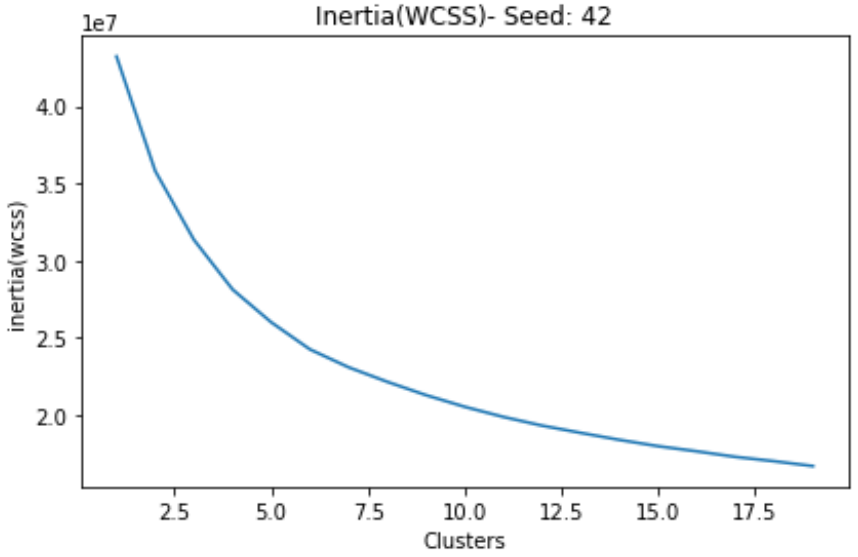
- **sklearn.cluster, KMeans** library is used for building the clusters. Some useful parameters are
  - n_clusters = no of clusters to be generated. Default = 8
  - init = no of initial clusters ('kmeans++', 'random')
  - max_iter = maximum no of iteration for a single run of the algorithm. Default = 300
- The code for plotting the elbow curve to determine initial ***k*** is given in the example
  - It is determined by using the **inertia_** attribute of the model
  - It is the Sum of squared distances of samples to their closest cluster centre (**WCSS**)
- Results **NBA rookie stats** to predict if a player will last 5 years in league as 2 clusters are given below
- https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans

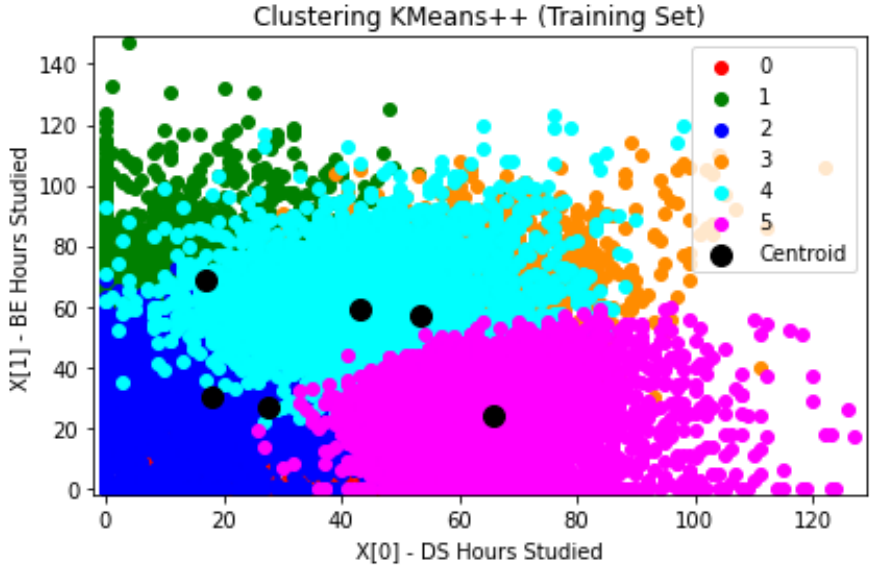


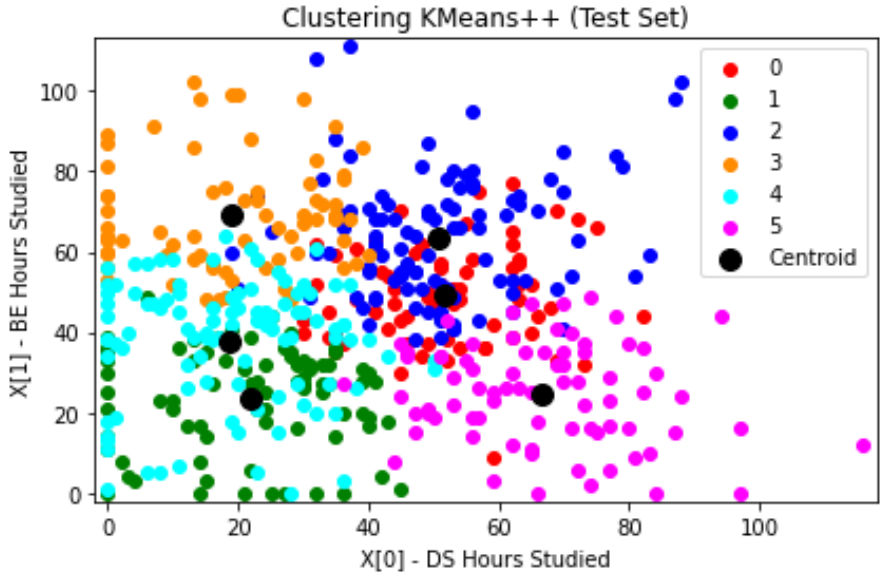


Chapter 10 - Clustering Algorithms - K-Means

- Results for Kaggle Tortuga Code dataset to **predict different students' developer category** as 6 clusters are given below







- ***NOTE*** *– The* ***clustering plots are using only 2 of the X input variables,*** *even though* ***all X variables are used for training the model***
- ***This is just to show you*** *what the plot of multiple clusters looks like*
- ***Accurate plotting*** *requires dimensionality reduction to 2D and then plotting the clusters*

