

See discussions, stats, and author profiles for this publication at: <https://www.researchgate.net/publication/344612287>

A Project Report on Data Aggregation and analysis with Python in the field of Renewable Energy Systems MASTER OF ENGINEERING IN RENEWABLE ENERGY SYSTEMS

Technical Report · October 2020

CITATIONS

0

READS

10,299

1 author:



[Suraj Jat](#)

Nordhausen University of Applied Science

1 PUBLICATION 0 CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:



Data Aggregation and analysis with Python in the field of Renewable Energy Systems [View project](#)



A Project Report on

Data Aggregation and analysis with Python in the field of Renewable Energy Systems

submitted by

SURAJ BHANU JAT (41677)

Project guide: **Dr. Ing. Birgit Lustermann**

Subject: **Scientific Project - II**

MASTER OF ENGINEERING IN RENEWABLE ENERGY SYSTEMS

HOCHSCHULE NORDHAUSEN,
NORDHAUSEN GERMANY

2020-21

Abstract

As each and every sector of the market is growing, data is building up day by day, we need to keep the record of the data which can be helpful for the analytics and evaluation. Now we don't have data in gigabyte or terabyte but in zetta byte and petabyte and this data can not be handled with the day to day software such as Excel or Matlab. Therefore in this report we will be dealing with large data sets with the high-level programming language 'Python'.

The main goal of this project is to aggregate and analyze the data collected from the different data sources available on the internet. This project mainly focuses on the usage of the python programming language in the field of renewable energy. This language has not only it's application in the field of just analyzing the data but also for the prediction of the upcoming scenarios in the energy field.

The purpose of using this specific language is due to its versatility, vast libraries (Pandas, Numpy, Matplotlib, etc.), speed limitations, and ease of learning. We will be analyzing large energy data sets in this project which can not be easily analyzed in other tools as compared to python. Python does not have it's limitation to only data analytics but also in many other fields such as Artificial intelligence, Machine learning, and many more.

Contents

1	Data Science Introduction	4
1.1	Data Science	4
1.2	Stages of Data Science	4
1.3	Use of Data Science in Renewable energy	6
2	Python Programming language basics	8
2.1	Why only Python?	8
2.2	Data structures in Python	9
2.3	Operators	13
2.4	Condition statements	17
2.4.1	If else statements	17
2.4.2	elif statements	18
2.5	Loops in python	20
2.5.1	For loop	20
2.5.2	While loop	21
2.6	Module, Package and Functions	22
3	Libraries in Python	24
3.1	Matplotlib	25
3.2	Pandas	26
3.3	NumPy	28
4	Data collection	29
5	Global Wind and Solar Production from 1990 to 2014	31
6	Ethanol production from 1981 to 2019 in USA	44
	Bibliography	56
	List of Figures	57

Chapter 1

Data Science Introduction

1.1 Data Science

Data science is the field of data analytics and data visualization in which raw data or the unstructured data is cleaned and made ready for the analysis purpose. Data scientists use this data to get the required information for the future purpose.[1] "Data science uses many processes and methods on the big data, the data may be structured or unstructured". Data frames available on the internet is the raw data we get. It may be either in unstructured or semi structured format. This data is further filtered, cleaned and then number of required task are performed for the analysis with the use of the high programming language. This data is further analyzed and then presented for our better understanding and evaluation.

One must be clear that data science is not about making complicated models or making awesome visualization neither it is about writing code but about using the data to create an impact for your company, for this impact we need tools like complicated data models and data visualization.

1.2 Stages of Data Science

There are many tools used to handle the big data available to us. [2] "Data scientists use programming tools such as Python, R, SAS, Java, Perl, and C/C++ to extract knowledge from prepared data".

Data scientists use many algorithms and mathematical models on the data.

Following are the stages and their cycle performed on the unstructured data.[3]

- Identifying the problem.
- Identify available data sources

- Identify available data sources
- Identify if additional data sources are needed.
- Statistical analysis
- Implementation, development
- Communicate results
- Maintenance



Figure 1.1: 7 steps that together constitute this life-cycle model of Data science[3]

Data science finds its application in many fields. With the assistance of data science it is easy to get the search query on search engines in plenty of time. A role of the data scientist is to have a deep understanding of the data as well as a good command on the programming language, he should also know how to work with the raw data extracted from the data source. Many programming languages are used to analyze and evaluate the data such as Python, Java, MATLAB, Scala, Julia, R., SQL and TensorFlow. Among which python is the most user friendly and vastly used programming language in the field of data science.

This life cycle is applied in each and every field, in this project we will be considering all this seven stages of data science to analyze the data. The process will be starting from data collection, data preparation, data modeling and finally data evaluation. For instance, As we have huge

amount of data we can create an energy model for a particular country by collecting its previous energy data, we can also predict the future requirement of it with the same data.

1.3 Use of Data Science in Renewable energy

As the number of renewable energy systems are increasing the renewable energy data is increasing through sensors and other aspects of energy systems. So again this big data can be helpful in not even understanding the current scenario of the renewable energy sector but can also be helpful in forecasting the renewable energy consumption as well as production both.

Following are the applications of data science which plays a major role in the field of renewable energy.

- **Improving the current technology** This is mostly used in the field of solar energy. Data of solar panels are collected using sensors and by analysing that data pattern we can improve the efficiency as well as life span of the particular solar panel.
- **Renewable energy consumption prediction** The consumption of renewable energy by the customers can also be predicted with the help of past data of energy consumption by the customers. This can be so helpful in fulfilling customers requirement in future.
- **Renewable energy production forecasting** Solar energy and wind energy production can be optimized by considering the weather condition and environmental condition data. With this data, forecasting can be easily done.
- **Reducing Renewable Energy Production Costs** With the help of the big energy data available to us we are able to predict the production cost of renewable energy easily from forecasting model. The price of the energies are declining just because of the big data and forecasting model available to us.[4] Renewable energy will be cost competent with its conventional counterparts.
- **Efficient backup facility for power plants** With the help of computational models we can easily get the high and low power usage and when there is abundant power we can save the power which may be wasted vice versa when there is shortage of power we can provide with the help of our renewable energy systems.

Below is the figure shows the role of Data Science and Big Data Analytics in the Renewable Energy Sector.

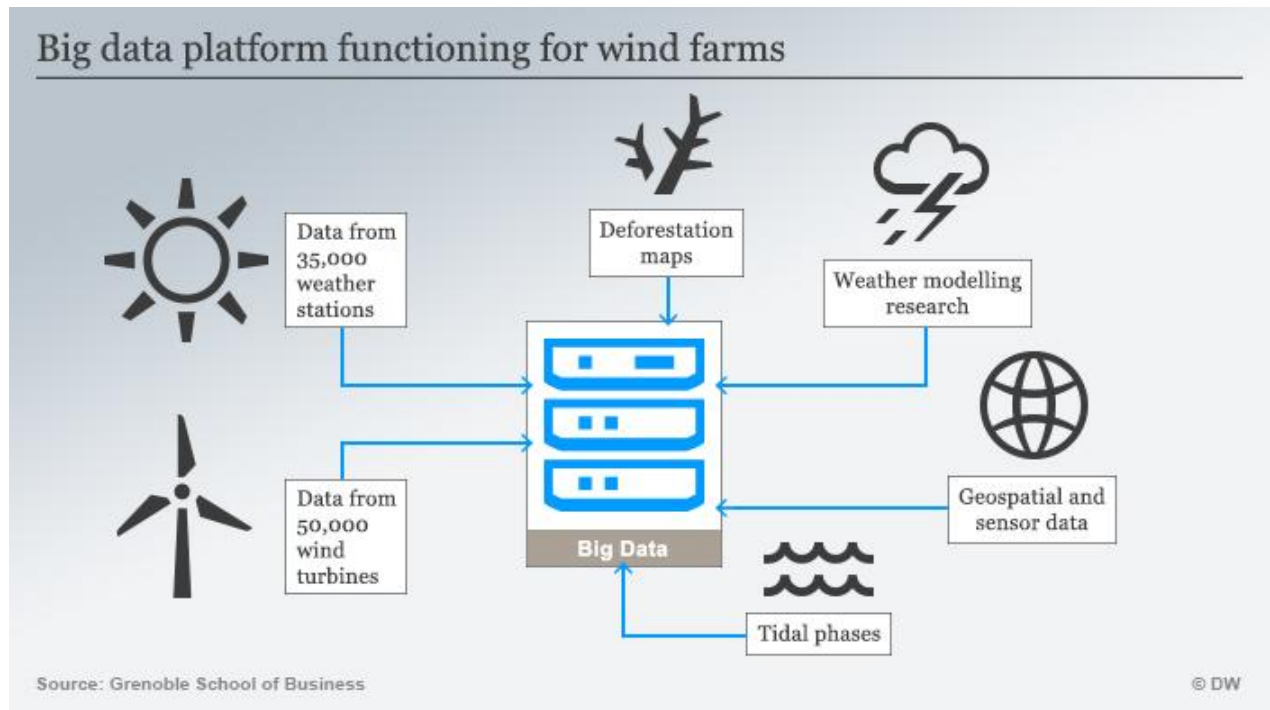


Figure 1.2: Role of Data Science and Big Data Analytics in the Renewable Energy Sector [5]

For instance in a wind energy project, we need a specific location where all the demands of projects are fulfilled, and this demands can be captured by considering multiple data as well as factors which will help in setting up the project. For example - weather data and wind data.

Chapter 2

Python Programming language basics

2.1 Why only Python?

"Python is an interpreted, object-oriented, high-level programming language with dynamic semantics".[6] This language consist of mainly data structures which make it very easy for the data scientists to analyse the data very effectively. It does not only help in forecasting and analysis it also helps in connecting the two different languages.Two best features of this programming language is that it does not have any compilation step as compared to the other programming language in which compilation is done before the program is being executed and other one is the reuse of the code, it consist of modules and packages due to which we can use the previously written code any where in between the program whenever is required.

There are multiple languages for example R., Java, SQL, Julia, Scala, MATLAB available in market which can be used to analyze and evaluate the data, but due to some outstanding features python is the most famous language used in the field of data science.

Python is mostly used and easy among all other programming languages is due to the following reasons.

2.2 Data structures in Python

Data structures are the way of storing the data so that we can easily perform different operations on the data whenever its required. When the data has been collected from the data source the data is available in different forms. So later it is easy for the data scientists to perform different operation on the data once it is sorted in to different data structures.

Data structures are mainly classified in to two categories and then further their subcategories shown below.

1. Primitive Data Structures.

They are also called as basic data structures. This type of data structures contains simple values of the data.[7]

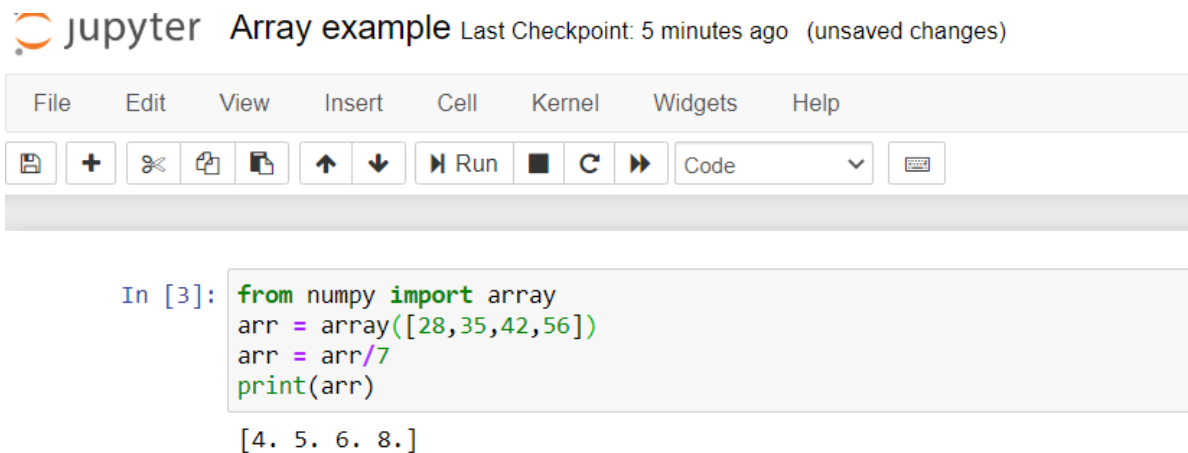
- **Integers** - All the whole numbers from negative infinity to positive infinity comes under integer data types. for example 4,9,-2,-6.
- **Float** - The decimal figure numbers or rational numbers comes under float data types. for example 3.1,2.2,8.96
- **Strings** - Collection of alphabets or characters are called strings. We enclose the string either in single or double quotes in python. for example 'hello' and "bread".
- **Boolean**- These are the built in data types which take two values that are 'True' and 'False'. True represents the 1 and False represents 0 in python.

2. Non-Primitive Data Structures

These are the derived type or reference variables data structures. They are called derived data structures because they derived from the basic data structures such as integer and float. Python has mainly five types of data structures.

Following are the non primitive data structures.

Array - Array is the collection of data types of same type. Arrays data structures are used mostly in the NumPy library of python. In the below example we have first imported the package array from numpy library and defined the array as variable arr then divided the array by 7 and we have printed our array to get output.



The image shows a Jupyter Notebook interface with the title "Array example" and a status bar indicating "Last Checkpoint: 5 minutes ago (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following Python code:

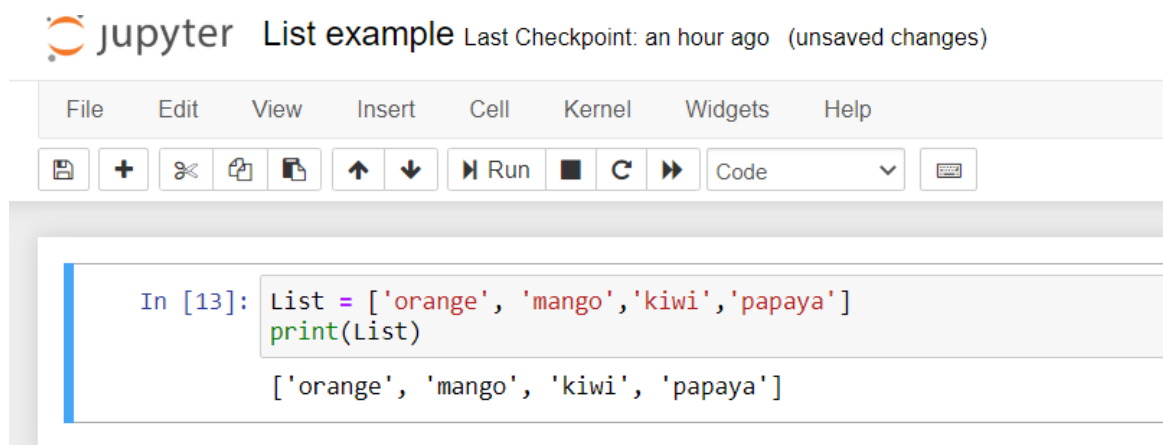
```
In [3]: from numpy import array
arr = array([28,35,42,56])
arr = arr/7
print(arr)
```

The output of the code is displayed below the code cell:

```
[4. 5. 6. 8.]
```

Figure 2.1: Array example

List - "A list is a value that contains multiple values in an ordered sequence".[8]. Values in the list referred to list itself, that is the value can be stored in a variable or passed to a function. List are changeable and values in the list are enclosed inside a square bracket, we can perform multiple operations such as indexing, slicing, adding and multiplying.



