Rohan Nyati

500075940

R177219148

Batch-5 (AI & Ml)

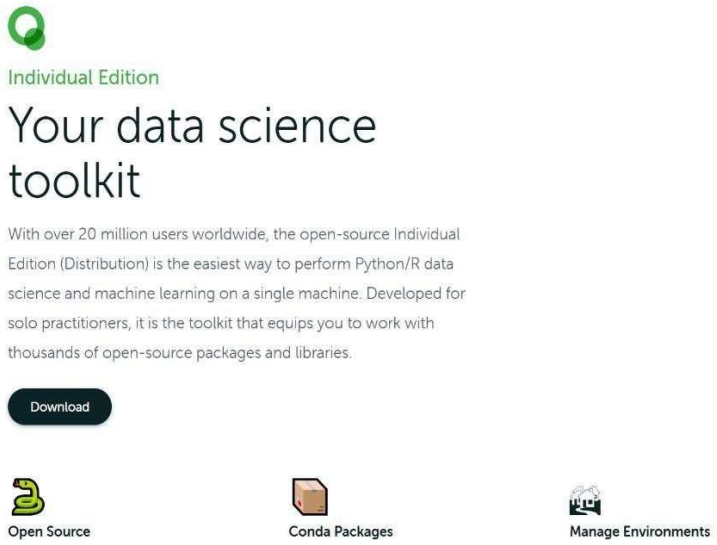
Lab File

Experiment 1

Anaconda & Spyder Installation for windows

1. Click on the link below to open the download page

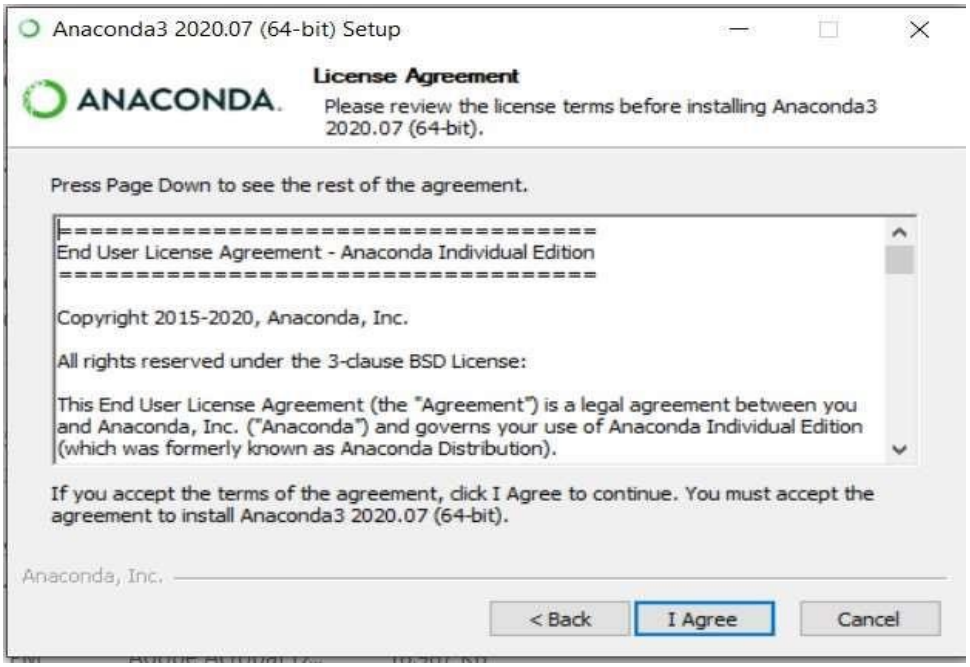
   https://www.anaconda.com/download/#windows



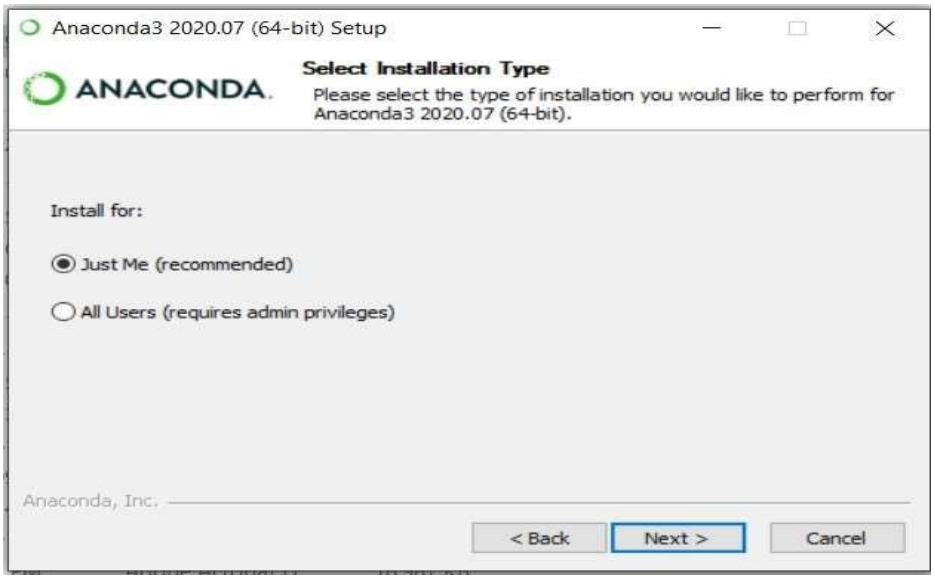
2. Click on the Download button and check for the compatibility of your system. Then, it will start downloading.