Chapter 10 - Clustering Algorithms - Hierarchical

Connectivity based Hierarchical clustering algorithms build nested clusters by merging or splitting them successively by minimizing a linkage (distance) criteria

- Observations closer to each other (distance) can be connected to form a cluster

- They can be agglomerative or divisive
  - **Agglomerative** algorithms use a bottom up approach starting with each sample observation X as a cluster and aggregates (merges) them successively into clusters
  - **Divisive** use a top down approach starting with the entire sample as a cluster and successively dividing it into clusters
- It is represented as a **hierarchical tree (dendrogram)**
  - Root is cluster of all samples and leaves are clusters of individual samples
- Different linkage criteria's are:
  - **Single** – minimizes distance between **closest observations** of pairs of clusters
  - **Average** - minimizes average distance between **all observations** of pairs of clusters
  - **Complete** (Maximum) - minimizes the maximum distance between **observations** of pairs of clusters
  - **Ward** – minimizes the sum of squared differences within all clusters. Minimizes variance. Similar to k-means
- Single linkage while being the worst hierarchical strategy, works well for large datasets and can be computed efficiently
- **sklearn.cluster.AgglomerativeClustering** library is used for building hierarchical clusters. Some useful parameters are:
  - n_clusters = no of clusters to be generated. Default = 2
  - affinity = metric to compute linkage ('Euclidean', 'l1', 'l2', 'Manhattan', 'cosine', 'precomputed')
  - linkage = which linkage criteria to use ('ward', 'complete', 'average', 'single'). Default = 'ward'
  - distance_threshold = linkage distance threshold above which clusters will not be merged
- **Useful websites** -
  - https://scikit-learn.org/stable/modules/generated/sklearn.cluster.AgglomerativeClustering.html#sklearn.cluster.AgglomerativeClustering
  - https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html#sphx-glr-auto-examples-cluster-plot-agglomerative-dendrogram-py

Chapter 10 - Clustering Algorithms – DBSCAN

Density based clustering algorithms define clusters as areas of high density separated by areas of low density

- A cluster is a set of **core samples** close to each other (areas of high density) and a set of **non-core samples** close to a core sample (neighbors) but are not core samples themselves
- The closeness is calculated using a distance metric

- **DBSCAN** (Density-Based Spatial Clustering of Applications with Noise) is a popular density based algorithm
  - It builds clusters using 2 parameters that define density
  - **min_samples** – minimum no of samples required in a neighborhood for a core sample
  - **eps** – maximum distance between two samples to be considered as each others' neighbors
  - Higher min_samples or lower eps indicates higher density to form a cluster
- Density based clusters can be of any shape (unlike k-means convex shape)
- **sklearn.cluster.DBSCAN** library is used for building the clusters. Some useful parameters are:
  - eps – default = 0.5
  - min_samples – default = 5
  - metric - metric used for calculating distance between samples
  - algorithm **-** NearestNeighbors algorithm to be used ('auto', 'ball_tree', 'kd_tree', 'brute')
- Useful websites -
  - https://scikit-learn.org/stable/modules/clustering.html#dbscan
  - https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

Chapter 10 - Clustering Algorithms - Gaussian Mixture Model

Distribution based clustering algorithms build clusters with samples belonging to the same distribution

- **Gaussian Mixture Model (GMM)** is a popular method within them
- GMM is a probabilistic model that assumes that all data points are generated from a mixture of finite Gaussian distributions (normal distribution) with unknown parameters
- It implements the **Expectation-Maximization algorithm**, which finds the **maximum-likelihood estimates** of unknown parameters of the model which depends on latent variables (inferred from observed samples)
- They suffer from limitations like scalability

- **sklearn.mixture.GaussianMixture** library is used for building the models
- **Useful websites –**
  - https://scikit-learn.org/stable/modules/mixture.html
  - https://scikit-learn.org/stable/modules/generated/sklearn.mixture.GaussianMixture.html

- **The results of the different algorithms show that KMeans gives the best results**
  - It is more widely used and has better interpretability
- Plotting in higher dimensions (above 3D) is difficult so dimension reduction techniques like LDA and PCA can be used
  - *This is covered in part 2 of the course*

Chapter 11 – Model Persistence and Deployment

- **Once the final trained model is ready, we want to use it for predictions using new data**
  - We don't want to to run the python file which trains, tests and validates the model
  - Instead, we want to call the model file with the new data to get the predicted values
- To do this we need to **persist the model to a file and deploy it on a webserver/cloud platform** where it is integrated into the live environment of the overall analytics solution
  - Generally, a repository (directory, endpoint) in a cloud hosted storage (AWS S3 bucket) or a local machine
  - Applications can make API calls to the model to return the results for new data
- We'll discuss **different approaches** for this
  - Local machine persistence and deployment with webserver (**joblib and flask**)
  - Persistence and Deployment on Cloud storage **(AWS S3 (Simple Storage Service))**
  - Persistence and Deployment on webserver in Cloud application environment (**AWS EB (Elastic Beanstalk)**)
  - Using Cloud ML platform to train, build, deploy and call model (**AWS Sagemaker**)
- **Data scientists are expected to know this** as it is part of the complete data science skill set
  - See the data science process lecture for this