The image shows a Jupyter Notebook interface with the title "List example" and a status bar indicating "Last Checkpoint: an hour ago (unsaved changes)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The code cell contains the following Python code:

```
In [13]: List = ['orange', 'mango', 'kiwi', 'papaya']
print(List)
```

The output of the code is displayed below the code cell:

```
['orange', 'mango', 'kiwi', 'papaya']
```

Figure 2.2: list example

Tuple - A tuple is a list of non changeable objects. The differences between tuples and lists are that the tuples cannot be changed, tuples use parentheses, whereas list uses square brackets.

```
File Edit View Insert Cell Kernel Widgets Help
[Icons] Run [Dropdown]

In [16]: Tuple = ('orange', 'mango', 'kiwi', 'papaya')
          #If you can see the change values are same as list but enclosed in parentheses
          print(Tuple)

          ('orange', 'mango', 'kiwi', 'papaya')
```

Figure 2.3: tuple example

Dictionary- These are nothing but a type of data structure which consist of key value pairs enclosed in the curly brackets. It is same as the any dictionary we use in day to day life in which we find the meaning of the particular words. So if I compare normal dictionary to this python dictionary data structure then the a word in a dictionary will be our key and its meaning will be the value of the dictionary. In the figure name, occupation and hobby are the keys and Suraj, data analyst and vlogging are the values assigned to the keys.

```
File Edit View Insert Cell Kernel Widgets Help
[Icons] Run [Dropdown]

In [25]: dictionary = {'name': 'Suraj', 'occupation': 'data analyst', 'hobby': 'vlogging'}
          dictionary['hobby']

Out[25]: 'vlogging'
```

Figure 2.4: dictionary example

Sets - Set are used for calculating mathematical operation such as union, intersection and symmetric difference.

Below is the data structure tree which explains the category and sub-category of each data types.

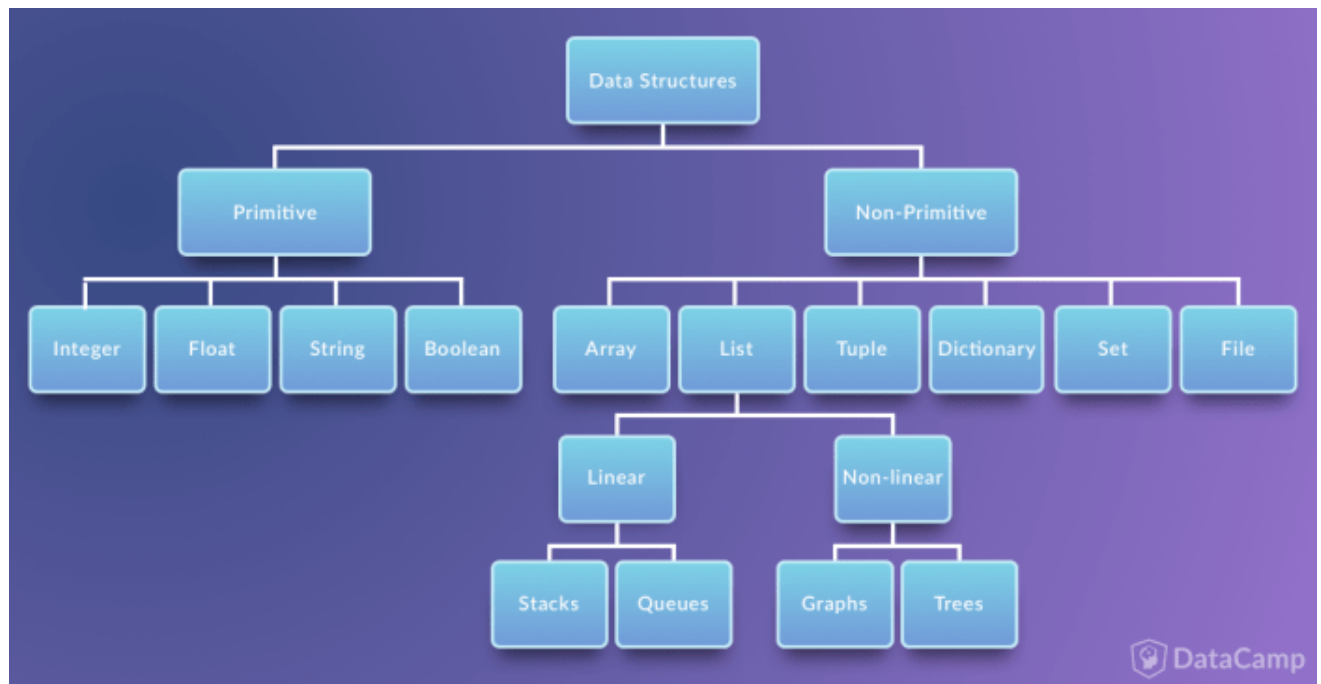


Figure 2.5: A data structure tree at glance
[7]

2.3 Operators

OPERATORS - Operators are the symbols in python that are used to perform Arithmetic or logical operations. Following are the different types of operators in python.

Arithmetic operators - Arithmetic operators carry out mathematical operations and they are mostly used with the numeric values.

Arithmetic operators		
Operator	Name	Example
+	Addition	A+B
-	Subtraction	A-B
*	Multiplication	A*B
/	Division	A/B
%	Modulus	A%B
**	Exponentiation	A**B
//	Quotient	A//B

Table 2.1: Arithmetic operators

A and B are the numeric values.

Assignment operators - As the name decides this operators are used for assigning the values to the variables.

ASSIGNMENT OPERATORS		
Operator	Example	may also be written
=	a = 6	a = 6
+=	a += 3	a = a + 3
-=	a -= 4	a = a - 4
*=	a *= 5	a = a * 5
/=	a /= 6	a = a / 6
%=	a %= 7	a = a % 7
//=	a //= 8	a = a // 8
**=	a **= 9	a = a ** 9
&=	a &= 1	a = a & 1

Table 2.2: Assignment Operators

Here a is any value and number of operations are performed on this value.

Logical operators - These operators are used to join conditional statements

Logical Operators		
Operator	Description	Example
and	if both statements are true it returns true	x <5 and x <10
or	if any of the two statement is true it returns true	x <4 or x <8
not	if the result is true it reverses the result and gives false	not (x <4 and x <8)

Table 2.3: Logical Operators

Here a is any value provided by us and on which multiple operations can be performed.

Comparison operators - These operators are used to compare two different values.

Comparison operators		
Operator	Name	Example
==	Equal	a == b
!=	Not equal	a!=b
>	Greater than	a >b
<	less than	a =	Greater than equal to	a>= b
<=	less than equal to	a <=b

Table 2.4: Comparison operators

Here a and b are two different values and these values are compared.

Membership operators - These operators are used to check membership of a particular value. It is used to check whether a specific value is present in the object or not.

Membership operators		
Operator	Description	Example
in	it returns a True if the value is present inside the object	a in b
not in	it returns a True if the value is not present inside the object	a not in b

Table 2.5: Membership operators

2.4 Condition statements

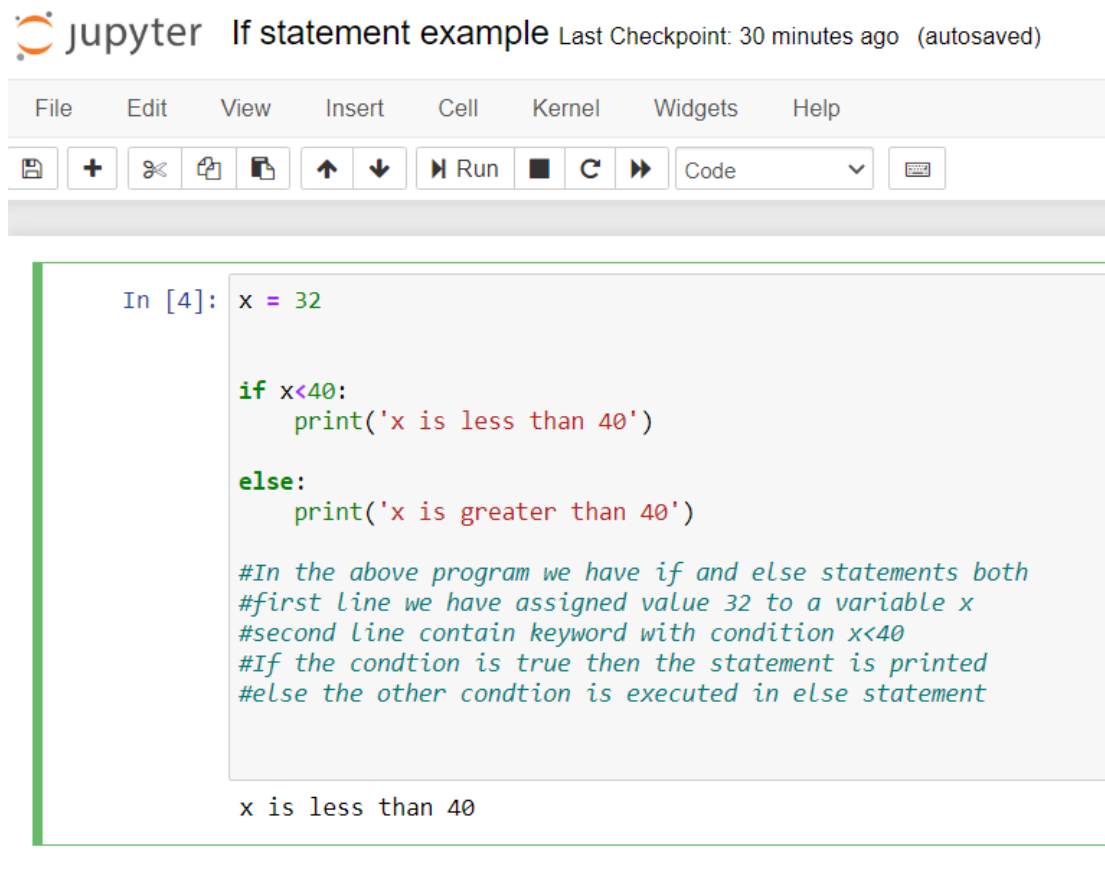
2.4.1 If else statements

”The most common type of statement is the if statement. if statement consist of a block which is called as clause”,[8] it is the block after if statement, it executed the statement if the condition is true. The statement is omitted if the condition is False. then the statement in the else part is printed

If statement consist of following -

- **If keyword itself**
- **Condition which may be True or False**
- **Colon**
- **If clause or a block of code**

Below is the figure shows how If and else statements are used with description inside it.



The screenshot shows a Jupyter Notebook interface with the title "If statement example" and a status bar indicating "Last Checkpoint: 30 minutes ago (autosaved)". The notebook has a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, running, and other functions. The main area of the notebook displays a code cell with the following content:

```
In [4]: x = 32

if x<40:
    print('x is less than 40')

else:
    print('x is greater than 40')
```

Below the code cell, there is a text area containing the following comments:

```
#In the above program we have if and else statements both
#first line we have assigned value 32 to a variable x
#second line contain keyword with condition x<40
#If the condtion is true then the statement is printed
#else the other condtion is executed in else statement
```

At the bottom of the notebook, the output of the code cell is displayed:

```
x is less than 40
```

Figure 2.6: if else statement

2.4.2 elif statements

In this statement only one statement is executed, There are many cases in which there is only one possibility to execute. "The elif statement is an else if statement that always follows an if or another elif statement"[8]. The elif statement provides another condition that is checked only if any of the previous conditions were False. In code, an elif statement always consists of the following:. The only difference between if else and elif statement is that in elif statement we have the condition where as in else statement we do not have any condition.

elif statement consist of following -

- **elif keyword itself**
- **Condition which may be True or False**
- **Colon**
- **elif clause or a block of code**

Below is the figure shows how elif statement is used with description inside it.

```
File Edit View Insert Cell Kernel Widgets Help
```

```

In [9]: var = 't'

if var == 'a':
    print('this is the vowel a')
elif var == 'e':
    print('this is the vowel e')
elif var == 'i':
    print('this is the vowel i')
elif var == 'o':
    print('this is the vowel o')
elif var == 'u':
    print('this is the vowel u')
else:
    print('The value in variable var is constant')

#In the above program we have if, else and elif statements.
#first line we have assigned value t to a variable 'var'
#second line contain keyword if with condition if var==a
#If the condtion is true then the statement is printed
#elif the other condtion is executed in elif statement in furhter lines
#if all the conditions are not true then we come to else statement
#and then statement in else block is printed on the output

```

The value in variable var is constant

Figure 2.7: elif example

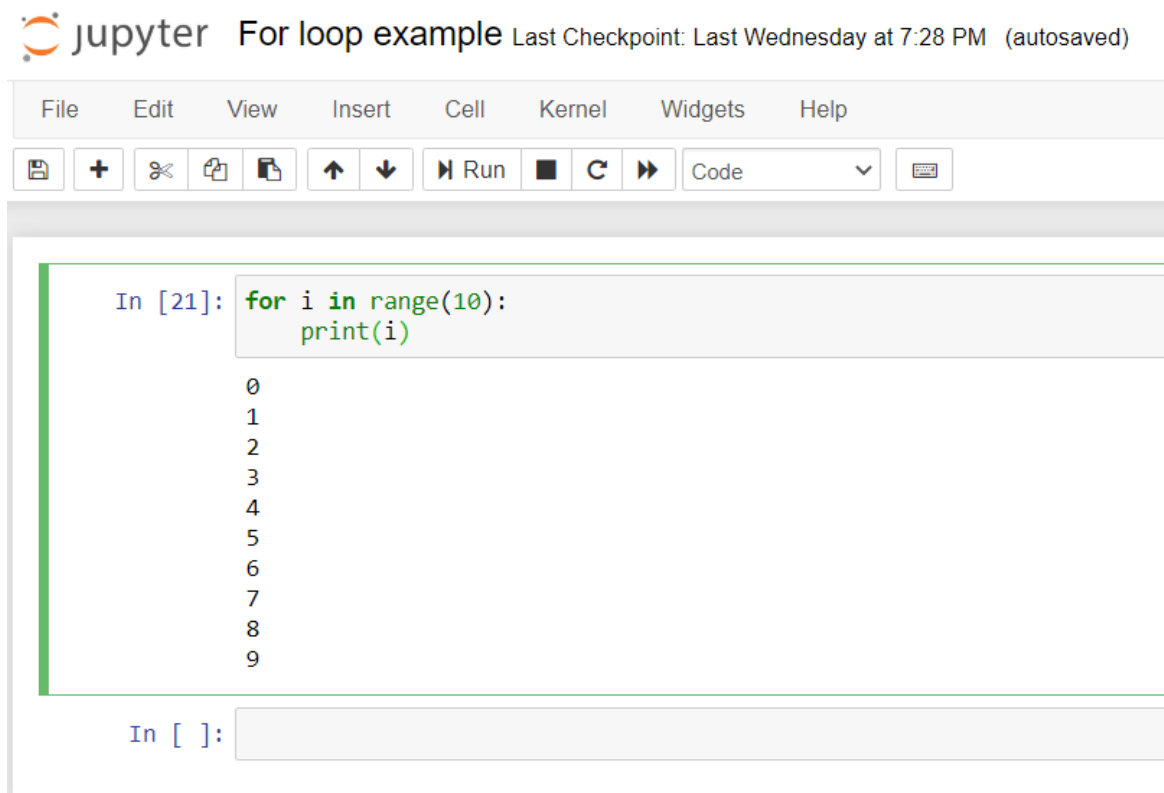
2.5 Loops in python

2.5.1 For loop

When do we use for loops ?

for loops are traditionally used when you have a block of code which you want to repeat a fixed number of times. The Python for statement iterates over the members of a sequence in order, executing the block each time.[9]

Range statement - This statement 'range()' is used with for loop statements where you can specify one value. For example, if you specify 10, the loop statement starts from 1 and ends with 9, which is n-1. Also, you can specify the start and end values. The following examples demonstrate loop statements.



