OASYS INSTITUTE OF TECHNOLOGY
PULIVALAM
TRICHY

FDS lab manual

Fundamentals of data science (Anna University)

CS 3352

FOUNDATION OF DATA SCIENCE LABORATORY

(LABORATORY MANUAL)

(R-2021)

III SEMESTER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Prepared by: J SETHURAMAN (AP/CSE)

COURSE OBJECTIVES:

- To understand the python libraries for data science
- To understand the basic Statistical and Probability measures for data science.
- To learn descriptive analytics on the benchmark data sets.
- To apply correlation and regression analytics on standard data sets.
- To present and interpret data using visualization packages in Python.

LIST OF EXPERIMENTS:

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.
2. Working with Numpy arrays
3. Working with Pandas data frames
4. Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.
5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:
   a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.
   b. Bivariate analysis: Linear and logistic regression modeling
   c. Multiple Regression analysis
   d. Also compare the results of the above analysis for the two data sets.
6. Apply and explore various plotting functions on UCI data sets.
   a. Normal curves
   b. Density and contour plots
   c. Correlation and scatter plots
   d. Histograms
   e. Three dimensional plotting.
7. Visualizing Geographic Data with Basemap

COURSE OUTCOMES(COs):

At the end of this course, our students will be able to:

CO1: Make use of the python libraries for data science

CO2: Make use of the basic Statistical and Probability measures for data science.

CO3: Perform descriptive analytics on the benchmark data sets.

CO4: Perform correlation and regression analytics on standard data sets

CO5: Present and interpret data using visualization packages in Python

INDEX

S.NO	DATE	LIST OF EXPERIMENTS	PAGE NO	STAFF SIGN
1.		Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages.		
2.		Working with Numpy arrays		
3.		Working with Pandas data frames		
4.		Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.		
5.		Applications using TCP sockets like: Chat		
4		Simulation of DNS using UDP sockets.		
5.		Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following: a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis. b. Bivariate analysis: Linear and logistic regression modeling c. Multiple Regression analysis d. Also compare the results of the above analysis for the two data sets		
6.		6. Apply and explore various plotting functions on UCI data sets. a. Normal curves b. Density and contour plots c. Correlation and scatter plots d. Histograms e. Three dimensional plotting.		
7.		Visualizing Geographic Data with Basemap		

1. Download, install and explore the features of NumPy, SciPy, Jupyter, Statsmodels and Pandas packages

Python is a high-level and general-purpose programming language with data science and machine learning packages. Use the video below to install on Windows, MacOS, or Linux. As a first step, **install Python for Windows, MacOS, or Linux**.

Install Python Packages

The power of Python is in the packages that are available either through the **pip or conda package managers**. This page is an overview of some of the best packages for machine learning and data science and how to install them.

We will explore the Python packages that are commonly used for data science and machine learning. You may need to install the packages from the terminal, Anaconda prompt, command prompt, or from the Jupyter Notebook. If you have multiple versions of Python or have specific dependencies then use an environment manager such as **pyenv**. For most users, a single installation is typically sufficient. The Python package manager **pip** has all of the packages (such as **NumPy, SciPy**) that we need for this course. If there is an administrative access error, install to the local profile with the **--user** flag.

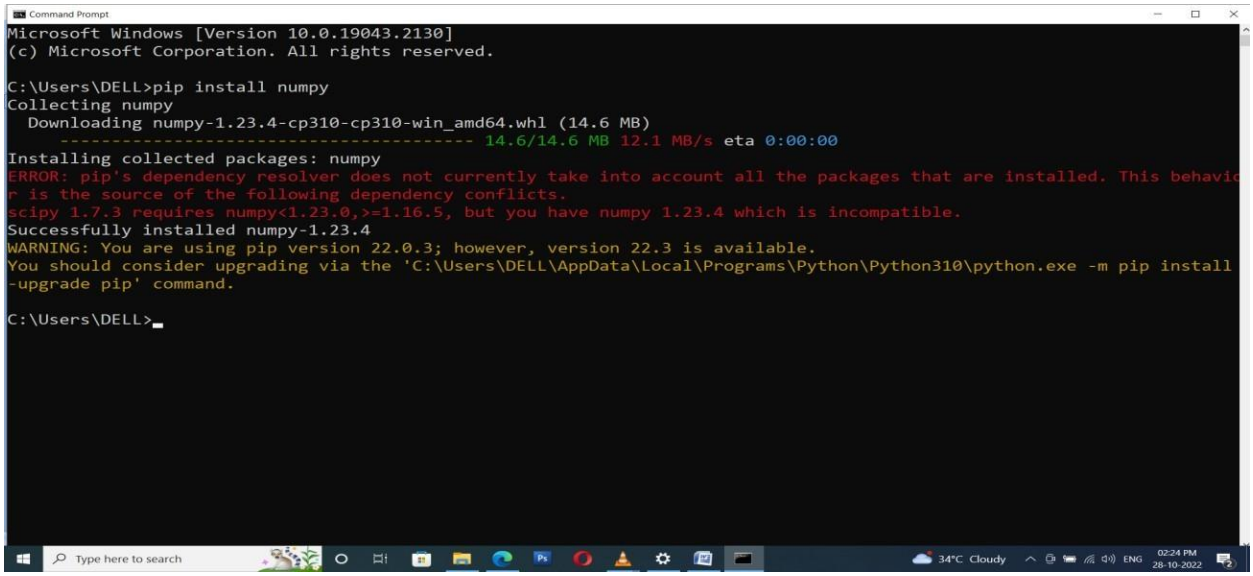
Install Method

Numpy

Numpy is a numerical computing package for mathematics, science, and engineering. Many data science packages use Numpy as a dependency.

Ex : pip install NumPy

Output:

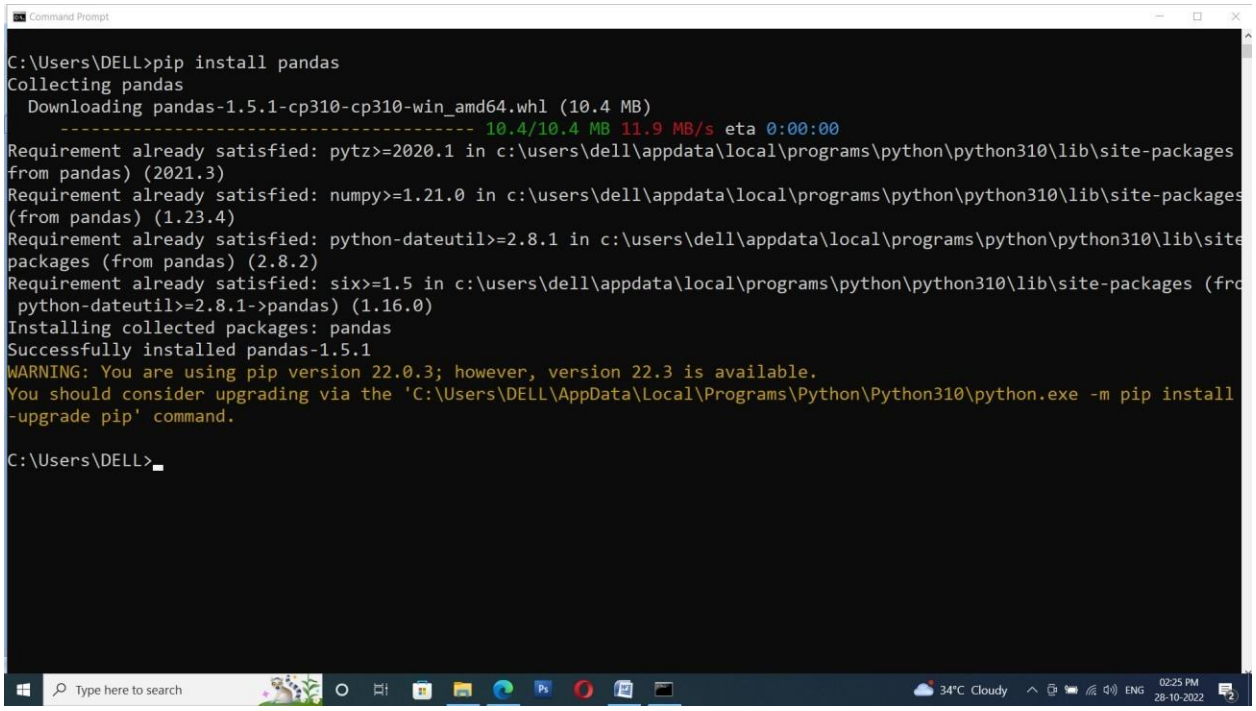


PANDAS

Pandas visualizes and manipulates data tables. There are many functions that allow efficient manipulation for the preliminary steps of data analysis problems.

Ex: pip install pandas

Output:

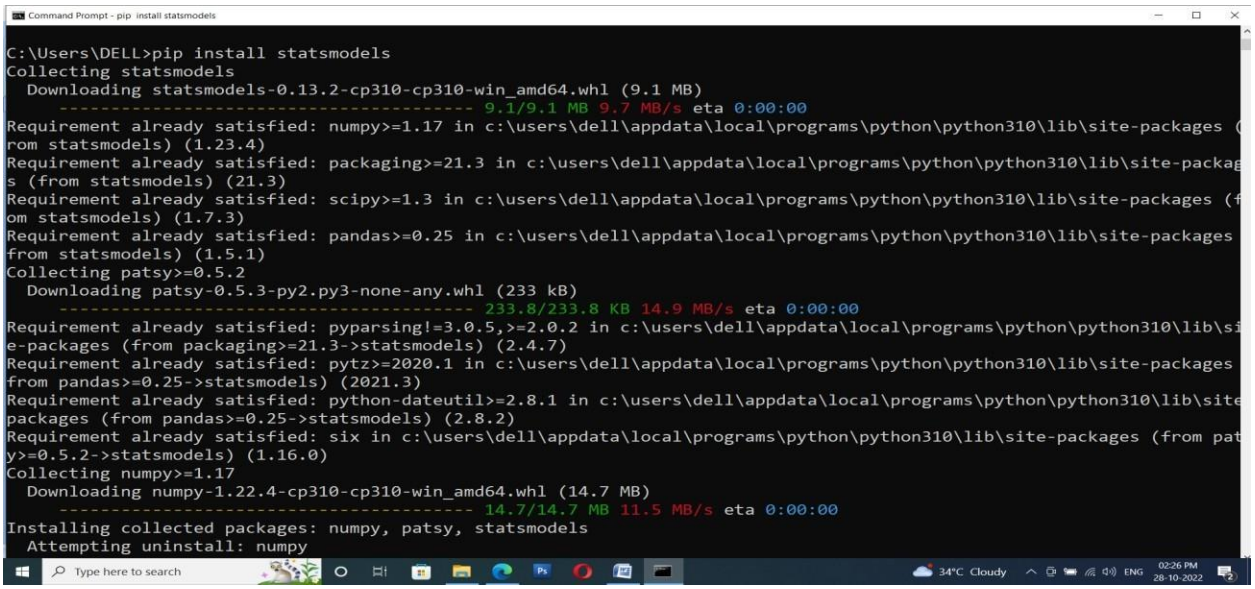


Statsmodel

Statsmodels is a package for exploring data, estimating statistical models, and performing statistical tests. It include descriptive statistics, statistical tests, plotting functions, and result statistics.

Ex: pip install statsmodels

Output

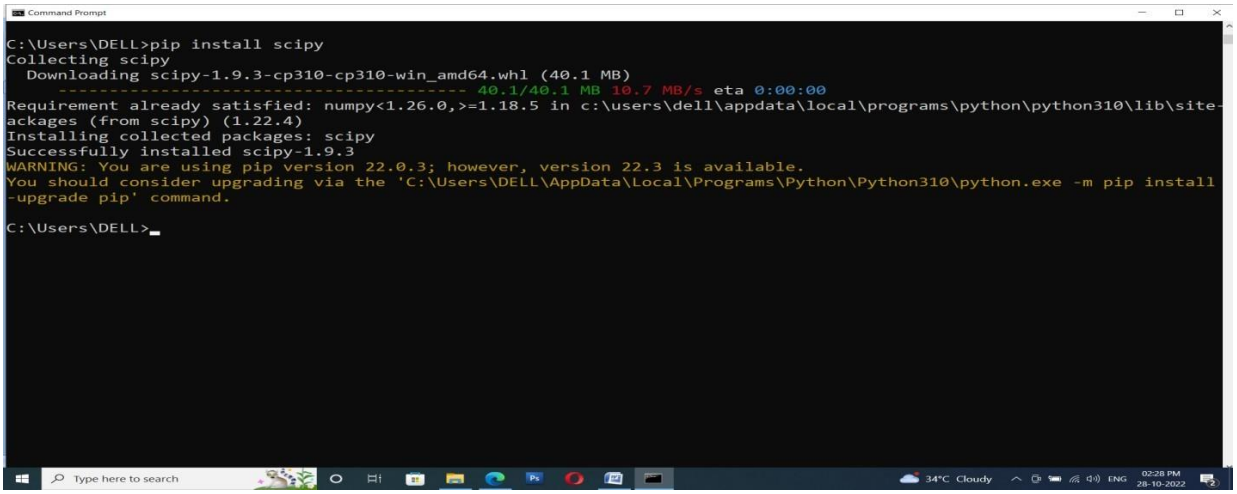


Scipy:

SciPy is a general-purpose package for mathematics, science, and engineering and extends the base capabilities of NumPy.

Ex: pip install scipy

Output:



2. Working with Numpy arrays

NumPy stands for Numerical Python. It is a Python library used for working with an array. In Python, we use the list for purpose of the array but it's slow to process. NumPy array is a powerful N-dimensional array object and its use in linear algebra, Fourier transforms, and random number capabilities. It provides an array object much faster than traditional Python lists.