Chapter 11 – Model Persistence and Deployment

- We **Persist** the model by using a serializer/de-serializer library like **joblib** to convert the python file into a byte stream (binary file) and vice versa
  - Serializing/Deserializing allows the model file to be loaded and used on any platform
  - **Joblib** – is an efficient python pipelining tool for parallel computing and persistence of random python objects containing large datasets
  - It is a replacement for python's pickle library
  - For persistence use its dump and load functions to save and load the model file for prediction on local machine
  - https://joblib.readthedocs.io/en/latest/persistence.html
- Then **Deploy** the model file on any webserver on a cloud platform using a WSGI framework like **Flask**
  - **Flask** is a python WSGI (web server gateway interface) library, that allows sending of REST API messages between the webserver and web application using HTTP Request protocol (GET, POST)
  - It enables building REST API's which can be used for prediction using the model file, sending/receiving JSON data
  - REST API protocols and JSON are used for their simplicity and standardization
  - **The model file and web app can be deployed (hosted) on a webserver like Apache on any cloud platform like AWS**
  - **Deployed model Call** - Data is sent by the web app to the model which returns the predicted value as JSON using REST API
  - https://flask.palletsprojects.com/en/1.1.x/quickstart/#a-minimal-application
  - https://flask.palletsprojects.com/en/1.1.x/deploying/#deployment
- The model can be retrained using new data and redeployed as a new model file by repeating the above process
- **Python examples** are given to show how to persist, deploy and call a model file for prediction on
  - Local machine
  - Local webserver using flask
- **Webserver configuration**
  - The **flask's built-in development webserver** will run on localhost - http://127.0.0.1:5000/
    - But it is not suitable for production. Use Apache or any other production server
  - Mac OS comes pre-installed **with Apache webserver** which can also be used
    - Make changes to the **httpd config file** if it is not up and running
  - There are lot of articles/guides available online for setting up localhost on Mac OS
- Useful website - https://scikit-learn.org/stable/modules/model_persistence.html

Chapter 12 – Deploying on AWS Cloud S3 using Boto3

- **Create a free AWS account to use this feature**
  - Use the free tiers to run the sample course code
  - ***Make sure you DELETE any RESOURCES (S3 buckets) CREATED on AWS after completing the course or YOU CAN GET BILLED!***
  - https://aws.amazon.com/free/
- **Boto3 is the AWS SDK for python** making it easy to create, configure and manage services like S3 (Simple Storage Service) on AWS Cloud
  - https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
- It provides an API that allows creating, reading and writing to files on S3
- **S3 is AWS's cloud storage** that stores data as objects within buckets
  - https://docs.aws.amazon.com/AmazonS3/latest/gsg/GetStartedWithS3.html
- Build and train the model on your local machine and deploy on AWS Cloud by storing as files on S3
  - All data files can be stored on S3
- There is no need for running a webserver (flask, etc)
- Users (or applications) can download the model and data files on their local machines (or application instance) using Boto3 API
- This makes it easy to experiment with different algorithms without incurring cloud computing charges
- Also, users can use any development environment (eg: Anaconda) and IDE (eg: Spyder) they prefer

Chapter 12 – Deploying on AWS Cloud S3 using Boto3

- Example is given to show how to setup and use it for deploying a model
  - https://boto3.amazonaws.com/v1/documentation/api/latest/guide/quickstart.html
  - **Install the latest Boto3 version** in your python environment
    - Use *$ conda install boto3* for anaconda
    - Make sure that the correct python version is installed for boto3
  - **Setup your AWS IAM authentication credentials** (see AWS IAM section) using python commands
    - https://docs.aws.amazon.com/IAM/latest/UserGuide/introduction.html
    - Create the user and group using the AWS IAM console
    - Make sure **AWS CLI SDK is installed** on your machine to use AWS services using command line (terminal)
      - https://docs.aws.amazon.com/cli/latest/userguide/install-cliv2-mac.html
      - *$ aws configure* to setup the credentials
      - Then Set aws_access_key_id and aws_secret_access_key
  - **Create an S3 bucket** in the AWS console - https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html
- **Once AWS environment setup is complete use boto3 API calls to read and write files on S3**

Chapter 12 – Deploying on AWS Cloud using Elastic Beanstalk

- **AWS Elastic Beanstalk (EB)** is ideal for cloud-based web applications as it makes it easy to persist, deploy and manage applications on AWS Cloud
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/concepts-webserver.html
- **No need for cloud computing knowledge** and infrastructure management like load balancing, scaling, development applications, etc
  - EB sets it up for you by default including **AWS EC2 instance** (Elastic Cloud Compute, for cloud computing resources. Micro by default)
- First **setup** Elastic Beanstalk on AWS
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/GettingStarted.html
- As **EB supports python**, we can use all related libraries like scikit-learn, pandas, flask, Apache server, databases, etc
- You can build and train the model on your local machine and then deploy on a webserver on AWS Cloud using **Elastic Beanstalk and flask**
- This makes it easy to experiment with different algorithms directly on the cloud, without incurring cloud compute costs
- Also, users can use any development environment (eg: Anaconda) and IDE (eg: Spyder) they prefer
- **Example** is given to show how to setup and use EB for deploying a model using python and flask
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/create-deploy-python-flask.html
- It requires a setup process to be followed using the **EB CLI (Command Line Interface)**, given in the next slides
  - Specific configuration settings are also required to launch and use EB
- See the **EBInstallDeploy_CLICommandsList.txt** file in the lecture folder for the step-by-step list of commands to execute for installing EBCLI and using it to deploy the model

Chapter 12 – Deploying on Elastic Beanstalk – Setup & Settings

- **Use EB CLI** which is a command line interface for AWS Elastic Beanstalk (EB)
  - It provides interactive commands that simplify creating, updating and monitoring environments from a local repository.
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3.html
- **Install the EB CLI** using the instructions at - https://github.com/aws/aws-elastic-beanstalk-cli-setup
  - ***Don't Install EB CLI with homebrew, it gives errors***
  - Check the version - *$ eb --version*
- **Configure the EB CLI** - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-configuration.html
  - Create a **project folder ebdeploy**
  - **Initialize it** using *$ eb init*
  - No need to setup Environment variables – AWS_ACCESS_KEY_ID and AWS_SECRET_ACCESS_KEY
  - As credentials file is already set during aws sdk setup under ~/.aws folder
- To **commit the application** and related dependencies
  - **Initialize** git - *$ git init*
  - Then install all related package dependencies and application files (.py) in the **folder ebdeploy**
    - All packages to be installed are in the **requirements.txt** file in the lecture folder
  - Make sure to **add Procfile** to your source bundle to configure **uWSGI** as the default WSGI server for your application
    - It should contain "web: uwsgi --http :8000 --wsgi-file application.py --master --processes 4 --threads 2"
  - Create the **requirements.txt file** using *$ pip list --format=freeze > requirements.txt*
  - Then **save** all project files using *$ git add .*
  - Then **commit** all changes *$ git commit -m "ebdsmodeldeploy application"*