The image shows a Jupyter Notebook interface with the title "For loop example" and a subtitle "Last Checkpoint: Last Wednesday at 7:28 PM (autosaved)". The interface includes a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, and running code. The main area displays a code cell with the following content:

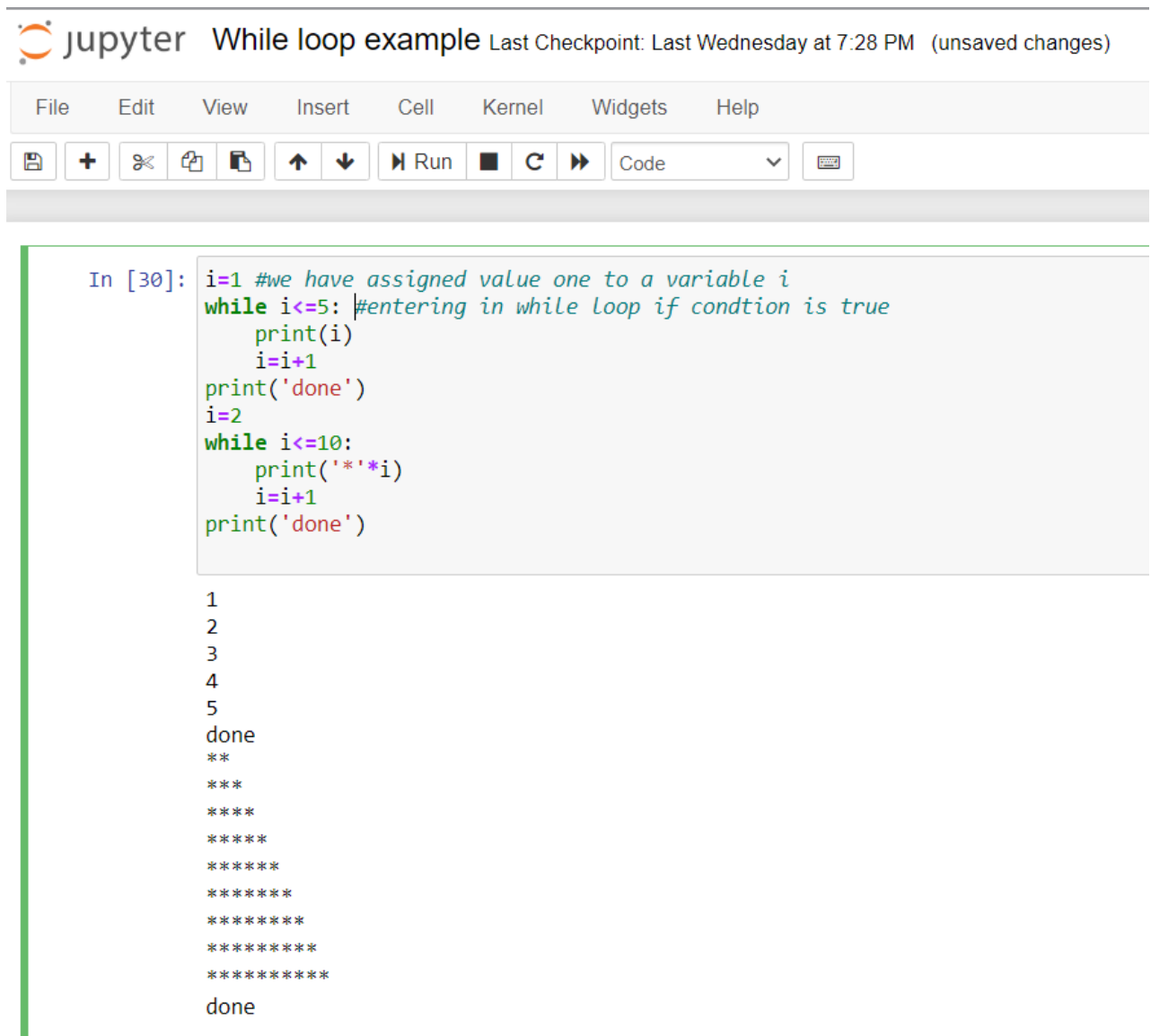
```
In [21]: for i in range(10):  
         print(i)
```

The output of the code cell is a list of numbers from 0 to 9, each on a new line, indented under the code. Below the code cell is an empty input field labeled "In []:".

Figure 2.8: for example with range statement

2.5.2 While loop

While loops are used for repeating the section of code but not same as for loop, the while loop does not run n times, but until a defined condition is no longer met. If the condition is initially false, the loop body will not be executed at all.



The image shows a Jupyter Notebook interface with the title "While loop example". The top bar includes the Jupyter logo, the title, and the text "Last Checkpoint: Last Wednesday at 7:28 PM (unsaved changes)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Under the menu bar is a toolbar with icons for saving, adding cells, undo, redo, copy, paste, up/down arrows, a "Run" button, a "Code" dropdown menu, and a "Help" icon. The main area contains a code cell with the following Python code:

```
In [30]: i=1 #we have assigned value one to a variable i
while i<=5: #entering in while loop if condition is true
    print(i)
    i=i+1
print('done')
i=2
while i<=10:
    print('*'*i)
    i=i+1
print('done')
```

The output of the code is displayed below the code cell:

```
1
2
3
4
5
done
**
***
****
*****
*****
*****
*****
*****
*****
*****
done
```

Figure 2.9: While loop example

2.6 Module, Package and Functions

- **Module**

Modules are Python files which has extension as .py. The name of the module will be the name of the file. A Python module can have a set of functions, classes or variables defined and implemented.

Module has some python codes, this codes can define the classes, functions and variables. The reason behind using the module is that it organizes your python code by grouping the python code so that it is easier to use.

- **Package**

A package consist of the collection of modules in which python codes are written with name init.py. It means that each python code inside of the python path, which contains a file named init.py, will be treated as a package by Python. Packages are used for organizing the module by using dotted names.

for example -

We have a package named simple package which consist of two modules a and b. We will import the module from package in following way.

```
from simple package import a, b
```

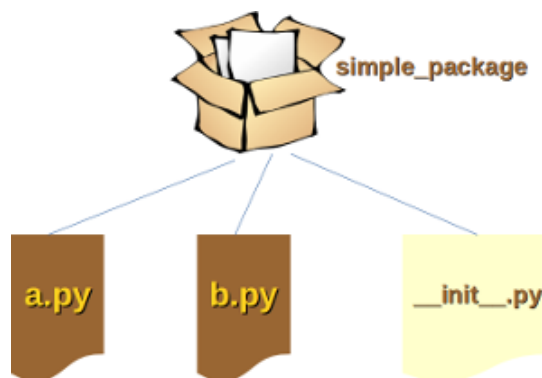
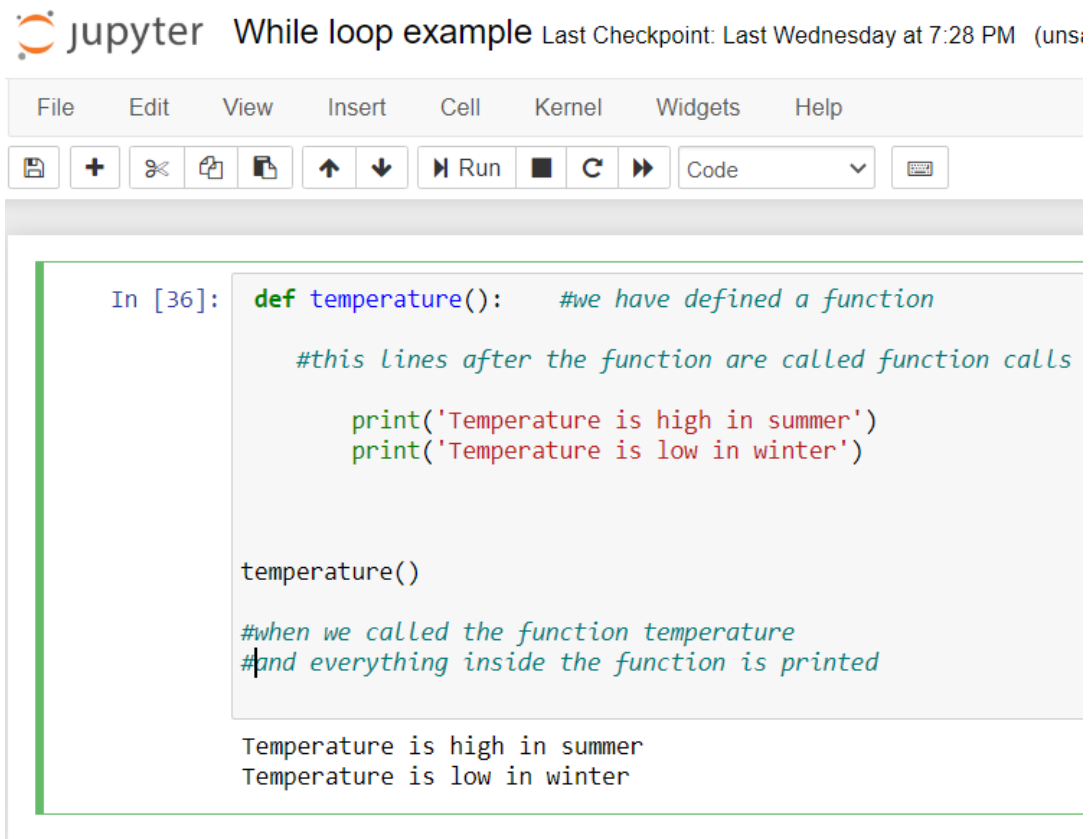


Figure 2.10: packages example [10]

- **Functions**

A function is a python code which can be reused at any anytime in the whole python code. Function performs specific task whenever it is called during the program. With the help of function the program is divided in to multiple codes.

- **Built in functions** - The functions which are already in the python programming and have specific action to perform are called as built in functions. This function are immutable. Some examples of this functions are -
chr() - used to get string
print() - used to print an object in terminal
min() - used to get minimum value in terminal
- **User defined functions** - This functions are user to defined functions and it starts with the key word 'def' as shown in the example below. We have defined the function names temperature and its task to be performed when called. Below is the example of it.



The image shows a Jupyter Notebook interface with a title bar that says "jupyter While loop example" and "Last Checkpoint: Last Wednesday at 7:28 PM (unsaved)". Below the title bar is a menu bar with "File", "Edit", "View", "Insert", "Cell", "Kernel", "Widgets", and "Help". Below the menu bar is a toolbar with icons for saving, adding, deleting, copying, pasting, undo, redo, and running code. The main area of the notebook shows a code cell with the following text:

```
In [36]: def temperature():    #we have defined a function

         #this lines after the function are called function calls

         print('Temperature is high in summer')
         print('Temperature is low in winter')

         temperature()

         #when we called the function temperature
         #and everything inside the function is printed

Temperature is high in summer
Temperature is low in winter
```

Figure 2.11: function example

Chapter 3

Libraries in Python

Python library is vast. There are built in functions in the library which are written in C language. This library provide access to system functionality such as file input output and that is not accessible to Python programmers. This modules and library provide solution to the many problems in programming.

Following are some Python libraries.

Matplotlib

Pandas

TensorFlow

Numpy

Keras

PyTorch

LightGBM

Eli5

SciPy

3.1 Matplotlib

”Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy”[11]. Matlab provides an application that is used in graphical user interface tool kits. Another such library is pylab which is almost same as MATLAB.

It is a library for 2D graphics, it finds its application in web application servers, graphical user interface toolkit and shell. Below is the example of a basic plot in python.