Types of Array:

1. One Dimensional Array
2. Multi-Dimensional Array

3. One Dimensional Array:

4. A one-dimensional array is a type of linear array.

5.

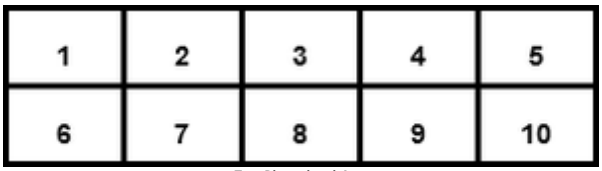
6. *One Dimensional Array*

Example Code:

```
# importing numpy

module import numpy as

np

# creating list

list = [1, 2, 3,

4]

# creating numpy array

sample_array =

np.array(list1) print("List

in python : ", list)


print("Numpy Array in python :", sample_array)
```

Multi-Dimensional Array:

Data in multidimensional arrays are stored in tabular form.



Two Dimensional Array

Example:

```
# importing numpy

module import numpy as

np

# creating list

list_1 = [1, 2, 3,

4]

list_2 = [5, 6, 7, 8]

list_3 = [9, 10, 11, 12]

# creating numpy array

sample_array = np.array([list_1, list_2, list_3])

print("Numpy multi dimensional array in python\n", sample_array)
```

Output:

```
Numpy multi dimensional array in python
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]]
```

3. Working with Pandas data frames

Pandas DataFrame is two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the **data**, **rows**, and **columns**.

Creating a Pandas DataFrame

In the real world, a Pandas DataFrame will be created by loading the datasets from existing storage, storage can be SQL Database, CSV file, and Excel file. Pandas DataFrame can be created from the lists, dictionary, and from a list of dictionary etc. Dataframe can be created in different ways here are some ways by which we create a dataframe:

Creating a data frame using List: DataFrame can be created using a single list or a list of lists.

Code:

```
import pandas as

pd import numpy

as np

sas=pd.Series([1,3,5,np.nan,

6]) sas
```

Dealing with Rows and Columns

A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. We can perform basic operations on rows/columns like selecting, deleting, adding, and renaming.

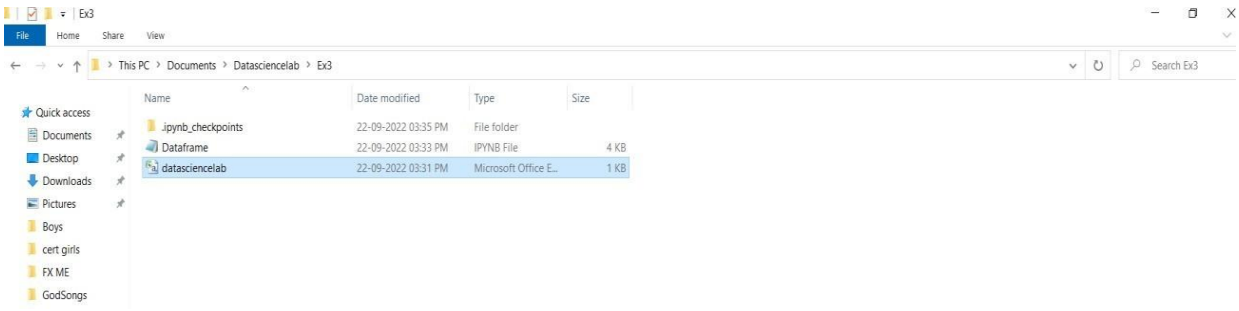
Code:

```
import pandas as

pd data={'apple':

[3,2,0],

    'orange' : [3,8,9]}

purchase=pd.DataFrame(da

ta) purchase
```

Out[11]:

	apple	orange
0	3	3
1	2	8
2	0	9

```
purchase.to_csv('datasciencelab.csv')
```



4. Reading data from text files, Excel and the web and exploring various commands for doing descriptive analytics on the Iris data set.

What is Exploratory Data Analysis?

Exploratory Data Analysis (EDA) is a technique to analyze data using some visual Techniques. With this technique, we can get detailed information about the statistical summary of the data. We will also be able to deal with the duplicates values, outliers, and also see some trends or patterns present in the dataset.

Now let's see a brief about the Iris dataset.

Iris Dataset

If you are from a data science background you all must be familiar with the Iris Dataset. If you are not then don't worry we will discuss this here.

Iris Dataset is considered as the Hello World for data science. It contains five columns namely – Petal Length, Petal Width, Sepal Length, Sepal Width, and Species Type. Iris is a flowering plant, the researchers have measured various features of the different iris flowers and recorded them digitally.

Note: This dataset can be downloaded from www.kaggle.com.

You can download the Iris.csv file from the above link. Now we will use the Pandas library to load this CSV file, and we will convert it into the dataframe. read_csv() method is used to read CSV files.

For Code:

```
import pandas as pd

data1=pd.read_csv("Iris.csv")

data1.head()
```

Out[23]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
0	1	5.1	3.5	1.4	0.2	Iris-setosa
1	2	4.9	3.0	1.4	0.2	Iris-setosa
2	3	4.7	3.2	1.3	0.2	Iris-setosa
3	4	4.6	3.1	1.5	0.2	Iris-setosa
4	5	5.0	3.6	1.4	0.2	Iris-setosa

data1.info()

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
dtypes: float64(4), int64(1), object(1)
memory usage: 7.2+ KB
```

data1.describe()

Out[6]:

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm
count	150.000000	150.000000	150.000000	150.000000	150.000000
mean	75.500000	5.843333	3.054000	3.758667	1.198667
std	43.445368	0.828066	0.433594	1.764420	0.763161
min	1.000000	4.300000	2.000000	1.000000	0.100000
25%	38.250000	5.100000	2.800000	1.600000	0.300000
50%	75.500000	5.800000	3.000000	4.350000	1.300000
75%	112.750000	6.400000	3.300000	5.100000	1.800000
max	150.000000	7.900000	4.400000	6.900000	2.500000

data1.isnull().sum()

Out[7]:

```
Id               0
SepalLengthCm    0
SepalWidthCm     0
PetalLengthCm    0
PetalWidthCm     0
Species          0
dtype: int64
```

data1.shape