Chapter 12 – Deploying on Elastic Beanstalk – Setup & Settings

- Then **Create a new environment in EB** using *$ eb create*
  - You can see the environment in the EB dashboard or by *$ eb status*
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb-cli3-getting-started.html#ebcli3-basics-create
  - *Create only if it does not exist, else not required*
- Now **Deploy the application** from your project folder ebdeploy using *$ eb deploy*
- Then **open the application web page** with its unique AWS URL using *$ eb open*
  - If all configurations are correct application web page will launch
  - Otherwise see **troubleshoot common errors** below
- **DELETE all RESOURCES created** (S3 Buckets, EB Environments, EC2 instance) after completing the lecture or **you will get BILLED $$'s DAILY!**

Troubleshoot common errors

- **Environment settings** - check the configuration *$ eb config* or *Health and Logs* in the dashboard
- **We will use uWSGI** as the default WSGI server
  - *Note* - Elastic Beanstalk provides Gunicorn as the default WSGI server, but gives random errors during setup
  - Flask's built-in server is not suitable for production as it doesn't scale well
  - You can deploy your Flask application to production using various WSGI servers
- Make sure to **add Procfile** to your source bundle to configure the WSGI server for your application.
  - We need this to **set default WSGI server as uWSGI** and the **configuration settings to use PORT 8000**
  - Otherwise it gives an ERROR 111: Connection refused) while connecting to upstream, client: 172.31.13.8, server: , request: "GET /.well-known/security.txt HTTP/1.1", upstream: "http://127.0.0.1:8000/

Chapter 12 – Deploying on Elastic Beanstalk – Setup & Settings

- **eb create – environment creation ERRORS**
  - If **any packages** like anaconda or conda **give ERRORS due to versions, DELETE them** from the requirements.txt file
  - Pip has older versions of anaconda, while using conda commands installs the latest ananconda versions
  - Check other unwanted packages (clyent, mkl-fft, etc) that give ERRORS and DELETE them too
  - **Best way is to remove .**ebcli-virtual-env folder. Also remove .git and ebdeploy folders
    - Then do a **clean install of ebcli**
- You **must use Git or CodeCommit** with EB otherwise deployment will fail
  - We are using Git, not CodeCommit
  - https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/eb3-cli-git.html
- **Useful Git commands** if getting errors during eb deploy
  - ***eb deploy --staged*** (this performs standard deployment, bypassing Codecommit)
  - (ERROR: CommandError - An error occurred while handling git command.)
  - Error code: 128 Error: fatal: unable to access 'https://git-codecommit.ap-south-1.amazonaws.com/v1/repos
  - The requested URL returned error: 403
- **Nginx reverse proxy**
  - EB uses nginx as the reverse proxy to map your application to the Elastic Load Balancing load balancer on port 80
  - By default, Elastic Beanstalk configures the nginx proxy to forward requests to your application on port 5000
  - A reverse proxy server is a type of proxy server that sits behind the firewall in a private network and directs client requests to the appropriate backend server

Chapter 13 – Building, Deploying and Calling on AWS Sagemaker – Cloud Machine Learning platform

- **AWS SageMaker** is a fully managed cloud machine learning platform that makes it easy to build, train, deploy and call ML models at scale
  - https://aws.amazon.com/sagemaker/
  - It provides a fully integrated web IDE and workflow for the entire data science process along with Cloud management
  - It makes it easy to manage security, access databases and deploy the model to applications
  - Supports the latest ML frameworks and toolkits – Scikit-Learn, TensorFlow, PyTorch, etc
- It has a **very comprehensive set of features** - https://aws.amazon.com/sagemaker/features/
- Tutorials and developer guide are available at - https://docs.aws.amazon.com/sagemaker/latest/dg/gs.html
- **AWS tutorial** example is given to show how to setup and use Sagemaker for deploying a model
  - https://aws.amazon.com/getting-started/hands-on/build-train-deploy-machine-learning-model-sagemaker/
- You can try the different examples in the normal free tier limits or incur minimal charges for small model/datasets
  - https://aws.amazon.com/sagemaker/pricing/
- **It is a very good and flexible ML platform. But it requires an initial learning curve and has some limitations**
  - To adapt your code to work with its format and process which can be time consuming
  - Also, your ML code is tied to it, which means porting to other cloud platforms can be difficult

Chapter 13 – Building, Deploying and Calling on AWS Sagemaker – Cloud Machine Learning platform

- **Note – To avoid UNNECESSARY BILLING,** use AWS Management Console to **DELETE ALL resources** that you created for any example - endpoints, s3 buckets, notebook instance, models, etc.
- ***OTHERWISE,*** *you get* ***BILLED $$'s DAILY !***
- This scikit-learn random forest example given for Sagemaker makes the model building and deployment ***unnecessarily complex*** and has ***limitations in training/hyperparameter tuning***
  - https://github.com/aws/amazon-sagemaker-examples/blob/master/sagemaker-python-sdk/scikit_learn_randomforest/Sklearn_on_SageMaker_end2end.ipynb
- There are web UI and automation options like **Sagemaker Studio (web IDE)** and **Sagemaker Autopilot (AutoML)**
  - They can be used to visually setup and deploy models and automate some data science process steps
  - But they require a lot of learning of their features and are not flexible like building on your development IDE (Eg: Anaconda, Spyder, etc)
  - They are still new and have limited help guides, so if you get stuck it is tough to find solutions in online forums/blogs, etc
  - https://docs.aws.amazon.com/sagemaker/latest/dg/autopilot-automate-model-development.html
- **Overall, it is better to learn how to build the models as per the course and then switch to AWS Sagemaker tools as required**