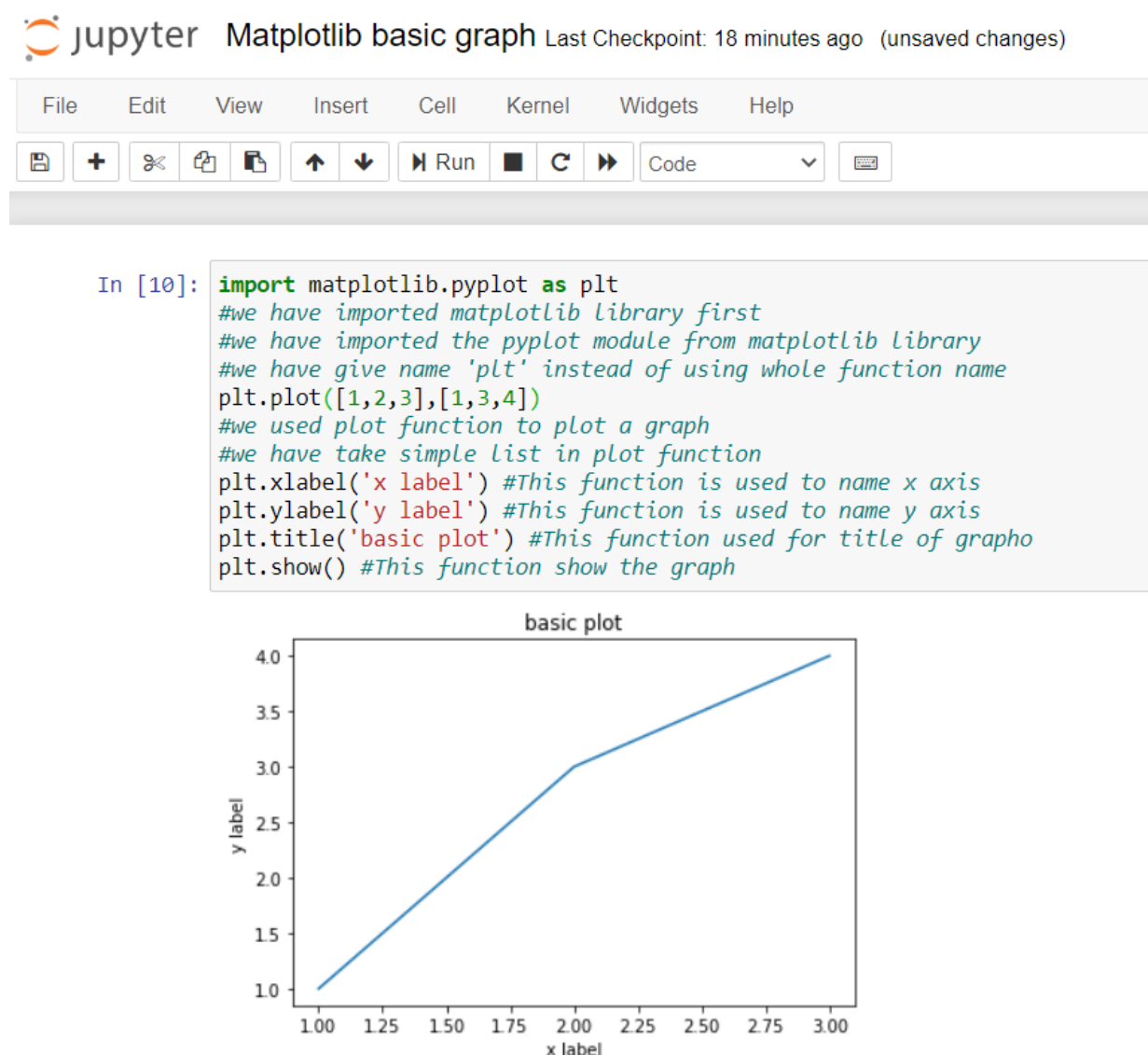


Figure 3.1: Matplotlib basic example

3.2 Pandas

Pandas is also a library or a data analysis tool in python which is written in python programming language. It is mostly used for data analysis and data manipulation. It is also used for data structures and time series.

We can see the application of python in many fields such as - Economics, Recommendation Systems - Spotify, Netflix and Amazon, Stock Prediction, Neuro science, Statistics, Advertising, Analytics, Natural Language Processing. Data can be analyzed in pandas in two ways -

Data frames - In this data is two dimensional and consist of multiple series. Data is always represented in rectangular table.

Series - In this data is one dimensional and consist of single list with index.

File
Edit
View
Insert
Cell
Kernel
Widgets
Help

Code

```

In [7]: #SERIES
import pandas as pd
#We are first importing the Librabry
#and keeping its name as 'pd' for our convenience
odd_numbers = pd.Series([3,9,13,15])
#we have imported series array from pandas
odd_numbers

```

```

Out[7]: 0      3
        1      9
        2     13
        3     15
        dtype: int64

```

```

In [32]: #DATAFRAME - IT HAS TWO OR MORE ARRAYS IN IT
info_stu = {'students':['john','mike','harry','robert'],
            'age':[28,26,29,25],
            'country':['spain','rome','holand','russia']}
#here we have defind our 3 differnet arrays

#then we have imported 'DataFrame' from pandas

info =pd.DataFrame(info_stu)

info

#0,1,2,3 are the index number of different rows

```

```

Out[32]:

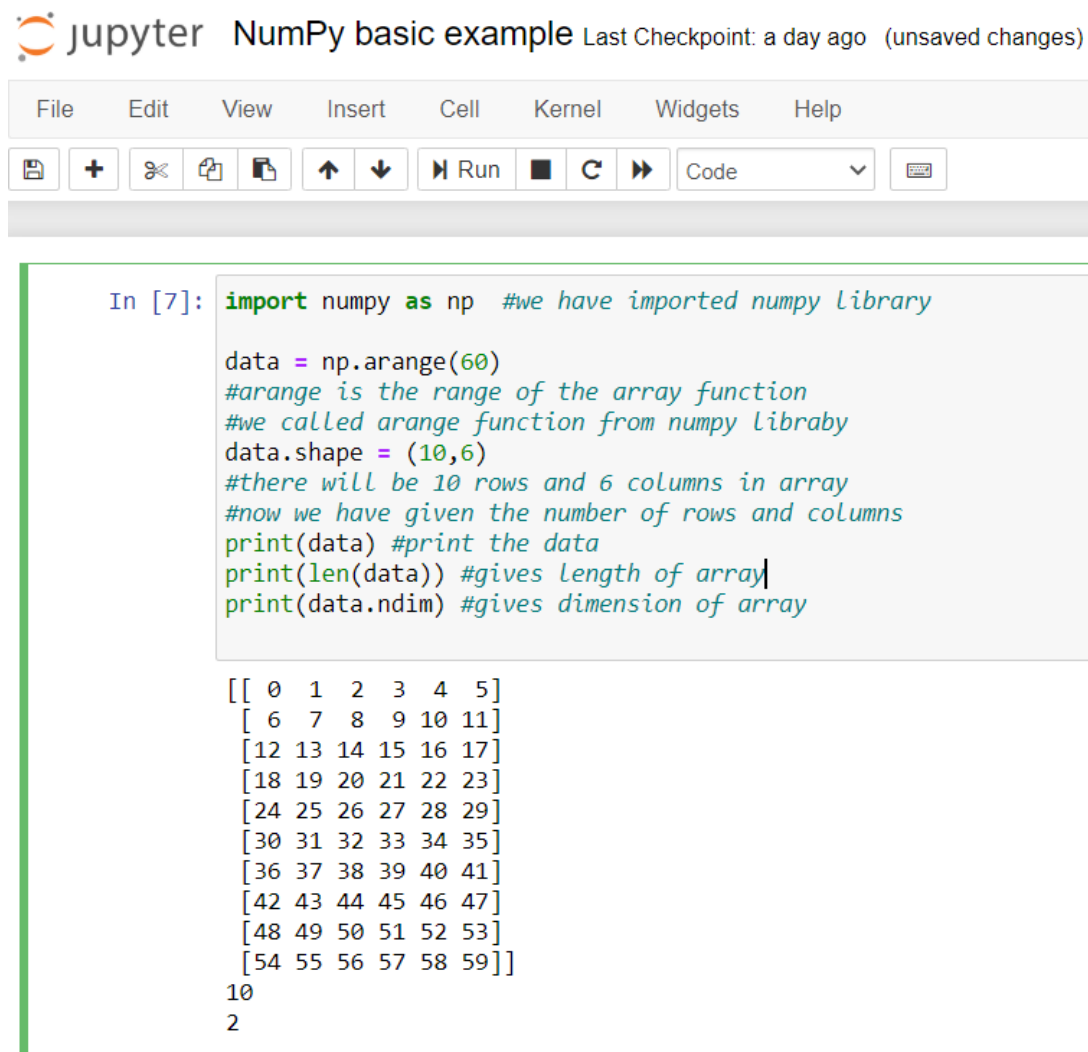
```

	students	age	country
0	john	28	spain
1	mike	26	rome
2	harry	29	holand
3	robert	25	russia

Figure 3.2: series and data frame in pandas

3.3 NumPy

”NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays”. The previous similar programming of NumPy is Numeric, and this language was originally created by Jim Hugunin with contributions from several other developers. In 2005, Travis Oliphant created NumPy by incorporating features of the competing Numarray into Numeric, with extensive modifications. [12] It is an open source library and free of cost.



```
In [7]: import numpy as np #we have imported numpy library

data = np.arange(60)
#arange is the range of the array function
#we called arange function from numpy library
data.shape = (10,6)
#there will be 10 rows and 6 columns in array
#now we have given the number of rows and columns
print(data) #print the data
print(len(data)) #gives length of array
print(data.ndim) #gives dimension of array

[[ 0  1  2  3  4  5]
 [ 6  7  8  9 10 11]
 [12 13 14 15 16 17]
 [18 19 20 21 22 23]
 [24 25 26 27 28 29]
 [30 31 32 33 34 35]
 [36 37 38 39 40 41]
 [42 43 44 45 46 47]
 [48 49 50 51 52 53]
 [54 55 56 57 58 59]]
10
2
```

Figure 3.3: NumPy basic example

Chapter 4

Data collection

Before analyzing and visualization we need the raw data and this raw data can be gathered from different open source data websites available on the internet. This data will be in raw form, it may be the PV solar panel sales, renewable energy consumption or production in any specific area or regions where solar or wind which one is more favorable. As here we are focusing on the renewable energy data sets so we will be considering following websites where this data is available.

<https://www.eia.gov/>

This website contains the energy data mostly of US. EIA is the abbreviated form of Energy Information Administration. Here we have different data of prices, consumption, production, exports and imports of the energy data.

<https://www.energy.gov/>

Energy.gov is the other website for data related to renewable energy. This Energy Department is responsible to make sure USA's Energy Future and solve the energy related problems

<https://openei.org/>

Open Energy Information is a website for policy makers, researchers, technology investors, venture capitalists, and market professionals with energy data, information, analyzes, tools, images, maps, and other resources.

<https://data.world/>

Here we can find data related to each and every field, it is the most widely used website for data analysis. We can also gather energy related data from this website.

<https://catalog.data.gov/dataset>

Data.gov is powered by two open source applications, CKAN and WordPress, and it is developed publicly on GitHub. Data.gov is managed and hosted by the U.S. General Services Administration, Technology Transformation Service.

<https://www.kaggle.com/>

The data sets available here is not specifically for renewable energy. Kaggle is general data sets website, here you can get generalized data.

Chapter 5

Global Wind and Solar Production from 1990 to 2014

This raw data consist of imports, exports, production, consumption, transformation and energy losses of all the countries from the year 1990 to 2014. It consist of mostly renewable energy data as well as the other energy data. We are going to analyze the data of wind and solar from the year 1990 to 2014, we will also focus on the other countries trends and compare them. We will also be comparing the wind energy production with the solar energy production between the countries.

Data collection methodology The data set has many data points such as country, commodity transaction, year, unit, quantity, and category (Type of energy used in that particular sector).

Statistical and Analytic Issues There are is blank column in the data set named quantity footnotes as it consist of the blank cells in the data sets for which data has not been provided. So we have omitted that column. Blank cells shows the data that were not available.

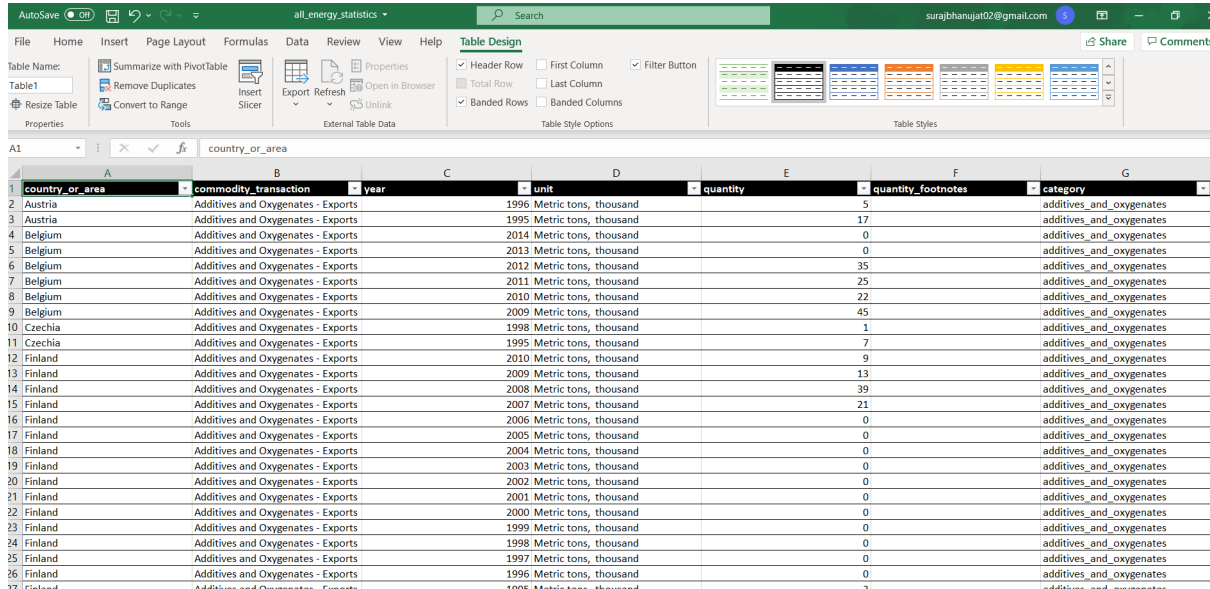
Data description of the data set of column

Data	Description
country or area	This column consist of most of the countries of the world.
commodity transac- tion	In this column there are different sectors of production and consumption according to the energy sector.
year	This column consist of the year of production it ranges from 1990 to 2014.
units	It consist of the unit of the different sectors of production.
quantity	These are the values respective to the commodity transac- tion column. There are zero values in most of the cells which means the data of that particular year is not provided.
category	In this column you are given with all the different types of energy sources.