```
Out[27]: (150, 6)
```

```
data = data1.drop_duplicates(subset ="Species",)
data
```

5. Use the diabetes data set from UCI and Pima Indians Diabetes data set for performing the following:

This dataset is originally from the National Institute of Diabetes and Digestive and Kidney Diseases. The objective of the dataset is to diagnostically predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset. Several constraints were placed on the selection of these instances from a larger database. In particular, all patients here are females at least 21 years old of Pima Indian heritage. The datasets consists of several medical predictor variables and one target variable, Outcome. Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

The datasets consist of several medical predictor (independent) variables and one target (dependent) variable, Outcome. Independent variables include the number of pregnancies the patient has had, their BMI, insulin level, age, and so on.

a. Univariate analysis: Frequency, Mean, Median, Mode, Variance, Standard Deviation, Skewness and Kurtosis.

Descriptive Statistics is the building block of data science. Advanced analytics is often incomplete without analyzing descriptive statistics of the key metrics. In simple terms, descriptive statistics can be defined as the measures that summarize a given data, and these measures can be broken down further into the measures of central tendency and the measures of dispersion.

Measures of central tendency include mean, median, and the mode, while the measures of variability include standard deviation, variance, and the interquartile range. In this guide, you will learn how to compute these measures of descriptive statistics and use them to interpret the data.

We will cover the topics given below:

Mean

Median

Mode

Standard Deviation

Variance

Interquartile Range

Skewness

Data

In this guide, we will be using fictitious data of loan applicants containing 600 observations and 10 variables, as described below:

1. Marital_status: Whether the applicant is married ("Yes") or not ("No").
2. Dependents: Number of dependents of the applicant.
3. Is_graduate: Whether the applicant is a graduate ("Yes") or not ("No").
4. Income: Annual Income of the applicant (in USD).
5. Loan_amount: Loan amount (in USD) for which the application was submitted.
6. Term_months: Tenure of the loan (in months).
7. Credit_score: Whether the applicant's credit score was good ("Satisfactory") or not ("Not_satisfactory").
8. Age: The applicant's age in years.
9. Sex: Whether the applicant is female (F) or male (M).
10. approval_status: Whether the loan application was approved ("Yes") or not ("No").

Let's start by loading the required libraries and the data.

Code:

```
import pandas as

pd import numpy

as np import

statistics as st #

Load the data

df =

pd.read_csv("diabetes.csv")

print(df.shape)

print(df.info())
```

```
(768, 9)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
None
```

Measures of Central Tendency

Measures of central tendency describe the center of the data, and are often represented by the mean, the median, and the mode.

Mean

Mean represents the arithmetic average of the data. The line of code below prints the mean of the numerical variables in the data. From the output, we can infer that the average age of the applicant is 49 years, the average annual income is USD 705,541, and the average tenure of loans is 183 months. The command df.mean(axis = 0) will also give the same output.

Code:

```
df.mean()
```

```
Out[2]: Pregnancies                 3.845052
        Glucose                   120.894531
        BloodPressure              69.105469
        SkinThickness              20.536458
        Insulin                    79.799479
        BMI                        31.992578
        DiabetesPedigreeFunction    0.471876
        Age                        33.240885
        Outcome                     0.348958
        dtype: float64
```

Code:

```
print(df.loc[:,'Age'].mean())

print(df.loc[:,'Income'].mean())
```

```
33.240885416666664
79.79947916666667
```

Median

In simple terms, median represents the 50th percentile, or the middle value of the data, that separates the distribution into two halves. The line of code below prints the median of the numerical variables in the data. The command df.median(axis = 0) will also give the same output.

Code:

```
df.median()
```

```
Out[5]: Pregnancies                 3.0000
        Glucose                   117.0000
        BloodPressure              72.0000
        SkinThickness              23.0000
        Insulin                    30.5000
        BMI                        32.0000
        DiabetesPedigreeFunction    0.3725
        Age                        29.0000
        Outcome                     0.0000
        dtype: float64
```

Mode

Mode represents the most frequent value of a variable in the data. This is the only central tendency measure that can be used with categorical variables, unlike the mean and the median which can be used only with quantitative data.

The line of code below prints the mode of all the variables in the data. The .mode() function returns the most common value or most repeated value of a variable. The command df.mode(axis = 0) will also give the same output.

Code:

```
df.mode()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1.0	99	70.0	0.0	0.0	32.0	0.254	22.0	0.0
1	NaN	100	NaN	NaN	NaN	NaN	0.258	NaN	NaN

Standard Deviation

Standard deviation is a measure that is used to quantify the amount of variation of a set of data values from its mean. A low standard deviation for a variable indicates that the data points tend to be close to its mean, and vice versa. The line of code below prints the standard deviation of all the numerical variables in the data.

Code:

```
df.std()
```

```
Out[7]: Pregnancies                 3.369578
        Glucose                    31.972618
        BloodPressure              19.355807
        SkinThickness              15.952218
        Insulin                   115.244002
        BMI                         7.884160
        DiabetesPedigreeFunction    0.331329
        Age                        11.760232
        Outcome                     0.476951
        dtype: float64
```

Variance

Variance is another measure of dispersion. It is the square of the standard deviation and the covariance of the random variable with itself. The line of code below prints the variance of all the numerical variables in the dataset. The interpretation of the variance is similar to that of the standard deviation.

Code:

```
df.var()
```

```
Out[8]: Pregnancies                 11.354056
        Glucose                   1022.248314
        BloodPressure              374.647271
        SkinThickness              254.473245
        Insulin                  13281.180078
        BMI                         62.159984
        DiabetesPedigreeFunction     0.109779
        Age                        138.303046
        Outcome                      0.227483
        dtype: float64
```

Interquartile Range (IQR)

The Interquartile Range (IQR) is a measure of statistical dispersion, and is calculated as the difference between the upper quartile (75th percentile) and the lower quartile (25th percentile). The IQR is also a very important measure for identifying outliers and could be visualized using a boxplot. IQR can be calculated using the iqr() function. The first line of code below imports the 'iqr' function from the scipy.stats module, while the second line prints the IQR for the variable 'Age'.

Code:

```
from scipy.stats import
```

```
iqr iqr(df['Age'])
```

```
Out[10]: 17.0
```

Skewness

Another useful statistic is skewness, which is the measure of the symmetry, or lack of it, for a real-valued random variable about its mean. The skewness value can be positive, negative, or undefined. In a perfectly symmetrical distribution, the mean, the median, and the mode will all have the same value. However, the variables in our data are not symmetrical, resulting in different values of the central tendency. We can calculate the skewness of the numerical variables using the skew() function, as shown below.

Code:

```
print(df.skew())
```

```
Pregnancies                0.901674
Glucose                    0.173754
BloodPressure             -1.843608
SkinThickness              0.109372
Insulin                    2.272251
BMI                       -0.428982
DiabetesPedigreeFunction   1.919911
Age                        1.129597
Outcome                    0.635017
dtype: float64
```

b. Bivariate analysis: Linear and logistic regression modeling

Bivariate Regression Analysis is a type of statistical analysis that can be used during the analysis and reporting stage of quantitative market research. It is often considered the simplest form of regression analysis, and is also known as Ordinary Least-Squares regression or linear regression.