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

Objective – Give students ideas on how to build a RoboAdvisor algo – ***(NOT for LIVE MARKET INVESTMENT)***

- For a multi-asset portfolio using forward predicted asset prices for different timeframes
- **Create a medium to long-term portfolio for small/medium investors**
- **Predict 3 year forward monthly average asset prices and direction** (stock indices, commodities, bonds, currencies, real estate) using historic macroeconomic and central bank forecasts for US, UK and EU. Only the 5 main macro factors were used
- **Predict forward Return and Risk for a multi-asset portfolio** (stock indices, commodities, currencies, real estate, bonds) using pre-defined asset weights and predicted forward monthly asset prices for different time frames – 3 months to 3 years
- **Backtest** the results by comparing predicted historic asset prices against actual for historic actual macro data
- **Project has been split** into 2 parts. Part 1 is given
- ***The model can be deployed*** *in the cloud using any of the techniques given in the course*
  - **Sample python code, data and results** are given in this course
- ***Note*** *- Part 2 related to portfolio optimization will be given later in the next course*

Standard Data science process was used

- **Data loading** from files
- **EDA/Data Wrangling** – Cleaning, formatting, deleting, merging, selecting, etc
- **Model selection and predicting** (**GridSearchCV** was used for algorithm selection and hyperparameter tuning)
- **Analysis and presenting** results using charts
- **Deployment** can be done using **AWS Elastic Beanstalk and flask –** not part of the tutorial

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

Code Structure uses a collection of modules and methods - *Chapter14_E2EProjectRoboadvisor.py*

- **Loading and Wrangling** the data - *Chapter14_RoboadvisorDataLoadWrangling.py*
- **Predicting** asset and bond prices using random forest algorithm - *Chapter14_RoboadvisorAssetBondPricePrediction.py*
- **Asset returns** are calculated for monthly/annual timeframe **-** Net Rate of Return (NRR), Average, Annualized - *Chapter14_RoboadvisorAssetReturnsCalculator.py*
- **Bond returns** are calculated separately due to yield and price calculation - *Chapter14_RoboadvisorBondReturnsCalculator.py*
- **Predicting directional movement** (Long/Short) - *Chapter14_RoboadvisorAssetBondPredVsActDirectionCalculator.py*
- **Portfolio Return & Risk** using portfolio weights for different timeframes and asset types - *Chapter14_RoboadvisorPortfolioReturnRiskCalculator.py*
- **Backtesting** results by comparing predictions using actual and forecasted data

Data Used

- **US, UK and EU** monthly main macroeconomic data for last 21 years (Jan 99 – Dec 19) was used as independent variables (X)
- Average monthly close price over the same period was used for assets and bonds as the dependent variable (Y)
- All 3 countries' macroeconomic data was combined to see their impact on each asset price
- ***Note -*** *Data needs to be updated to latest 2021 and models rebuilt to get latest results*

Algorithm Logic - Forward Asset Price Prediction

- For 3 year forward price and direction forecasts, **central bank forecasts** were added to historic actual macro data in monthly forward loops
- Model is trained and tested using Historic + Forecast Macro data to predict forward asset price
- Predicted asset price is then added to historic data for re-training model and predicting for next forward timeframe

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

Forward Price Asset model

- **$FPA_t = RFModel_{t-2} (Macro_{t-1})$**
- For forward asset price **prediction for month 't'**
  - Model is trained/tested on asset price and macro data till 't-2'
  - Macro data for 't-1' is used to predict asset price for 't', ensuring it's not used for training/testing the model (prevent leakage)
- **Portfolio is constructed** using a typical composition of different asset classes – Stock Index, Bonds, Gold, Real Estate, Currencies, Cash and Commodity Index
- **Create a medium to long-term portfolio for small/medium investors**
- **Investment strategy** will determine asset class and weight allocation. Example:

Low risk	Medium risk	High risk
Conservative	Moderate	Aggressive

- **Highly liquid,** Daily priced, Exchange traded Indices/Assets futures were considered
- **Formulas for return and risk calculations are given**

Portfolio Return

- **Portfolio Return = $\sum_{i=1 \text{ to } n} AR_i * W_i$** = Weighted sum of asset returns, over a given period (months to years)
- AR = Asset Return for period (Yearly/Monthly)
- W = % Weight of Asset in Portfolio for period
- Portfolio Weight grid example:

Period	StkIndex	REIndex	Gold	OilWTI	Bond10Yr	Cash	Total
3yr	0.4	0.05	0.2	0.1	0.2	0.05	1

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

Portfolio Risk - Variance (Volatility)

- **Portfolio Risk = Variance of Portfolio** = Volatility = V
- Calculated using **Modern Portfolio Theory** formula which considers both risk and correlation of assets in the portfolio
- **$V = \sum_{i=1 \text{ to } n} (\text{Variance}(AR_i) * W_i^{\wedge 2}) + \sum_{i=1 \text{ to } n} \sum_{j=i+1 \text{ to } n} [2 * W_i * W_j * \text{Covariance}(AR_i, AR_j)]$**
- **$\text{Covariance}(AR_i, AR_j) =$** $\text{CorrelationCoefficient}(AR_i, AR_j) * SDAR_i * SDAR_j$
- **SDAR** = Standard Deviation of Asset Return
- **Standard Deviation** measures deviation of the returns from the mean
  - **Standard Deviation (Portfolio) = [Variance(Portfolio)]^1/2**
  - Variance (x) = $\sum_{i=1 \text{ to } n}((x_i - \text{Average } x)^{\wedge}2)/n$. Spread from mean. Excess weight to outliers
  - Covariance (x,y) = $\sum_{i=1 \text{ to } n}[(x_i - \text{Avg } x) * (y_i - \text{Avg } y)]/n-1$. Directional relationship between returns of 2 assets (x,y)
  - Correlation Coefficient - Strength of relationship between 2 assets -1 to 1