Table 5.1: Data description

DATA VISUALIZATION AND ANALYSIS

We will be analyzing the data with the help of some questions. Below is the figure of the data sheet in excel that will give you the hint that how the data is available to us.



country_or_area	commodity_transaction	year	unit	quantity	quantity_footnotes	category
Austria	Additives and Oxygenates - Exports	1996	Metric tons, thousand	5		additives_and_oxygenates
Austria	Additives and Oxygenates - Exports	1995	Metric tons, thousand	17		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2014	Metric tons, thousand	0		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2013	Metric tons, thousand	0		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2012	Metric tons, thousand	35		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2011	Metric tons, thousand	25		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2010	Metric tons, thousand	22		additives_and_oxygenates
Belgium	Additives and Oxygenates - Exports	2009	Metric tons, thousand	45		additives_and_oxygenates
Czechia	Additives and Oxygenates - Exports	1998	Metric tons, thousand	1		additives_and_oxygenates
Czechia	Additives and Oxygenates - Exports	1995	Metric tons, thousand	7		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2010	Metric tons, thousand	9		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2009	Metric tons, thousand	13		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2008	Metric tons, thousand	39		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2007	Metric tons, thousand	21		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2006	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2005	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2004	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2003	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2002	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2001	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	2000	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	1999	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	1998	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	1997	Metric tons, thousand	0		additives_and_oxygenates
Finland	Additives and Oxygenates - Exports	1996	Metric tons, thousand	0		additives_and_oxygenates

Figure 5.1: Different Energy Production Data Sets

Analysis will be easier by giving explanation to the following set of questions.

1. What were the wind energy production trends in Germany, Spain and France from 1990 to 2014 ? Where was the maximum wind production from these 3 European countries ?

Solution: The explanation of each and every line is provided inside the program itself, A line beginning with hash tag is the explanation of that particular line of code.

```
In [ ]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns

#importing the required packages
#assigning their names as pd,np,ml,plt,sns respectively
```

Figure 5.2: Importing the required packages

```
In [ ]: sns.set_style('darkgrid')
#we are setting the grid style
#it can be darkgrid, whitegrid, dark, white, ticks
```

Figure 5.3: Setting up the grid style as 'Dark'

```
In [ ]: df = pd.read_csv("Global_energy_production.csv")
#We are giving name as df to our data frame
#we used pd.read_csv to read the csv file
df.drop("quantity_footnotes", axis=1, inplace=True)
#this above drop function is used to eliminate the unwanted column
```

Figure 5.4: Reading the csv file and drop function use

```
In [6]: df.head() #it is used to print the starting few values of data frame
df.tail() #it is used to print last few values of the data frame.
```

Out[6]:

	country_or_area	commodity_transaction	year	unit	quantity	category
1189477	Viet Nam	Electricity - total wind production	2012	Kilowatt-hours, million	92.0	wind_electricity
1189478	Viet Nam	Electricity - total wind production	2011	Kilowatt-hours, million	87.0	wind_electricity
1189479	Viet Nam	Electricity - total wind production	2010	Kilowatt-hours, million	50.0	wind_electricity
1189480	Viet Nam	Electricity - total wind production	2009	Kilowatt-hours, million	10.0	wind_electricity
1189481	Viet Nam	Electricity - total wind production	2008	Kilowatt-hours, million	1.0	wind_electricity

Figure 5.5: Use of head and tail function

```
In [8]: df_a = df.iloc[:,[1,4]] #this returns values in column 1 and 4.
df_a #use of iloc function
```

Out[8]:

	commodity_transaction	quantity
0	Additives and Oxygenates - Exports	5.0
1	Additives and Oxygenates - Exports	17.0
2	Additives and Oxygenates - Exports	0.0
3	Additives and Oxygenates - Exports	0.0
4	Additives and Oxygenates - Exports	35.0
...
1189477	Electricity - total wind production	92.0
1189478	Electricity - total wind production	87.0
1189479	Electricity - total wind production	50.0
1189480	Electricity - total wind production	10.0
1189481	Electricity - total wind production	1.0

1189482 rows × 2 columns

Figure 5.6: Use of iloc function to get specific column values

```
In [26]: Germany_eu = df[df.country_or_area.isin(["Germany"])].sort_values('year')
Spain_eu = df[df.country_or_area.isin(["Spain"])].sort_values('year')
France_eu = df[df.country_or_area.isin(["France"])].sort_values('year')
#The isin() function is used to check each element in the DataFrame is contained in values or not.
#sort_values() function sorts a data frame in Ascending or Descending order of passed Column
```

Figure 5.7: Use of isin and sort function

```
In [28]: Germany_eu_Wind = Germany_eu[Germany_eu.commodity_transaction == "Electricity - total wind production"].sort_values("year")
Spain_eu_Wind = Spain_eu[Spain_eu.commodity_transaction == "Electricity - total wind production"].sort_values("year")
France_eu_Wind = France_eu[France_eu.commodity_transaction == "Electricity - total wind production"].sort_values("year")
#finding the wind production of these countries
```

Figure 5.8: Again sorting and assigning for wind production

```
In [ ]: plt.figure(figsize=(10,5))#it decides size of figure
plt.xticks( fontsize=10)#Location where tick marks appear on x axis
plt.yticks( fontsize=10)#Location where tick marks appear on y axis
plt.plot(Germany_eu_Wind["year"],Germany_eu_Wind["quantity"],label="Germany")
plt.plot(Spain_eu_Wind["year"],Spain_eu_Wind["quantity"],label="Spain")
plt.plot(France_eu_Wind["year"],France_eu_Wind["quantity"],label="France")
#Label is used to identify the data
plt.legend(fontsize=10)
plt.ylabel("Millions of Kilowatts-Hour",fontsize=20)
#naming the y label and its font size
plt.xlabel('Year',fontsize=10)
#naming the y label and its font
plt.title('Wind production Trends in Germany, Spain and France',fontsize=15)
#giving the title to the graph
plt.xlim(1990, 2020)
#it sets the limit of the years on x axis
plt.show()
#this show() function is used to show thw grapho
```

Figure 5.9: Plotting the graph

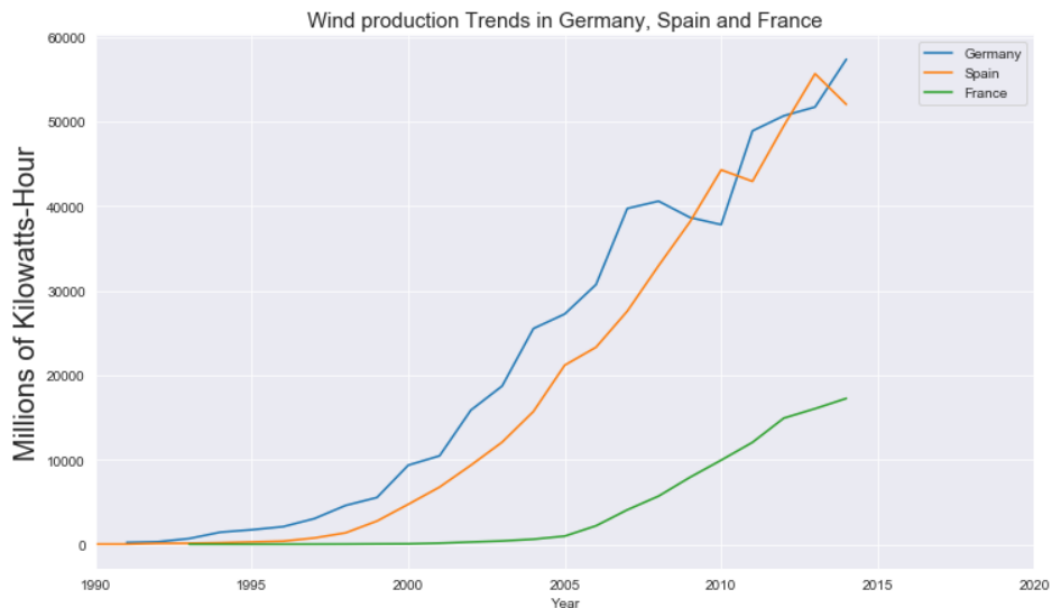


Figure 5.10: Wind energy production trends in Germany, Spain and France

Conclusion - In 2010 wind energy production of Germany was less than the wind energy production of Spain. In 2010 wind energy production of Germany was below 40000 millions of kilowatts hour where as for the same time it was more than 40000 millions of kilowatts hour for Spain. This value was 3 times more than that of France in 2010. But in 2014 Germany was having the highest wind energy production among this 3 countries with around 60000 millions of kilowatts hour.

2. Compare the wind energy production of India with respect to Germany from 1990 to 2014 ?

Solution: Explanation of the code is provided inside the program itself after each line of code.

```
In [67]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style('darkgrid')
%matplotlib inline
df = pd.read_csv("Global_energy_production.csv")
df.drop("quantity_footnotes", axis=1, inplace=True)

#again importing the packages
#dropping the unwanted column
```

Figure 5.11: Importing the packages

```
In [17]: Germany_eu = df[df.country_or_area.isin(["Germany"])].sort_values('year')
India = df[df.country_or_area.isin(["India"])].sort_values('year')
#extracting the country germany and india from data frame and sorting it.
#putting these values in germany_eu and India named variable
```

Figure 5.12: Extracting the countries India and Germany from column year

```
In [18]: Germany_eu_Wind = Germany_eu[Germany_eu.commodity_transaction == "Electricity - total wind production"].sort_values("year")
India_Wind = India[India.commodity_transaction == "Electricity - total wind production"].sort_values("year")

#extracting the commodity wind production value from data frame and sorting it.
#putting these values in germany_eu_Wind and India_Wind named variable
```

Figure 5.13: Extracting the commodity that need to be measured of both countries and assigning them in to variables

```
In [ ]: br = .5
plt.figure(figsize=(12,8))
#setting up the size of the graph
plt.barh(Germany_eu_Wind["year"],Germany_eu_Wind["quantity"],height = br , label="Germany Wind energy production")
plt.barh(India_Wind["year"] , India_Wind["quantity"] , height=br-.2, label="India Wind energy production" )
#plotting the graph and assiging the value of Germany and india.
plt.legend(fontsize=16)
plt.ylabel("Years", fontsize=14) #naming y axis
plt.xlabel("Quantity (Millions of Kilowatts-Hour)", fontsize=14) #naming x axis
plt.title("Wind energy production in the India as comapred to Germany", fontsize=16)
#Providing the title to the graph
plt.xlim(0, 73500)
#setting the limit of x axis
plt.show()
#show function is used to show the graph
```

Figure 5.14: Plotting the final bar graph

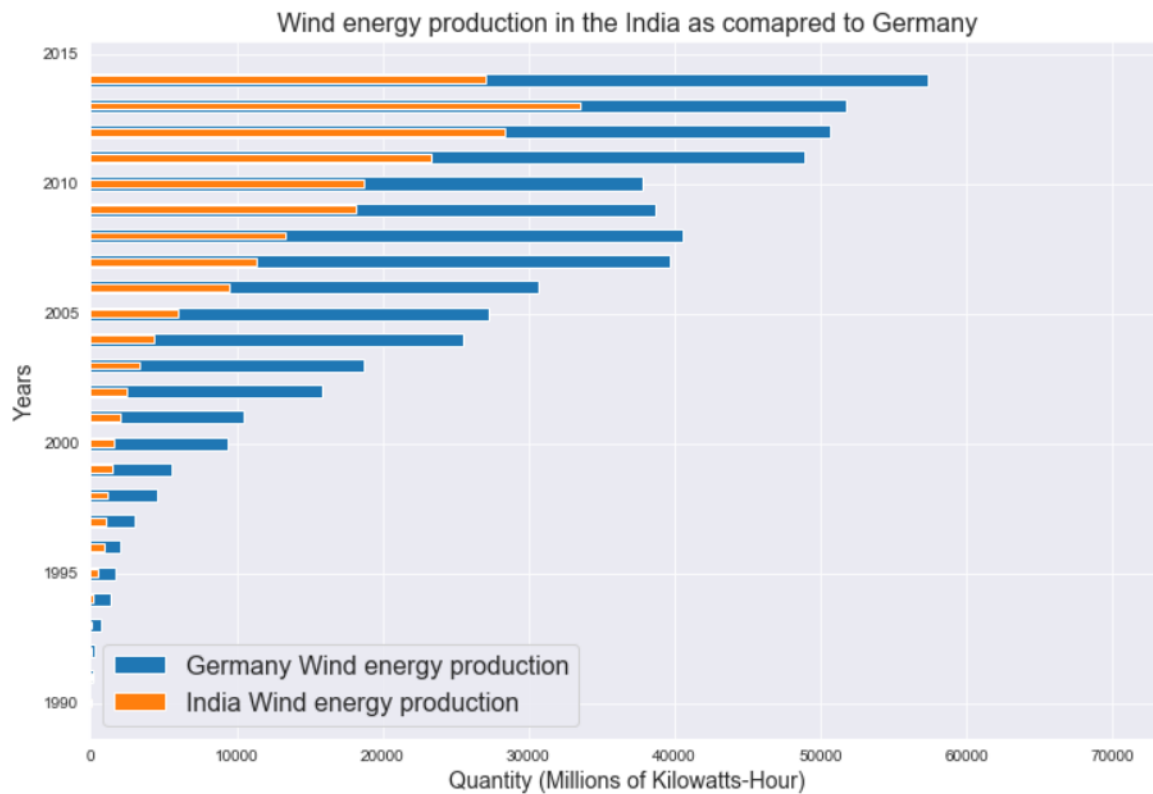


Figure 5.15: Wind energy production in India as compared to Germany

Conclusion - In the year 2014 the Germany's wind energy production was more than 55000 millions of kilowatts-hour where as in India it was around 27000 millions of kilowatts-hour. In the above graph as we can see that the wind energy production in Germany was double then India in the year 2014. Continuously from 1990 to 2014 wind energy production was always greater in Germany from India in each year. In the year 2000 it Germany's production was 5 times more than that of India.