Code:

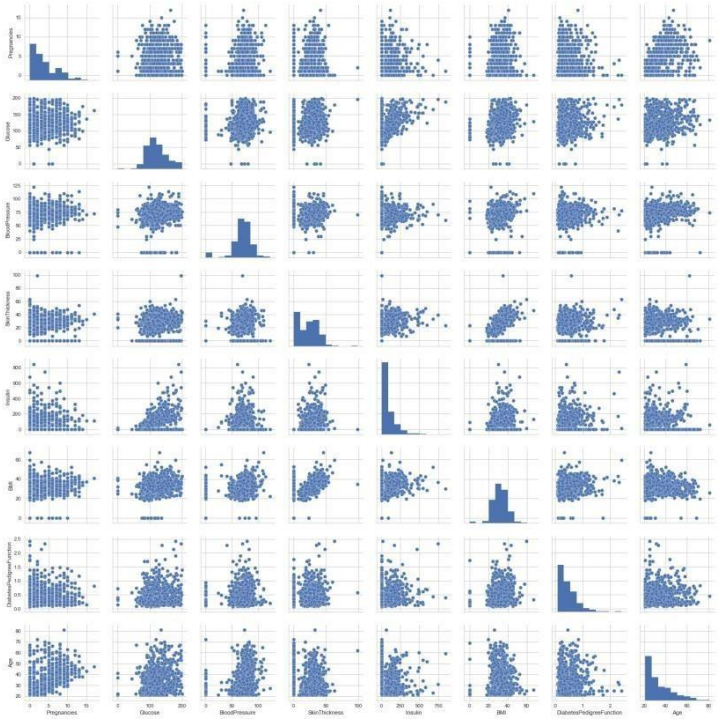
```
import pandas as pd

df = pd.read_csv(diabetes.csv')

df.head()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	1	85	66	29	0	26.6	0.351	31	0
1	1	89	66	23	94	28.1	0.167	21	0
2	5	116	74	0	0	25.6	0.201	30	0
3	10	115	0	0	0	35.3	0.134	29	0
4	4	110	92	0	0	37.6	0.191	30	0

Code:

```
import matplotlib.pyplot  as

plt import  seaborn as sns

sns.set(style='whitegrid', context='notebook')

cols =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age']
```

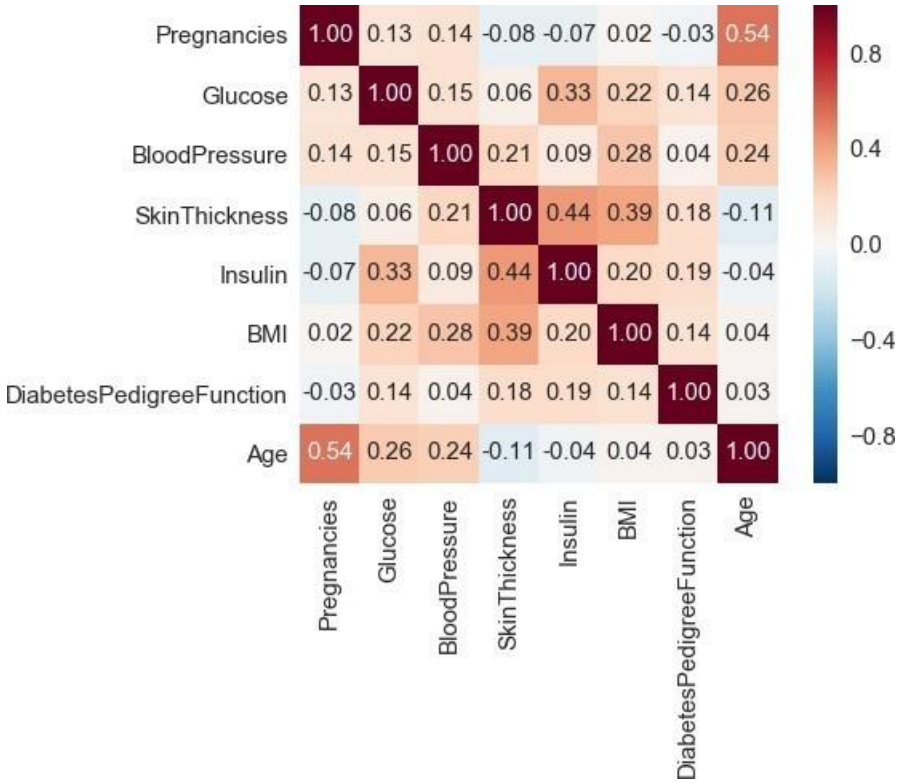


Code:

```
import numpy as np

cm =
```

```
np.corrcoef(df[cols].values.T)

sns.set(font_scale=1.5)
```

```
hm = sns.heatmap(cm,cbar=True,annot=True,square=True,fmt='.2f',annot_kws={'size': 15},yticklabels=cols,xticklabels=cols)

plt.show()
```



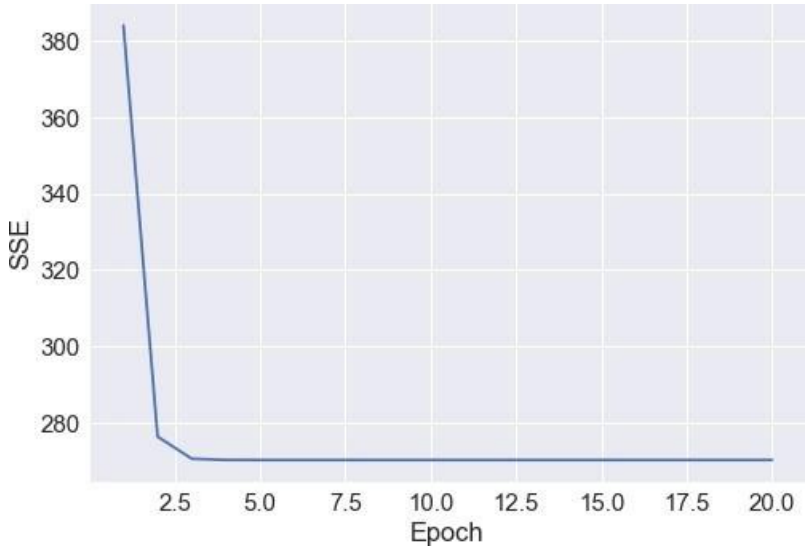
Code:

```
class LinearRegressionGD(object):

   def__init (self, eta=0.001, n_iter=20):

      self.eta = eta

      self.n_iter =

   n_iter def fit(self,

   X, y):

      self.w_ = np.zeros(1 +

      X.shape[1]) self.cost_ = []
```

```
for i in range(self.n_iter):
    output =
    self.net_input(X)
```

```
            errors = (y - output)
            self.w_[1:] += self.eta *
            X.T.dot(errors) self.w_[0] +=
            self.eta * errors.sum() cost =
            (errors**2).sum() / 2.0
            self.cost_.append(cost)
        return self
    def net_input(self, X):
        return np.dot(X, self.w_[1:]) +
    self.w_[0] def predict(self, X):
        return self.net_input(X)
X = df[['Age']].values
y = df['Pregnancies'].values
from sklearn.preprocessing import
StandardScaler sc_x = StandardScaler()
sc_y = StandardScaler()
X_std =
sc_x.fit_transform(X)
y_std =
sc_y.fit_transform(y) lr =
LinearRegressionGD()
lr.fit(X_std, y_std)
plt.plot(range(1, lr.n_iter+1),
```

```
lr.cost_) plt.ylabel('SSE')

plt.xlabel('Epoc

h') plt.show()
```



Code:

```
def lin_regplot(X,  y,
   model): plt.scatter(X, y,
   c='blue')
   plt.plot(X,  model.predict(X),
   color='red') return None
lin_regplot(X_std,  y_std, lr)
plt.xlabel('Age  (standardized)')
plt.ylabel('Pregnancies(standardized
)') plt.show()
```

Code:

```
age_std = sc_x.transform([20])

pregnancy_std =

lr.predict(age_std)

print("Pregnancy: %.3f"

%sc_y.inverse_transform(price_std)) print('Slope: %.3f'

% lr.w_[1])
```

c. Multiple Regression analysis:

Multiple regression is an extension of simple linear regression. It is used when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict is called the dependent variable (or sometimes, the outcome, target or criterion variable). The variables we are using to predict the value of the dependent variable are called the independent variables (or sometimes, the predictor, explanatory or regressor variables).

For example, you could use multiple regression to understand whether exam performance can be predicted based on revision time, test anxiety, lecture attendance and gender. Alternately, you could use multiple regression to understand whether daily cigarette consumption can be predicted based on smoking

duration, age when started smoking, smoker type, income and gender.

Multiple regression also allows you to determine the overall fit (variance explained) of the model and the relative contribution of each of the predictors to the total variance explained. For example, you might want to know how much of the variation in exam performance can be explained by revision time, test anxiety, lecture attendance and gender "as a whole", but also the "relative contribution" of each independent variable in explaining the variance.

This "quick start" guide shows you how to carry out multiple regression using SPSS Statistics, as well as interpret and report the results from this test. However, before we introduce you to this procedure, you need to understand the different assumptions that your data must meet in order for multiple regression to give you a valid result. We discuss these assumptions next.

Code:

```
from sklearn.model_selection import train_test_split

train_x, test_x, train_y, test_y =

train_test_split(X,Y,test_size=0.3,random_state=99) train_x.shape,

train_y.shape

from sklearn.linear_model import

MultipleRegression le = MultipleRegression()

le.fit(train_x,train_y)

y_pred =

le.predict(test_x)

y_pred

result = pd.DataFrame({'Actual': test_y,'Predict' :

y_pred}) result
```

	Actual	Predict
0	75.0	77.997933
1	128.0	170.444316
2	125.0	109.039494
3	332.0	223.843206
4	37.0	87.381459
...	...	...
128	48.0	202.350943
129	172.0	144.660842
130	51.0	82.400610
131	277.0	185.445124
132	94.0	101.563851

133 rows × 2 columns

Code:

```
print('coefficient',

le.coef_) print('intercept',

le.intercept_)
```

```
coefficient [  40.66018999 -313.29560706  517.1785363   386.06685795 -604.64498104
  275.32058758    3.91393457  172.38010275  661.95935148   62.25715134]
intercept 155.59114167162846
```

d. Also compare the results of the above analysis for the two data sets

For any one working in an analytical role, Comparing two data sets will be a day to day activity. Whether

that is to prove changes made are not impacting the rest of the data in the file which is typically called as "Regression testing" or to understand the difference between two files /data sets.

Installing datacompy

pip install datacompy

Details :

datacompy takes two dataframes as input and gives us a human-readable report containing statistics that lets us know the similarities and dissimilarities between the two dataframes. It will try to join two dataframes either on a list of join columns, or on indexes.

Code:

```
import datacompy

compare = datacompy.Compare(df1,df2,join_columns='acct_id',

abs_tol=0.0001,

rel_tol=0,df1_name='olddiabetes',df2_name='newdiabetes')

print(compare.report())
```

OUTPUT:

```
DataComPy Comparison
--------------------

DataFrame Summary
-----------------

DataFrame Columns Rows
0 original 5 7
1 new 4 6

Column Summary
--------------

Number of columns in common: 4
Number of columns in original but not in new: 1
Number of columns in new but not in original: 0
```

6. Apply and explore various plotting functions on UCI data sets

To load and quickly visualize the Multiple Features Dataset [1] from the UCI repository, which is available in mvlearn. This dataset can be a good tool for analyzing the effectiveness of multiview algorithms. It contains 6 views of handwritten digit images, thus allowing for analysis of multiview algorithms in multiclass or unsupervised tasks.

a. Normal curves

A probability distribution is a statistical function that describes the likelihood of obtaining the possible values that a random variable can take. By this, we mean the range of values that a parameter can take when we randomly pick up values from it. If we were asked to pick up 1 adult randomly and asked what his/her (assuming gender does not affect height) height would be? There's no way to know what the height will be. But if we have the distribution of heights of adults in the city, we can bet on the most probable outcome.A Normal Distribution is also known as a Gaussian distribution or famously Bell Curve. People use both words interchangeably, but it means the same thing. It is a continuous probability distribution.

Code:

```
import numpy as np

import matplotlib.pyplot as plt

# Creating a series of data of in range of

1-50. x = np.linspace(1,50,200)

#Creating a Function.

def normal_dist(x , mean , sd):

   prob_density = (np.pi*sd) *

   np.exp(-0.5*((x-mean)/sd)**2) return prob_density

#Calculate mean and Standard

deviation. mean = np.mean(x)

sd = np.std(x)

#Apply function to the

data. pdf =
```

```
normal_dist(x,mean,sd)
#Plotting the Results
plt.plot(x,pdf , color =
'red') plt.xlabel('Data
points')
```

```
plt.ylabel('Probability Density')
```

b. Density and contour plots

Contour plots also called level plots are a tool for doing multivariate analysis and visualizing 3-D plots in 2-D space. If we consider X and Y as our variables we want to plot then the response Z will be plotted as slices on the X-Y plane due to which contours are sometimes referred as Z-slices or iso-response.

Contour plots are widely used to visualize density, altitudes or heights of the mountain as well as in the meteorological department. Due to such wide usage matplotlib.pyplot provides a method contour to make it easy for us to draw contour plots.

Code:

```
import matplotlib.pyplot as

plt import numpy as np

feature_x = np.arange(0, 50, 2)

feature_y = np.arange(0, 50,

3) # Creating 2-D grid of

features

[X, Y] = np.meshgrid(feature_x,

feature_y) fig, ax = plt.subplots(1, 1)
```

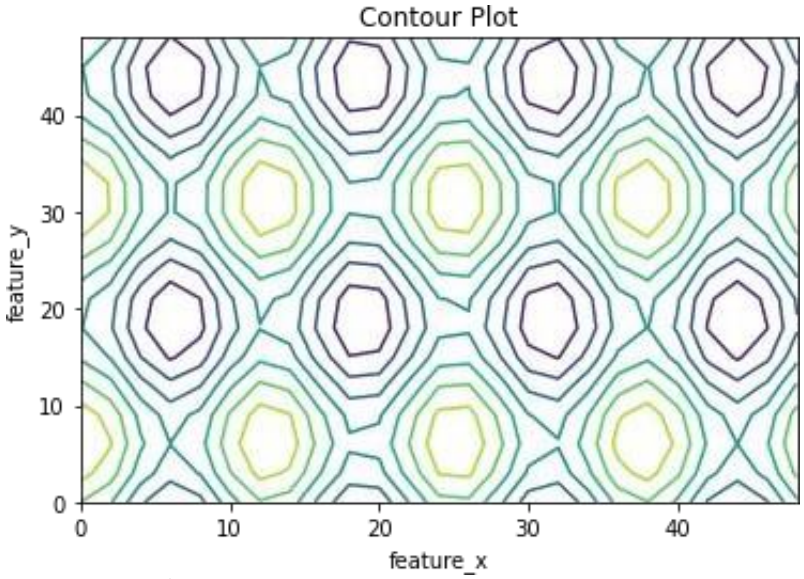
```
Z = np.cos(X / 2) +