Backtesting Logic

- **Backtesting** is used to measure accuracy of the model by predicting asset prices for actual macro and asset price data
  - Predict asset's monthly average price using actual Macro data for US, UK and EU
  - Loop in reverse, start at last year and month of actual data, drop 1 month each time, till last 1 year of Macro data is left
  - Use macro, asset data till t-2 to train model. Then t-1 macro data to predict asset price for t
  - We need at least 1 year’s macro, asset data to predict future asset price
- **Accuracy** - measures how accurately the model is predicting asset prices
  - Accuracy = Predicted monthly asset price - Actual monthly price, for historic data
- **Direction** - Calculates Predicted vs Actual direction using actual prices and macro data

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

Important Points

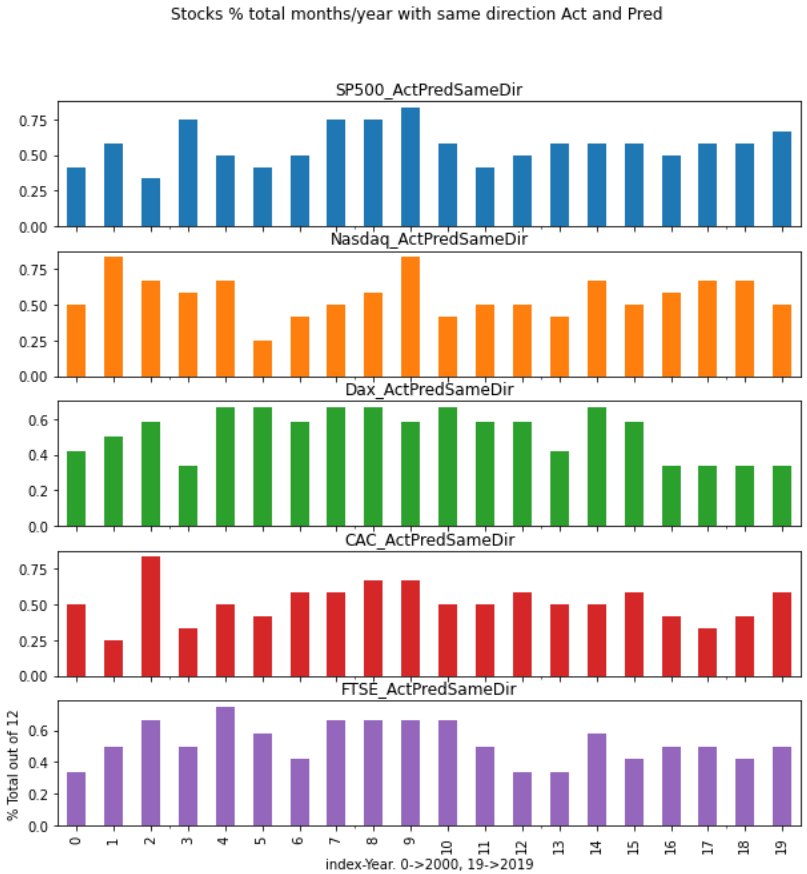
- ***The algorithm takes ~ 2 hrs to run*** *(depending on no of years' data and hyperparameters like trees, etc)*
- For this project demo it is run from local machine files
- Make sure to point the source directory to the folder containing the data and code
- The algorithm can be easily **deployed** to AWS cloud as a webserver API using Elastic Beanstalk and flask
- **S3** can used for cloud storage of files/code/model/results. These can then be accessed by different applications/users
- ***Legal Disclaimer*** *–* ***This RoboAdvisor model is for research only. It is not an investment advisory or a sales pitch.***