3. Plot a regression plot for Wind energy production between the Spain and France ?

Solution: Explanation of the code is provided inside the program itself after each line of code.

```
In [6]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns #sns package is used for regression plot
sns.set_style('darkgrid')
%matplotlib inline
df = pd.read_csv("Global_energy_production.csv")
df.drop("quantity_footnotes", axis=1, inplace=True)

#again importing the packages
#dropping the unwanted column
```

Figure 5.16: Importing the seaborn package

```
In [7]: Spain = df[df.country_or_area.isin(["Spain"])].sort_values('year')
France = df[df.country_or_area.isin(["France"])].sort_values('year')

#setting up variable as spain and France and extracting from data set
```

Figure 5.17: Setting up the variable as Spain and France from dataset

```
In [8]: #we need the commodity as well of this countries so
Spain_Wind = Spain[Spain.commodity_transaction == "Electricity - total wind production"].sort_values("year")
France_Wind = France[France.commodity_transaction == "Electricity - total wind production"].sort_values("year")
```

Figure 5.18: Setting up the variable as Spain Wind and France Wind for extracting commodity values


```

In [ ]: #setting up the axes Spain
x1 = Spain_Wind.year
y1 = Spain_Wind.quantity
#setting up the axes for france
x2 = France_Wind.year
y2 = France_Wind.quantity
#size of the graph
plt.figure(figsize=(10,5))
#using the sns package and regplot function for regression plot
sns.regplot(x1,y1, order=3)
sns.regplot(x2,y2, order=3)
#Labeling the graph
plt.legend(labels=["Spain", "France"])
#setting up the title of the plot
plt.title("Regression plot for Wind energy prodction between the Spain and the France ")
#show function is used to display the plot
plt.show()

```

Figure 5.19: Code for plotting the regression graph

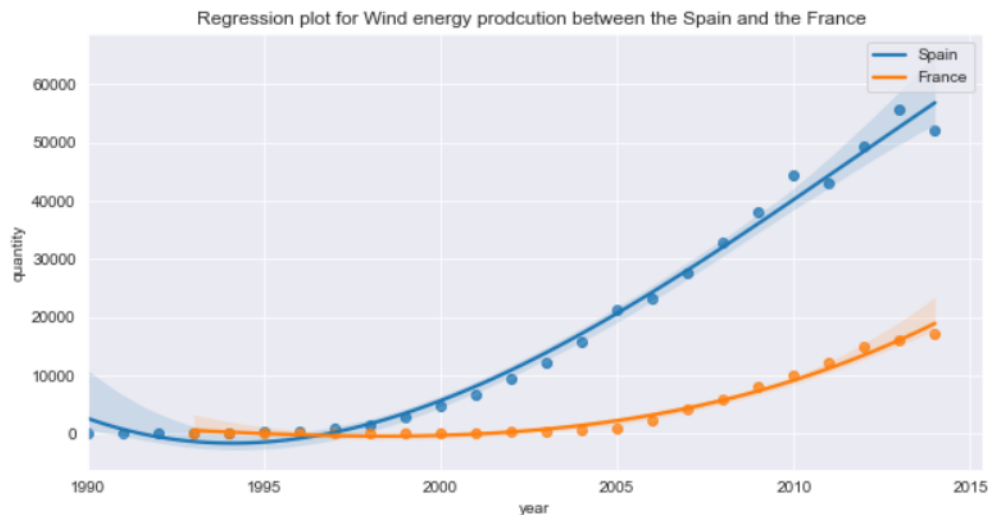


Figure 5.20: Final output of regression plot of Spain and France

Conclusion - In regression plot we normally measure that how much a particular line missed the data point from a vertical line. In the above graph it can be seen that there are multiple points on the lines of the different quantities in each year, and this points together make a regression line and that line is representing the wind energy production of two different countries that is France and Spain. It is totally clear from the graph that the wind energy production of Spain was far more as compared to the wind energy production of France.

4. What were the trends of Solar energy production and Wind energy production in USA from 1990 to 2014 ? Plot a bar graph for both energies. Which of energy production was maximum ?

Solution: Explanation of the code is provided inside the program itself after each line of code.

```
In [182]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns

df = pd.read_csv("Global_energy_production.csv")
df.drop("quantity_footnotes", axis=1, inplace=True)

#again importing the packages
#dropping the unwanted column
```

Figure 5.21: Importing the required packages for bar plot

```
In [188]: United_States = df[df.country_or_area.isin(["United States"])].sort_values('year')
#Selecting the country and assigning to the variable United_States.
#use of sort function which sort the data according to year
```

Figure 5.22: Selecting the country and assigning to the variable United States.

```
In [189]: United_States_Solar = United_States[United_States.commodity_transaction == "Electricity - total solar production"]
United_States_Wind = United_States[United_States.commodity_transaction == "Electricity - total wind production"]
#now from the commodity column we are selecting wind and solar production value
```

Figure 5.23: Selecting the country and assigning to the variable United States.

```
In [190]: United_States_Wind.iloc[:,[2,4]]
```

```
Out[190]:
```

	year	quantity
1189460	1990	3066.0
1189459	1991	3051.0
1189458	1992	2917.0
1189457	1993	3053.0
1189456	1994	3483.0
1189455	1995	3196.0
1189454	1996	3410.0
1189453	1997	3254.0
1189452	1998	3018.0
1189451	1999	4802.0
1189450	2000	5650.0
1189449	2001	6806.0
1189448	2002	10459.0
1189447	2003	11300.0
1189446	2004	14291.0
1189445	2005	17881.0
1189444	2006	26676.0
1189443	2007	34603.0
1189442	2008	55696.0
1189441	2009	74226.0
1189440	2010	95148.0
1189439	2011	120854.0
1189438	2012	141922.0
1189437	2013	169713.0
1189436	2014	183892.0

Figure 5.24: Use of iloc function and displaying the values for wind

```
In [191]: United_States_Solar.iloc[:,[2,4]]
```

```
Out[191]:
```

	year	quantity
1026691	1990	666.0
1026690	1991	782.0
1026689	1992	749.0
1026688	1993	901.0
1026687	1994	828.0
1026686	1995	828.0
1026685	1996	906.0
1026684	1997	897.0
1026683	1998	890.0
1026682	1999	689.0
1026681	2000	709.0
1026680	2001	785.0
1026679	2002	830.0
1026678	2003	848.0
1026677	2004	965.0
1026676	2005	1120.0
1026675	2006	1287.0
1026674	2007	1673.0
1026673	2008	2091.0
1026672	2009	2514.0
1026671	2010	3934.0
1026670	2011	6153.0
1026669	2012	10145.0
1026668	2013	15872.0
1026667	2014	24603.0

Figure 5.25: Use of iloc function and displaying the values for solar

```
In [ ]: bar_width = .7
plt.figure(figsize=(20,10))
#setting the size of plot
plt.xticks(fontsize=12) #setting up fontsize
plt.yticks(fontsize=14)
x = np.arange(len(United_States_Wind["year"]))
#use of numpy to arrange the values
plt.bar(x, United_States_Wind["quantity"], bar_width, color='g', capstyle='projecting', label="US Wind Production", alpha=.5)
plt.bar(x, United_States_Solar["quantity"], bar_width, color='b', label="US Solar Production", alpha=.6)
#plt.bar is used to plot bar graphs
#color=g means we green color of bar and b is for blue color
plt.legend(fontsize=16)
plt.xlabel("Years", fontsize=20)
plt.ylabel("Millions of Kilowatts-Hour", fontsize=20) #Labeling the axis
plt.title("Total Wind and Solar production in the US", fontsize=24) #providing the title
plt.xticks(x + bar_width / 6, United_States_Wind["year"])
plt.show()
```

Figure 5.26: Code for plotting the bar graph

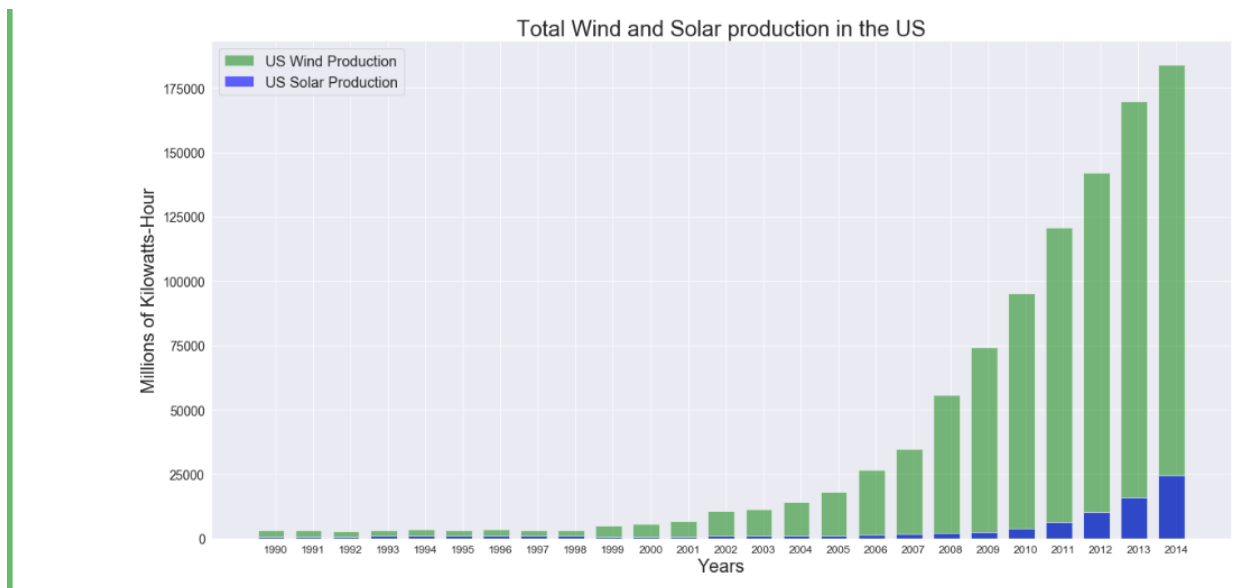


Figure 5.27: Final bar plot of wind and solar energy production in Unites States

Conclusion - According to the above bar plot we can visualize that electricity wind energy production was always higher then the electricity solar energy production. Both energy production are growing from 1990 and they have been continuously growing till date. The maximum solar energy production was 25000 millions of kilowatts-hours in the year 2014, where as in the same year the electricity from wind energy production was 175000 millions of kilowatts-hour.

Chapter 6

Ethanol production from 1981 to 2019 in USA

This data is the overview of fuel ethanol in USA from the year 1981 to 2019, the raw data consist of all the information related to the fuel ethanol in multiple units for example trillion Btu (British thermal unit), gallons and barrels as we are going to count the production and consumption of fuel ethanol in this unit.

Working on the raw data :

There are multiple columns in the data which are almost identical, the values in the column are same but having different units for example there are two columns of fuel ethanol production with one having unit in million gallons and the other column has unit in trillion btu (British thermal unit).Value are same but having different units so we will use only one column and delete the other.

Data description of the data set of column

Data	Description
Annual total	This column consist of the year of production it ranges from 1981 to 2019.
Ethanol Excluding Denaturant Feed stock Trillion Btu	This column consist of ethanol quantity without denaturant feed stock and the unit of these values are in trillion btu.
Ethanol Excluding Denaturant Losses and Co products Trillion Btu	This column consist of ethanol quantity without denaturant losses and co products and the unit of these values are in trillion btu.
Ethanol Denaturant Thousand Barrels	It consist of the ethanol denaturant and the unit of the quantity ism in thousands barrels.
Ethanol Production	There are three columns of ethanol production in the data set with three different units as trillion btu, thousand barrels and million gallons.
Ethanol Consumption	In this section as well there are three columns of ethanol consumption in the data set with three different units as trillion btu, thousand barrels and million gallons.

Table 6.1: Data description for ethanol data set

We will analyse the data by giving the answers to the following questions related to the data set.

A	B	C	D	E	F	G	H	I	J	K	L	M	N
Annual_Tot	Ethanol_Exc	Ethanol_Exc	Ethanol_Den	Ethanol_Prc	Ethanol_Prc	Ethanol_Prc	Ethanol_Net	Ethanol_Sto	Ethanol_Sto	Ethanol_Coi	Ethanol_Coi	Ethanol_Coi	Ethanol_Exc
1981	12.72	5.86	39.559	1977.96	83.074	7.049	Not Availabl	Not Availabl	Not Availabl	1977.96	83.074	7.049	6.86
1982	34.404	15.784	107.375	5368.748	225.487	19.134	Not Availabl	Not Availabl	Not Availabl	5368.748	225.487	19.134	18.62
1983	63.143	28.843	197.796	9889.799	415.372	35.247	Not Availabl	Not Availabl	Not Availabl	9889.799	415.372	35.247	34.3
1984	77.302	35.162	243.006	12150.325	510.314	43.304	Not Availabl	Not Availabl	Not Availabl	12150.325	510.314	43.304	42.14
1985	93.151	42.191	293.868	14693.416	617.123	52.367	Not Availabl	Not Availabl	Not Availabl	14693.416	617.123	52.367	50.96
1986	107.099	48.299	339.079	16953.942	712.066	60.424	Not Availabl	Not Availabl	Not Availabl	16953.942	712.066	60.424	58.8
1987	122.725	55.105	389.941	19497.033	818.875	69.487	Not Availabl	Not Availabl	Not Availabl	19497.033	818.875	69.487	67.62
1988	124.058	55.458	395.592	19779.599	830.743	70.494	Not Availabl	Not Availabl	Not Availabl	19779.599	830.743	70.494	68.6
1989	125.378	55.798	401.243	20062.164	842.611	71.502	Not Availabl	Not Availabl	Not Availabl	20062.164	842.611	71.502	69.58
1990	110.867	49.127	356.033	17801.639	747.669	63.445	Not Availabl	Not Availabl	Not Availabl	17801.639	747.669	63.445	61.74
1991	128	56.46	412.546	20627.296	866.346	73.516	Not Availabl	Not Availabl	Not Availabl	20627.296	866.346	73.516	71.54
1992	145.005	63.665	469.059	23452.953	985.024	83.586	Not Availabl	1791	Not Availabl	23452.953	985.024	83.586	81.34
1993	169.336	74.016	549.68	27484	1154.328	97.953	244	2114	323	27405	1151.01	97.671	95.047
1994	188.391	81.955	613.78	30689	1288.938	109.376	279	2393	279	30689	1288.938	109.376	106.436
1995	197.737	85.627	646.5	32325	1357.65	115.206	387	2186	-207	32919	1382.598	117.323	114.17
1996	141.284	60.897	463.56	23178	973.476	82.606	313	2065	-121	23612	991.704	84.153	81.892
1997	186.315	79.931	613.48	30674	1288.308	109.322	85	2925	860	29899	1255.758	106.56	103.696
1998	202.474	86.451	669.06	33453	1405.026	119.226	66	3406	481	33038	1387.596	117.747	114.583
1999	210.809	89.834	697.62	34881	1465.002	124.316	87	4024	618	34350	1442.7	122.423	119.133
2000	233.146	99.179	772.54	38627	1622.334	137.667	116	3400	-624	39367	1653.414	140.304	136.533
2001	253.344	107.582	840.56	42028	1765.176	149.788	315	4298	898	41445	1740.69	147.71	143.74
2002	306.762	130.036	1019.12	50956	2140.152	181.607	306	6200	1902	49360	2073.12	175.919	171.191
2003	399.556	167.976	1335.44	66772	2804.424	237.975	292	5978	-222	67286	2826.012	239.807	233.363
2004	482.102	200.975	1621.16	81058	3404.436	288.891	3542	6002	24	84576	3552.192	301.429	293.328
2005	549.526	227.117	1859.22	92961	3904.362	331.313	3234	5563	-439	96634	4058.628	344.404	335.148
2006	683.239	279.906	2325.88	116294	4884.348	414.472	17408	8760	3197	130505	5481.21	465.12	452.62
2007	906.708	368.222	3105.26	155263	6521.046	553.357	10457	10535	1775	163945	6885.69	584.3	568.597
2008	1286.284	517.598	4432.74	221637	9308.754	789.914	12610	14226	3691.001	230556	9683.352	821.702	799.619