np.sin(Y / 4) # plots contour

lines ax.contour(X, Y, Z)
```

ax.set_title('Contour

Plot')

ax.set_xlabel('feature_x

')



ax.set_ylabel('feature_y

') plt.show()

c. Correlation and scatter plots

Correlation means an association, It is a measure of the extent to which two variables are related.

1. Positive Correlation: When two variables increase together and decrease together. They are positively correlated. '1' is a perfect positive correlation. For example – demand and profit are positively correlated the more the demand for the product, the more profit hence positive correlation.

2. Negative Correlation: When one variable increases and the other variable decreases together and vice-versa. They are negatively correlated. For example, If the distance between magnet increases their attraction decreases, and vice-versa. Hence, a negative correlation. '-1' is no correlation

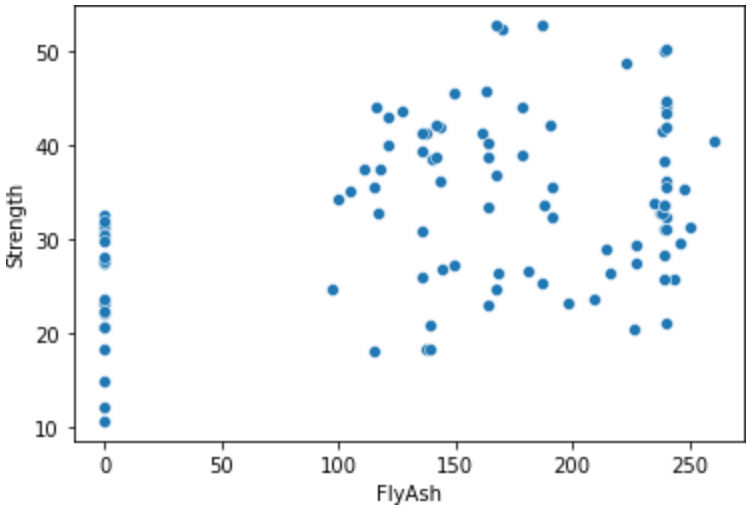
3. Zero Correlation(No Correlation): When two variables don't seem to be linked at all. '0' is a perfect negative correlation. For Example, the amount of tea you take and level of intelligence.

Code:

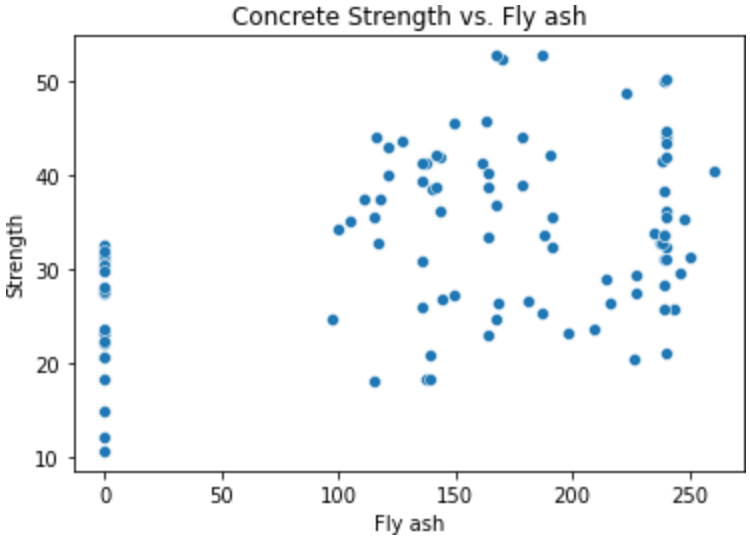
import pandas as pd

```
con =
pd.read_csv('concrete.csv')
con
list(con.columns)
```

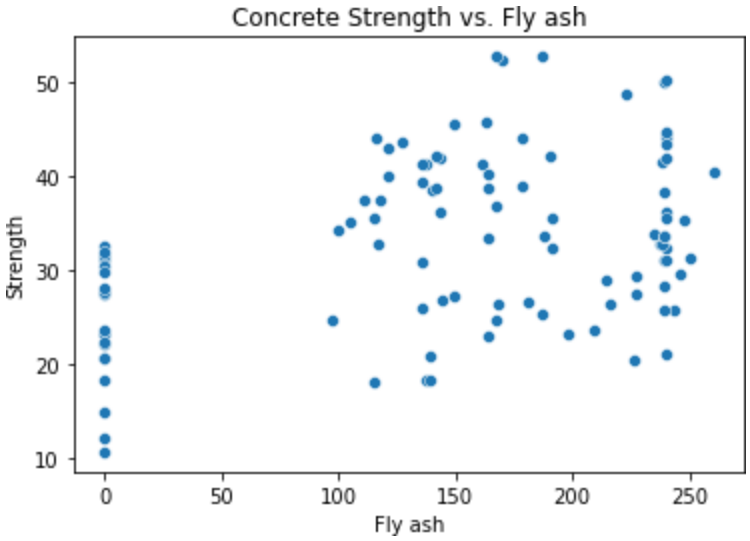
```
con.head()

con['cement'] =

con['cement'].astype('category')

con.describe(include='category')