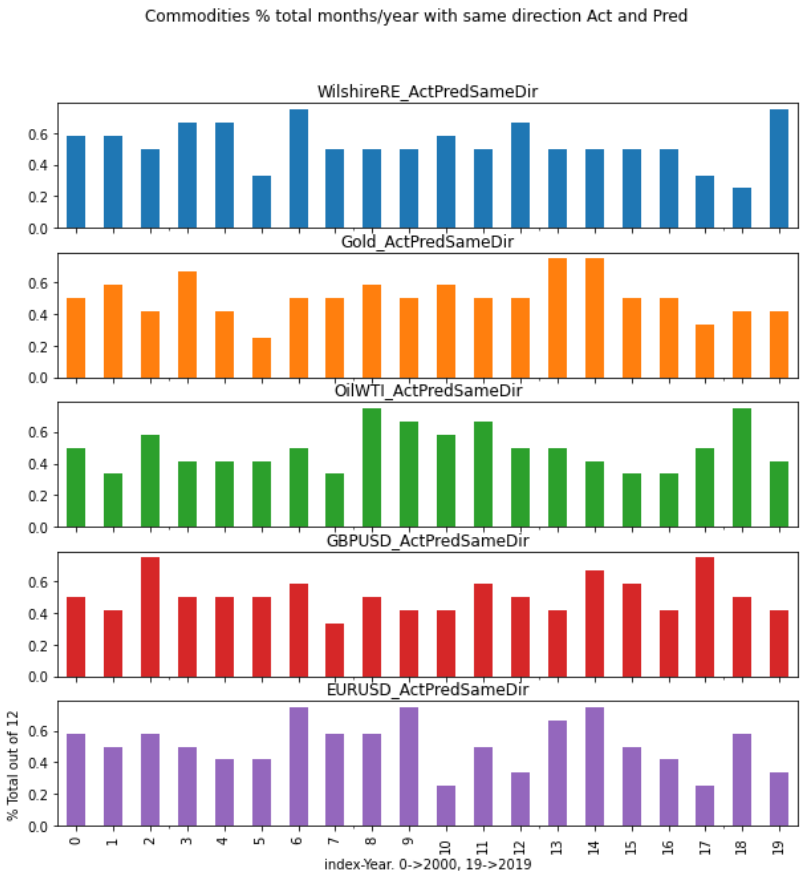
Conclusion

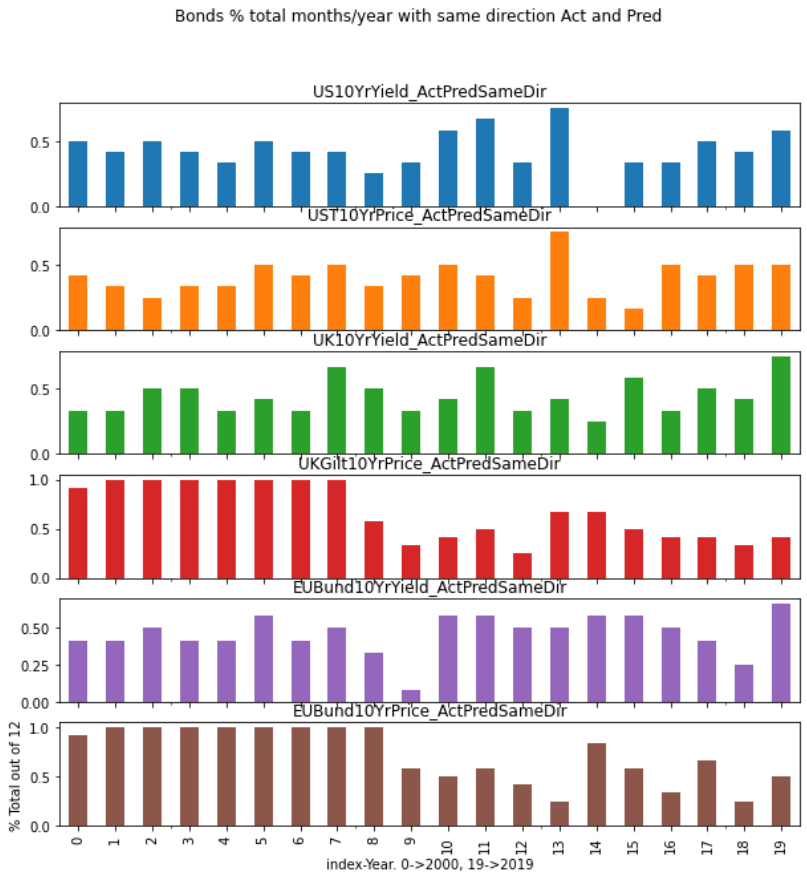
- **Predicted values are always lower than the actual – limitation of the algorithms**
  - Models learn from the range of data given for training
- **Portfolio returns are not good considering the recent market performance**
  - Only 5 macro factors were considered, and model requires latest data for re-training for more accurate results
  - **Portfolio optimization** is required monthly for rebalancing the weights and long/short positions, which will be covered in part 2 of the course
- **Actual and Forward Price/Direction Prediction -** Results are good but not accurate enough
  - While the predicted direction and prices are not as accurate as desired, they still show validity of the algorithm logic for all assets
- As other macro/market data and more samples are included, expectation is that it will improve the results
- **Charts summarising the results** are shown in the next slide
- ***Note -*** *This algorithm is still under development, with Part 2 to be released later*
  - ***Users*** *can expand or change the algorithm to forecast prices for any asset and use different risk & return techniques*
  - More details are given in: ***https://github.com/datawisdomx/RoboAdvisorAlgoPart1/***

Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

% total months for each year with same direction for Act and Pred, split by asset class – equities, commodities, currencies, bonds