Figure 6.1: Overview of the raw data of the fuel Ethanol

1. What were the ethanol production and ethanol consumption trends in USA from 1981 to 2019 ?

Solution: The explanation of each and every line is provided inside the program itself, A line beginning with hash tag is the explanation of that particular line of code.

```
In [14]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns
df = pd.read_csv("fuel_ethanol.csv")
#reading the csv file from the system
df.head()
#displaying the first five rows of data set

Out[14]:
```

	Annual_Total	Ethanol_Excluding_Denaturant_Feedstock_Trillion_Btu	Ethanol_Excluding_Denaturant_Los
0	1981		12.720
1	1982		34.404
2	1983		63.143
3	1984		77.302
4	1985		93.151

Figure 6.2: Importing packages and displaying some rows of data set


```
In [18]: plt.figure(figsize=(12,7))#deciding the plot size

plt.plot( df.Annual_Total, df.Ethanol_Production_Million_Gallons,label='Ethanol_Production_Million_Gallons')

plt.plot(df.Annual_Total, df.Ethanol_Consumption_Million_Gallons, label='Ethanol_Consumption_Million_Gallons')

plt.scatter( df.Annual_Total, df.Ethanol_Production_Million_Gallons)

plt.scatter(df.Annual_Total, df.Ethanol_Consumption_Million_Gallons)
#combining the line plot and the scatter plot
#we use plt.scatter for scatter plot and plt.plot is for line plot

plt.title('Ethanol production and consumption trends in USA from 1981 to 2019',fontsize=15)
#title function is used name the graph and fontsize is used to for increasing font

plt.xlabel('Year',fontsize=13) #Labeling x axis
plt.ylabel('Million Gallons of Ethanol',fontsize=13) #Labeling y axis

plt.legend() #Legend function is used to for labelling

plt.show() #show function is used to display the plot
```

Figure 6.3: Code with explanation for plotting the graph

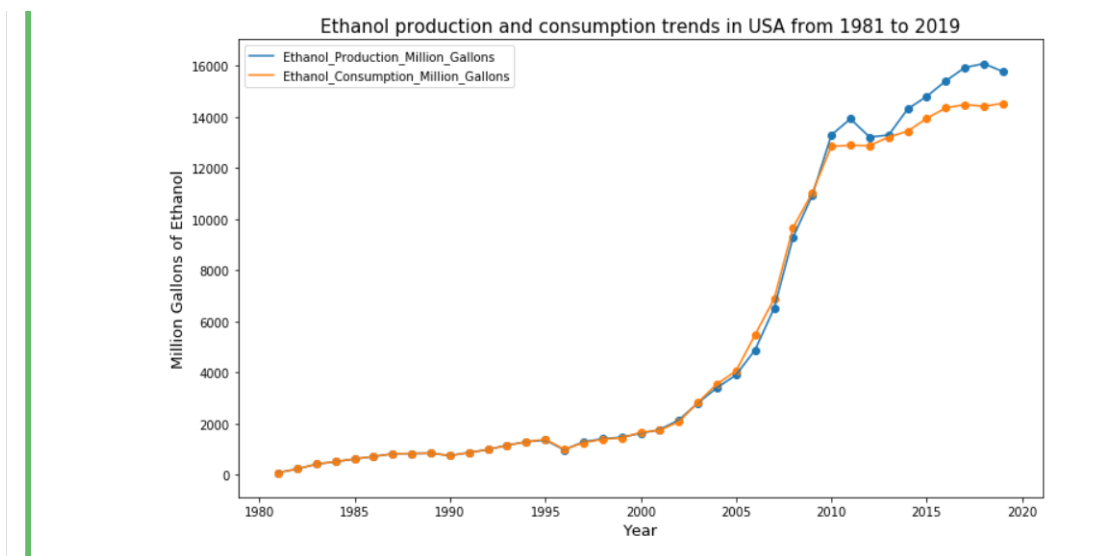


Figure 6.4: Line graph explaining the trend of production and consumption of Ethanol

Conclusion - As you can analyse from the graph that there has been consistent increase in the production as well as the consumption in the fuel ethanol from 1981. Consumption was approximately same in the year 2009, 2010 and 2011, which was around 13000 million gallons. If we talk about the production of ethanol there has been steep fall in the year 2010, but after that year production of ethanol was uniform again. The maximum production of ethanol was 16000 million gallons of ethanol in USA.

2. Which top five years were having the maximum Ethanol Production in Million Gallons ?
Plot a pie chart for it ?

Solution: Below are figures with codes inside explaining the maximum ethanol production in Million Gallons.

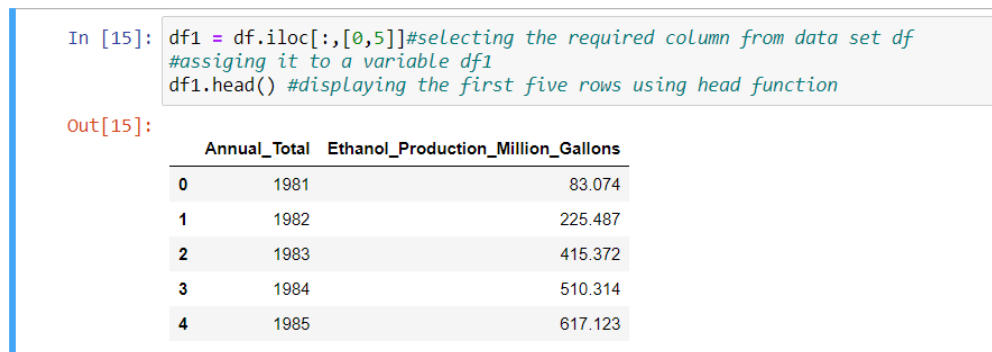


Figure 6.5: Use of iloc function for selecting required columns

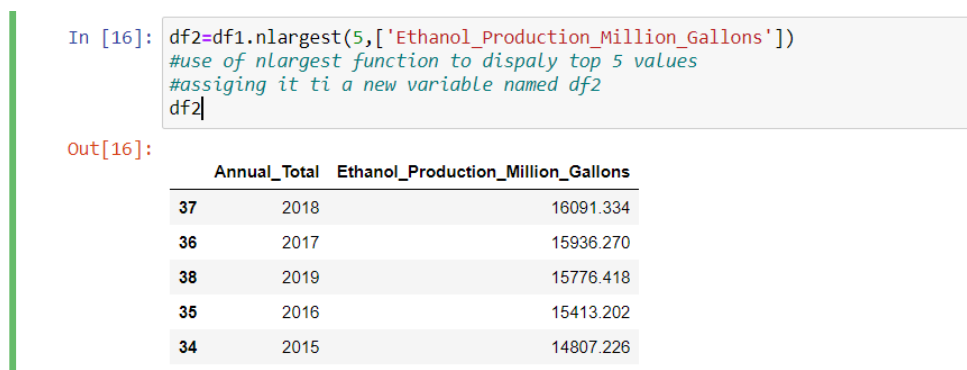


Figure 6.6: Selection of top 5 largest quantity of ethanol production

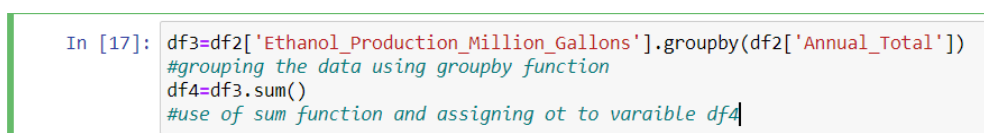


Figure 6.7: Use of groupby function

```

In [22]: values=df4.values
labels=df4.index
plt.figure(figsize=(18,8))
#selecting the size of the pie chart
plt.pie(values,labels=labels,autopct='%1.1f%%')
#selecting the auto percentage and use of plt.pie to plot pie chart
plt.title('Top 5 years with highest ethanol production', fontsize=15)
#use of title function to name the pie chart
#deciding the font size of the title after title

```

```

Out[22]: Text(0.5, 1.0, 'Top 5 years with highest ethanol production')

```

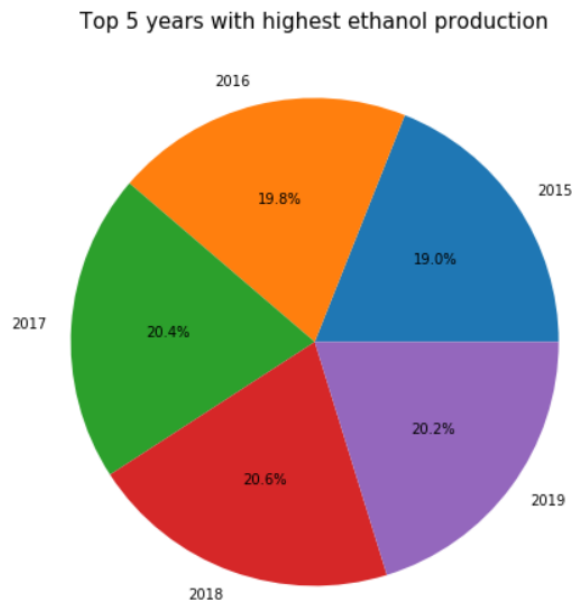


Figure 6.8: Plotting the pie chart

Conclusion - This pie chart can be easily visualized easily, as it describes the top five largest quantity of production of ethanol from the year 1981 to 2019. In the highest production of ethanol was in the year 2018 according to the pie chart. If we consider the top five years of production of ethanol then the maximum production was in the year 2018 which was maximum that is 20.6% of the five years. There was not much difference between the year 2017 and 2019 as the difference was only of 0.2%. We can visualize from the pie chart that there was decrease of 0.4% in the production of ethanol from the previous year in the year 2019.

3. What was the best month of the production of B100 bio diesel in the year 2019 and what was the maximum production in million gallons ?

Solution: Below are figures with codes inside explaining the maximum B100 bio diesel production in Million Gallons.

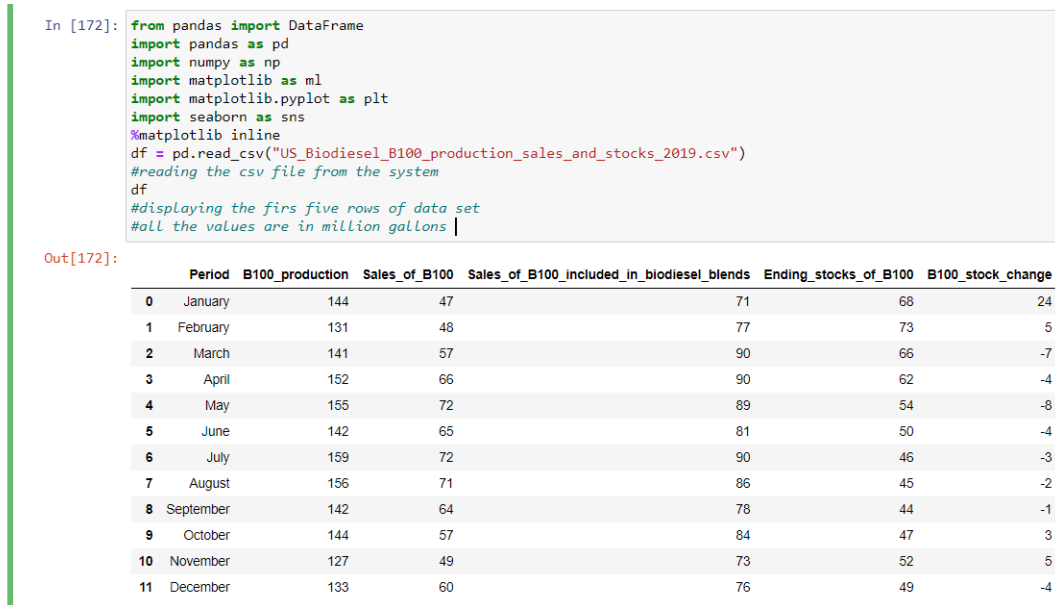


Figure 6.9: Importing the packages and displaying the head of the data

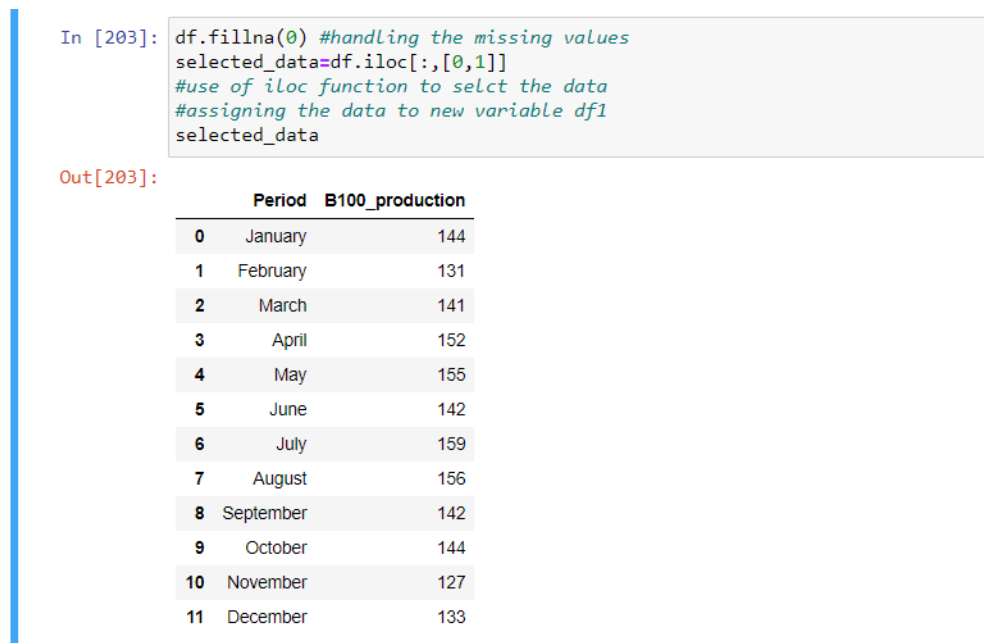


Figure 6.10: Choosing the desired column for our plot with use of iloc function

```
In [ ]: months = selected_data.Period
#selecting the column and assigning to the variable months
production_of_B100 = selected_data.B100_production
#selecting the column and assigning to the variable production_of_B100
sns.set_style('darkgrid')
#selecting the grid style
plt.figure(figsize=(7,7))
#selecting the plot size
plt.barh(months,production_of_B100,align='center',height=0.3, color='purple')
#use of plt.barh for plotting horizontal bar plot
#selecting the alignment,height and color of the bars
plt.xlabel('Mllion gallons of B100 biodiesel',fontsize=16)
plt.ylabel('Months',fontsize=16)
#naming the axes and deciding their fontsize
plt.title('US Biodiesel production in each month of 2019',fontsize=20)
#naming the title of the plot
plt.show()
#use of show function to display the plot
```