import  seaborn as sns

sns.scatterplot(x="water", y="coarseagg", data=con);
```




```
ax = sns.scatterplot(x="water", y="coarseagg",

data=con) ax.set_title("Concrete Strength vs. Fly

ash") ax.set_xlabel("coarseagg");
```



```
sns.lmplot(x="water",  y="coarseagg", data=con);
```



d. Histograms:

A histogram is basically used to represent data provided in a form of some groups.It is accurate method for the graphical representation of numerical data distribution.It is a type of bar plot where X-axis represents the bin ranges while Y-axis gives information about frequency.

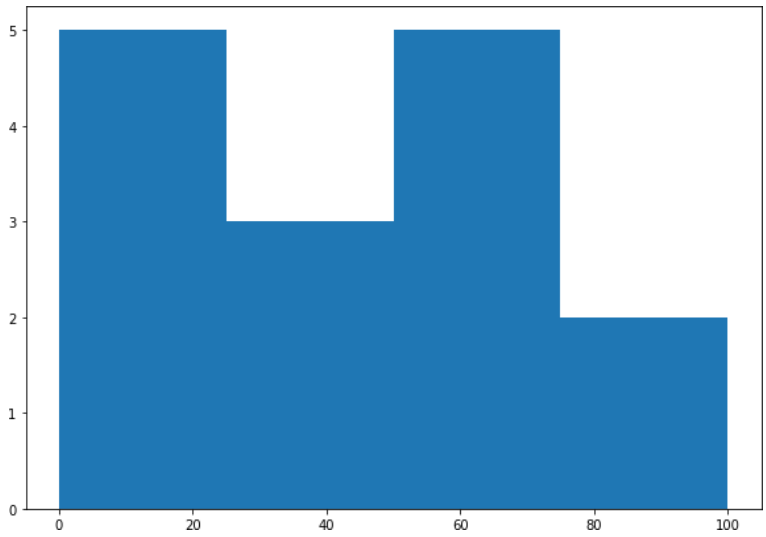
Creating a Histogram

To create a histogram the first step is to create bin of the ranges, then distribute the whole range of the values into a series of intervals, and count the values which fall into each of the intervals. Bins are clearly identified as consecutive, non-overlapping intervals of variables. The matplotlib.pyplot.hist() function is used to compute and create histogram of x.

Code:

```
from matplotlib import pyplot as

plt import numpy as np

# Creating dataset

a = np.array([22, 87, 5, 43, 56,

              73, 55, 54, 11,

              20, 51, 5, 79, 31,

              27])
```

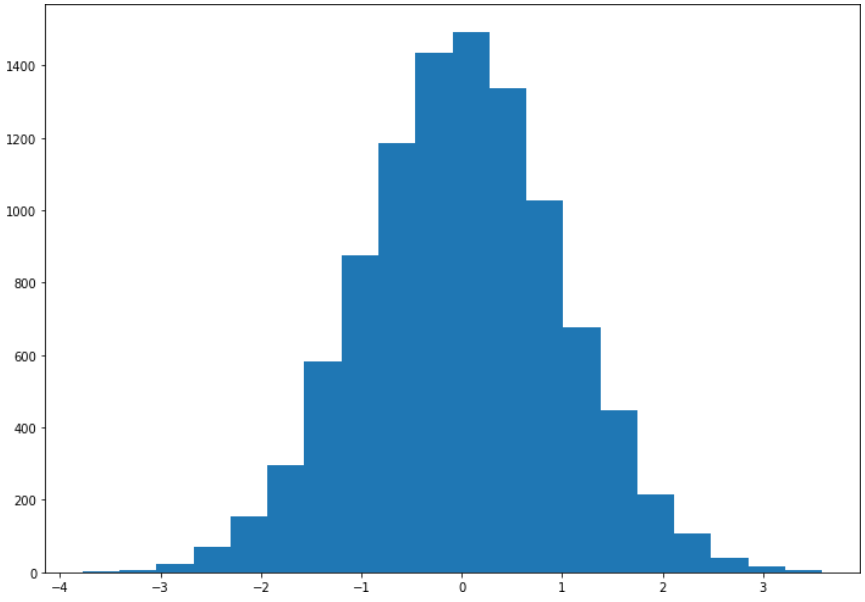
```
# Creating histogram
fig, ax = plt.subplots(figsize =(10, 7))
ax.hist(a, bins = [0, 25, 50, 75, 100])
# Show plot
plt.show()
```



Code:

```
import matplotlib.pyplot as plt
import numpy as np
from matplotlib import colors
from matplotlib.ticker import PercentFormatter
# Creating dataset
np.random.seed(23685752)
N_points = 10000
n_bins = 20
# Creating distribution
```

```
x = np.random.randn(N_points)

y = .8 ** x + np.random.randn(10000)  + 25

# Creating  histogram

fig,  axs = plt.subplots(1,  1,figsize  =(10, 7),tight_layout  = True)

axs.hist(x,  bins  = n_bins)

# Show plot

plt.show()
```



e. Three dimensional plotting

Matplotlib was introduced keeping in mind, only two-dimensional plotting. But at the time when the release of 1.0 occurred, the 3d utilities were developed upon the 2d and thus, we have 3d implementation of data available today! The 3d plots are enabled by importing the mplot3d toolkit. In this article, we will deal with the 3d plots using matplotlib.

Code:

```
from mpl_toolkits  import  mplot3d

import  numpy  as np
```

```
import matplotlib.pyplot as plt
fig = plt.figure()
# syntax for 3-D projection
ax = plt.axes(projection
='3d') # defining axes
z = np.linspace(0, 1, 100)
x = z * np.sin(25 * z)
y = z * np.cos(25 * z)
c = x + y
ax.scatter(x, y, z, c = c)
# syntax for plotting
ax.set_title('3d Scatter plot')
plt.show()
```

7. Visualizing Geographic Data with Basemap

One common type of visualization in data science is that of geographic data. Matplotlib's main tool for this type of visualization is the Basemap toolkit, which is one of several Matplotlib toolkits which lives under the mpl_toolkits namespace. Admittedly, Basemap feels a bit clunky to use, and often even simple visualizations take much longer to render than you might hope. More modern solutions such as leaflet or the Google Maps API may be a better choice for more intensive map visualizations. Still, Basemap is a useful tool for Python users to have in their virtual toolbelts. In this section, we'll show several examples of the type of map visualization that is possible with this toolkit.

Installation of Basemap is straightforward; if you're using conda you can type this and the package will be downloaded:

conda install basemap

Code:

```
%matplotlib

inline import

numpy as np

import matplotlib.pyplot as plt

from mpl_toolkits.basemap import

Basemap plt.figure(figsize=(8, 8))

m = Basemap(projection='ortho', resolution=None, lat_0=50,

lon_0=-100) m.bluemarble(scale=0.5);
```

```
fig = plt.figure(figsize=(8, 8))

m = Basemap(projection='lcc',
```

resolution=None, width=8E6,

height=8E6,

lat_0=45, lon_0=-100,)

```
m.etopo(scale=0.5, alpha=0.5)
# Map (long, lat) to (x, y) for
plotting  x, y = m(-122.3, 47.6)
plt.plot(x,  y, 'ok', markersize=5)
plt.text(x,  y, ' Seattle',
fontsize=12);
```


```
from mpl_toolkits.basemap  import
Basemap import matplotlib.pyplot  as plt
fig = plt.figure(figsize  =
(12,12)) m = Basemap()
m.drawcoastlines()
m.drawcoastlines(linewidth=1.0,  linestyle='dashed',
color='red') plt.title("Coastlines",  fontsize=20)
```

```
plt.show()
```

```
import numpy as
np import pandas
as pd
import matplotlib.pyplot as
plt import seaborn as sns
import geopandas as
gpd import shapefile
as shp
from shapely.geometry import
Point sns.set_style('whitegrid')
fp = r'Maps_with_python\india-
polygon.shp' map_df = gpd.read_file(fp)
map_df_copy =
gpd.read_file(fp)
plt.plot(map_df ,
markersize=5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x254015d94e0>
```