3. Double click the installer to launch.
4. Click on Next.

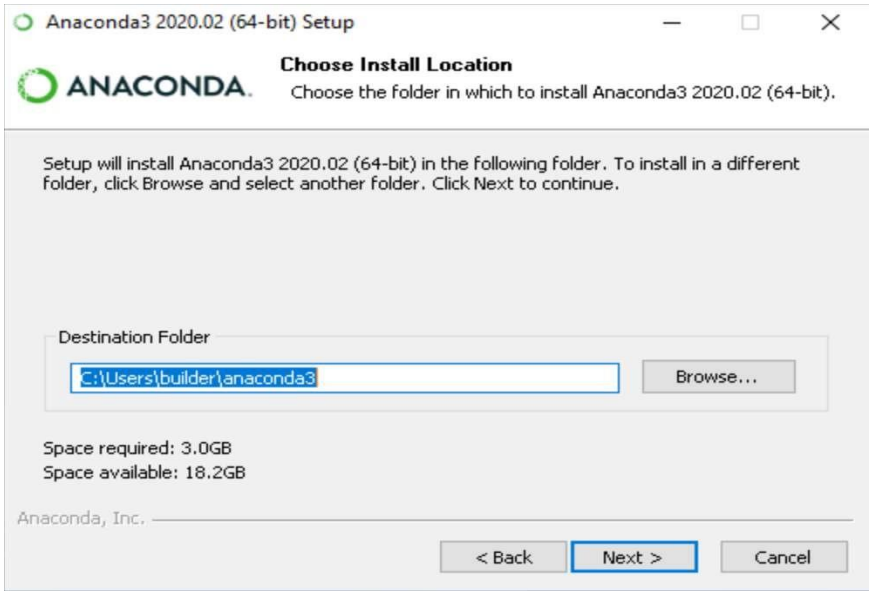
5. Read the license agreement and click on "I Agree".



6. Select installation type "Just Me" unless you're installing it for all users (which require Windows Administrator privileges) and click on Next.



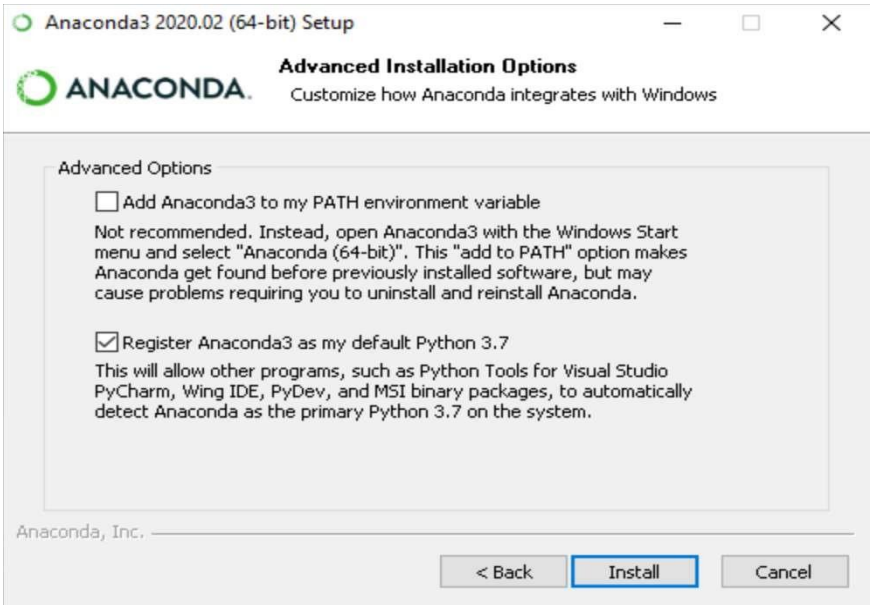
7. Select a destination folder to install Anaconda and click the Next button.