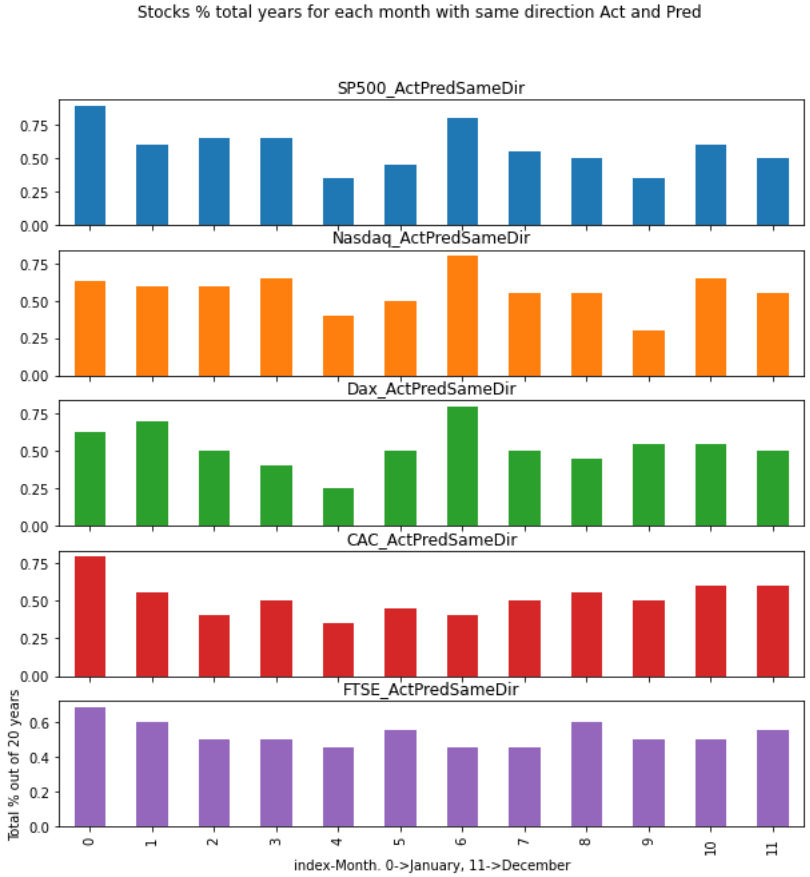


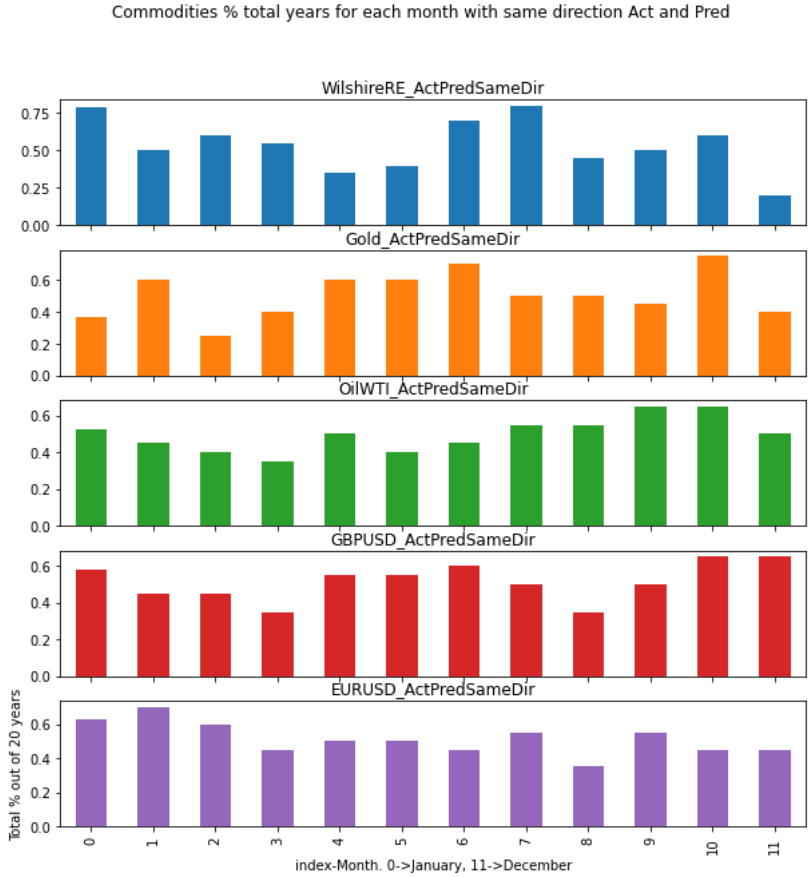


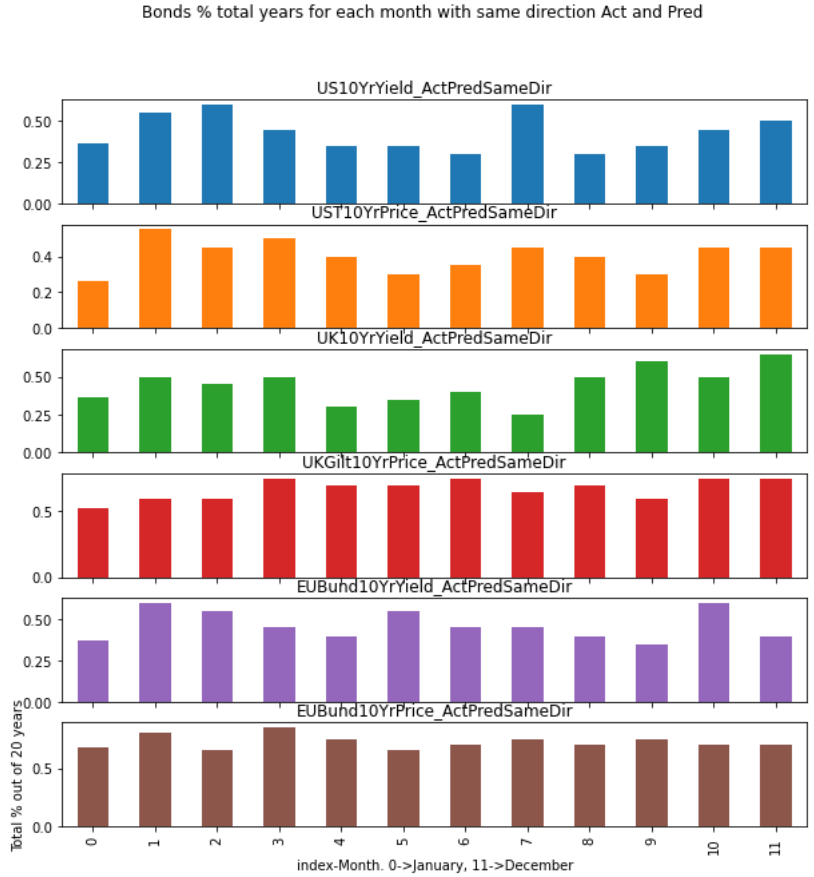


Chapter 14 – End 2 End Project 1 – Building a RoboAdvisor

% total years for each month with same direction for Act and Pred, split by asset class – equities, commodities, currencies, bonds

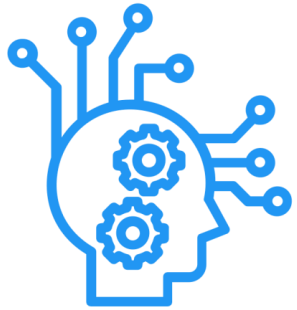






References, Data Sources, About Us, Contact

- **References** to online guides, books, videos, data sources are given for each section of the course
- There are a lot of free **data sources** available online but make sure to check license policy for commercial use
  - **Scikit-learn** - https://scikit-learn.org/stable/datasets.html
  - **Kaggle** - https://www.kaggle.com/datasets
  - **Data.world** - https://data.world/
  - **Google** – https://datasetsearch.research.google.com/
  - **Kdnuggets** - https://www.kdnuggets.com/datasets/index.html
  - **UCI Machine Learning Repository** - http://archive.ics.uci.edu/ml/datasets.php
    - Non-commercial, only use for research
  - **Github** - https://github.com/awesomedata/awesome-public-datasets
  - **Wikipedia** - https://en.wikipedia.org/wiki/List_of_datasets_for_machine-learning_research
- **About Us, Contact** – https://datawisdomx.com/index.php/about-us/
- nitin@datawisdomx.com, contact@datawisdomx.com



DataWisdomX

Practical Data Science for Everyone