Figure 6.11: Code for plotting the bar plot

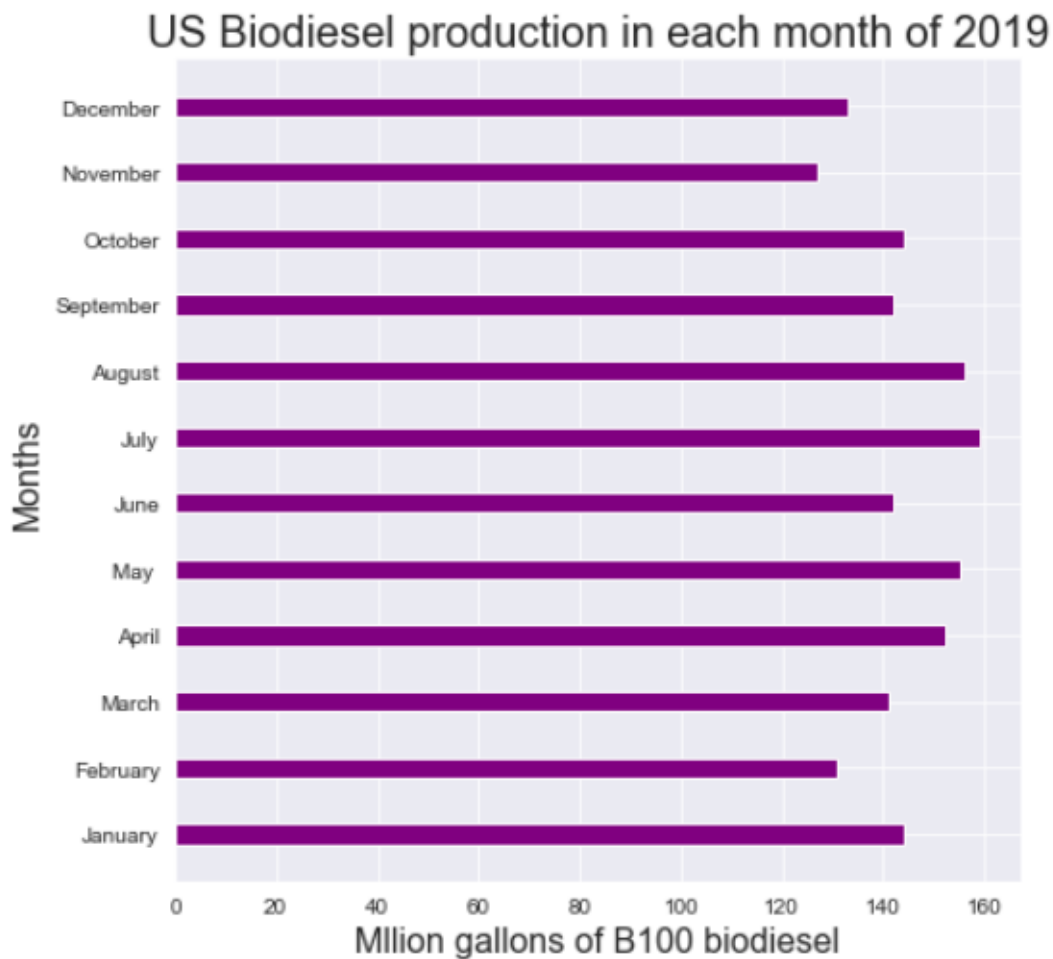


Figure 6.12: Final bar plot of B100 Bio diesel production in the year 2019

Conclusion - From the above graph we have analyzed and answered our question that in the month of July there was highest production of B100 bio diesel. If we talk about the quantity then the production quantity of bio diesel was 159 million gallons. The above graph is plotted for the country USA. If we visualize our graph then we can also draw a result that there consecutive fall in the production of B100 bio diesel after it has been reached to the maximum value in the month of July. The lowest production of bio diesel was in the month of November which was around 127 million gallons.

4. What were the trends of ethanol for Ethanol Excluding Denaturant Feed stock Trillion Btu, Ethanol Excluding Denaturant Losses and Co products Trillion Btu, Ethanol Production Trillion Btu and Ethanol Consumption Trillion Btu ? Plot a stack plot for all the values ?

Solution:Below are figures with codes inside explaining the ethanol trends in trillion btu.

```
In [103]: from pandas import DataFrame
import pandas as pd
import numpy as np
import matplotlib as ml
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
df = pd.read_csv("fuel_ethanol.csv")
#reading the csv file from the system
df.head(7)

#displaying the first five rows of data set
#all the values are in million gallons
```

Out[103]:

	Annual_Total	Ethanol_Excluding_Denaturant_Feedstock_Trillion_Btu	Ethanol_Excluding_Denaturant_Losses_and_Coproducts_Trillion_Btu
0	1981	12.720	5.860
1	1982	34.404	15.784
2	1983	63.143	28.843
3	1984	77.302	35.162
4	1985	93.151	42.191
5	1986	107.099	48.299
6	1987	122.725	55.105

Figure 6.13: importing the packages and displaying first 7 rows

```
In [102]: ethanol = [(df.Ethanol_Excluding_Denaturant_Feedstock_Trillion_Btu),
(df.Ethanol_Excluding_Denaturant_Losses_and_Coproducts_Trillion_Btu),
(df.Ethanol_Production_Trillion_Btu)
,(df.Ethanol_Consumption_Trillion_Btu)]

#selecting the required columns and assigning to a variable ethanol
#selecting the required columns and assigning to a variable ethanol

year=df.Annual_Total
```

Figure 6.14: selecting the required column and assigning to the variable

```

In [ ]: labels = ["Excluding_Denaturant_Feedstock", "Excluding_Denaturant_Losses_and_Coproducts",
                 "Production",
                 "Consumption"]
#displaying the labels of the stack plots

fig, ax = plt.subplots()

ax.stackplot(year, ethanol, labels=labels, colors=['m', 'c', 'r', 'k'])
#using stackplot function to plot the stack plot
#displaying the color for all the values of columns

ax.legend(loc='upper left')
#setting up the location of the plot

plt.plot([],[], color='m', label='Excluding_Denaturant_Feedstock', linewidth=30)
plt.plot([],[], color='c', label='Excluding_Denaturant_Losses_and_Coproducts', linewidth=30)
plt.plot([],[], color='r', label='Production', linewidth=30)
plt.plot([],[], color='k', label='Consumption', linewidth=30)

#setting up the width of each label

ax.set_title('Ethanol Bio diesel overview')
#naming the title of the plot
ax.set_xlabel('Year')
#setting up the x label of the plot
ax.set_ylabel('Ethanol in trillion btu')
#setting up the y label of the plot

plt.show()
#use of show function to plot the graph

```

Figure 6.15: Code for plotting the stack plot

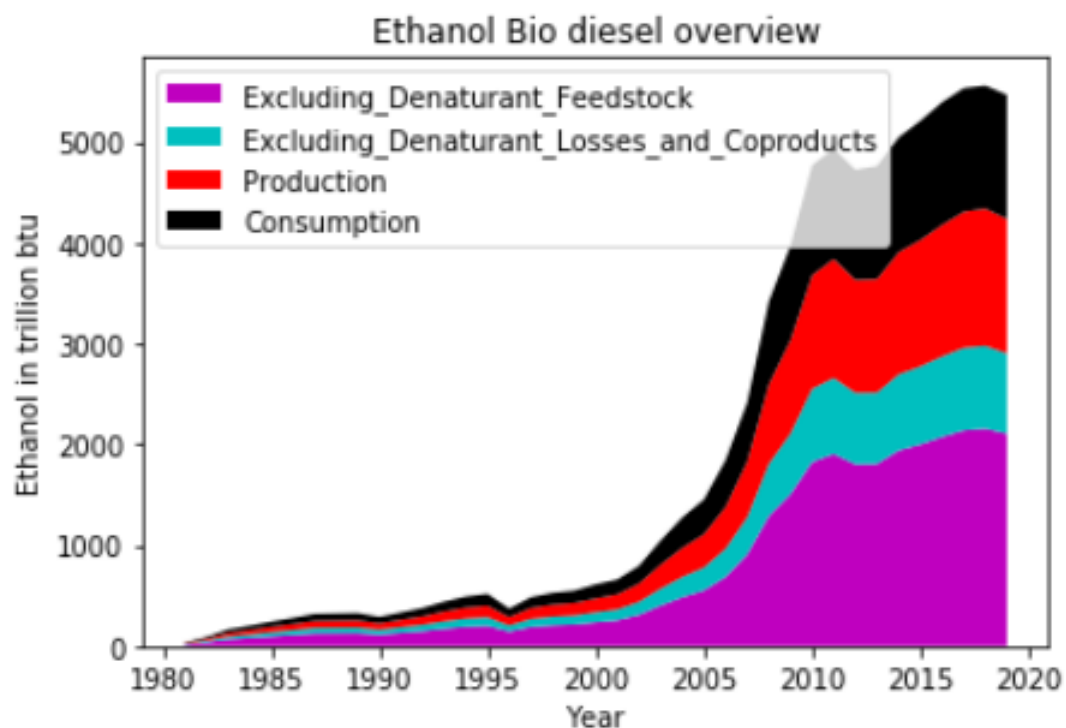


Figure 6.16: final stack plot of ethanol trends

Conclusion - As we can visualize from the above plot that the ethanol consumption was maximum which was more than 5000 trillion btu in the year 2019. The graph for each Ethanol Excluding Denaturant Feedstock Trillion Btu, Ethanol Excluding Denaturant Losses and Coproducts Trillion Btu, Ethanol Production Trillion Btu and Ethanol Consumption Trillion Btu has been consecutively increased from the year 1981 to the year 2019. The values of ethanol production was started from 1000 trillion btu ethanol and it has reached to around more than 5000 trillion btu ethanol till the year 2019.

Bibliography

- [1] Data science https://en.wikipedia.org/wiki/Data_science Accessed on 27-06-2020.
- [2] A book on data science by Dr. Ossama Embarak, https://www.academia.edu/37886932/Data_Analysis_and_Visualization_Using_Python_-_Dr._Ossama_Embarak.pdf Accessed on 27-06-2020.
- [3] A blog on quora
<https://www.quora.com> Accessed on 27-06-2020.
- [4] Smart data collective site
<https://www.smartdatacollective.com> Accessed on 28-06-2020
- [5] A blog on by Grenoble School of business
<https://www.stoodnt.com/index.php/blog/> Accessed on 28-06-2020.
- [6] Python website <https://www.python.org/doc/essays/blurb/> Accessed on 29-06-2020.
- [7] Data camp tutorial
<https://www.datacamp.com/community/tutorials/data-structures-python#adt> Accessed on 29-06-2020
- [8] Automate the Boring Stuff with Python Practical Programming for total Beginners (Author AL SWEIGART) Accessed on 29-06-2020
- [9] <https://wiki.python.org/> Accessed on 03-07-2020
- [10] https://www.python-course.eu/python3_packages.php Accessed on 03-07-2020
- [11] Matplotlib <https://en.wikipedia.org/wiki/Matplotlib> Accessed on 04-07-2020
- [12] Numpy online <https://en.wikipedia.org/wiki/NumPy> Accessed on 07-07-2020

List of Figures

1.1	7 steps that together constitute this life-cycle model of Data science[3]	5
1.2	Role of Data Science and Big Data Analytics in the Renewable Energy Sector [5]	7
2.1	Array example	10
2.2	list example	10
2.3	tuple example	11
2.4	dictionary example	11
2.5	A data structure tree at glance	12
2.6	if else statement	17
2.7	elif example	19
2.8	for example with range statement	20
2.9	While loop example	21
2.10	packages example [10]	22
2.11	function example	23
3.1	Matplotlib basic example	25
3.2	series and data frame in pandas	27
3.3	NumPy basic example	28
5.1	Different Energy Production Data Sets	33
5.2	Importing the required packages	33
5.3	Setting up the grid style as 'Dark'	33
5.4	Reading the csv file and drop function use	34
5.5	Use of head and tail function	34
5.6	Use of iloc function to get specific column values	34
5.7	Use of isin and sort function	34
5.8	Again sorting and assigning for wind production	35
5.9	Plotting the graph	35
5.10	Wind energy production trends in Germany, Spain and France	35
5.11	Importing the packages	36
5.12	Extracting the countries India and Germany from column year	36

5.13	Extracting the commodity that need to be measured of both countries and as- signing them in to variables	36
5.14	Plotting the final bar graph	36
5.15	Wind energy production in India as compared to Germany	37
5.16	Importing the seaborn package	38
5.17	Setting up the variable as Spain and France from dataset	38
5.18	Setting up the variable as Spain Wind and France Wind for extracting commod- ity values	38
5.19	Code for plotting the regression graph	39
5.20	Final output of regression plot of Spain and France	39
5.21	Importing the required packages for bar plot	40
5.22	Selecting the country and assigning to the variable United States.	40
5.23	Selecting the country and assigning to the variable United States.	40
5.24	Use of iloc function and displaying the values for wind	41
5.25	Use of iloc function and displaying the values for solar	42
5.26	Code for plotting the bar graph	43
5.27	Final bar plot of wind and solar energy production in Unites States	43
6.1	Overview of the raw data of the fuel Ethanol	46
6.2	Importing packages and displaying some rows of data set	46
6.3	Code with explanation for plotting the graph	47
6.4	Line graph explaining the trend of production and consumption of Ethanol . . .	47
6.5	Use of iloc function for selecting required columns	48
6.6	Selection of top 5 largest quantity of ethanol production	48
6.7	Use of groupby function	48
6.8	Plotting the pie chart	49
6.9	Importing the packages and displaying the head of the data	50
6.10	Choosing the desired column for our plot with use of iloc function	50
6.11	Code for plotting the bar plot	51
6.12	Final bar plot of B100 Bio diesel production in the year 2019	51
6.13	importing the packages and displaying first 7 rows	53
6.14	selecting thee required column and assigning to the variable	53
6.15	Code for plotting the stack plot	54
6.16	final stack plot of ethanol trends	54

List of Tables

2.1	Arithmetic operators	13
2.2	Assignment Operators	14
2.3	Logical Operators	15
2.4	Comparison operators	15
2.5	Membership operators	16
5.1	Data description	32
6.1	Data description for ethanol data set	45