8. Choose whether to add Anaconda to your PATH environment variable. We recommend NOT adding Anaconda to the PATH environment variable, since this can interfere with other softwares. Instead, use Anaconda software by opening Anaconda Navigator or the Anaconda Prompt from the Start Menu

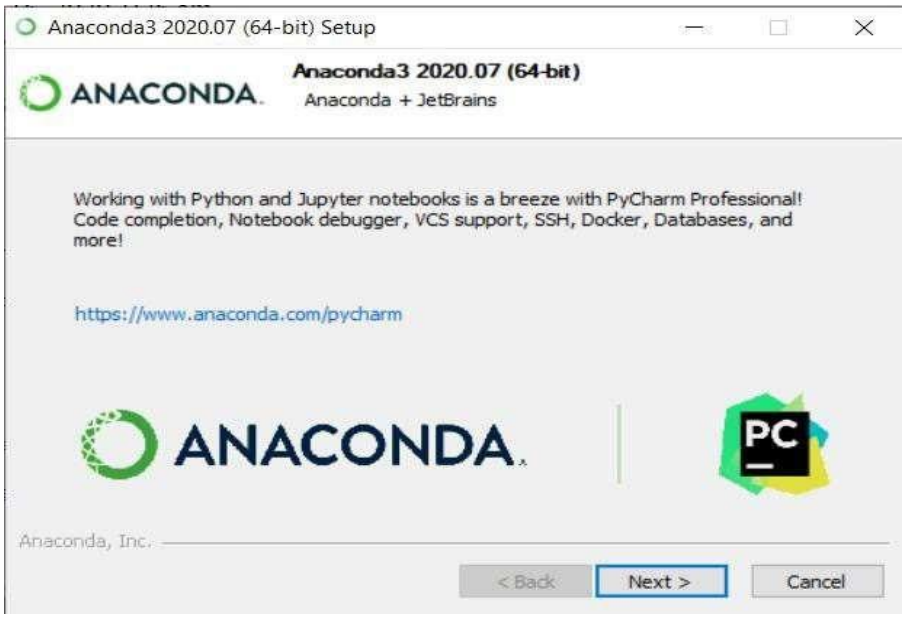
   Choose whether to register Anaconda as your default Python. Unless you plan to install and run multiple versions of Anaconda or multiple versions of Python, accept the default version and leave this box checked.

9. Click the Install button.

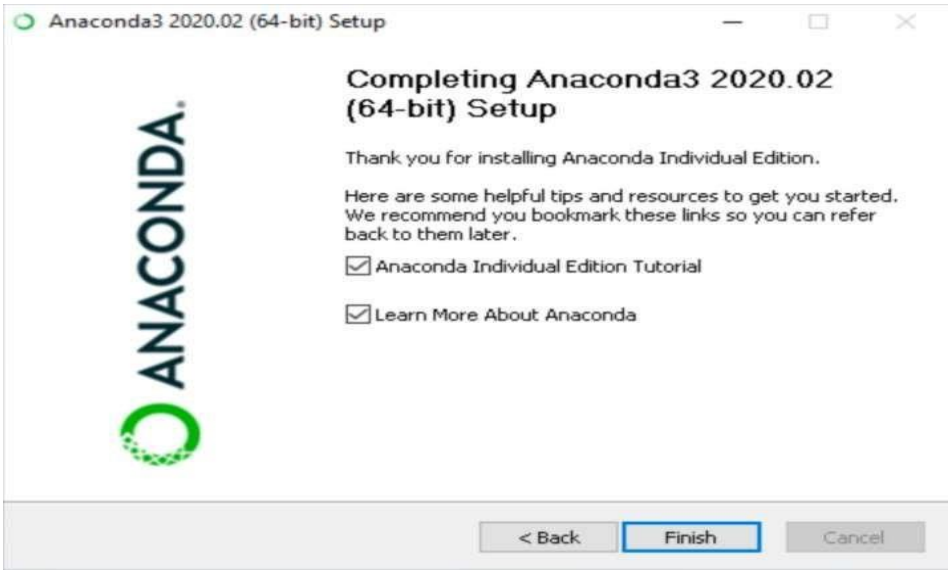


If you want to watch the packages Anaconda is installing, click on Show Details.

10. Click on the Next button.

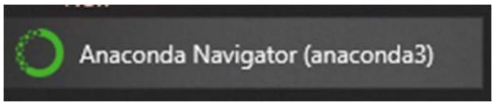


11. And then click the Finish button.

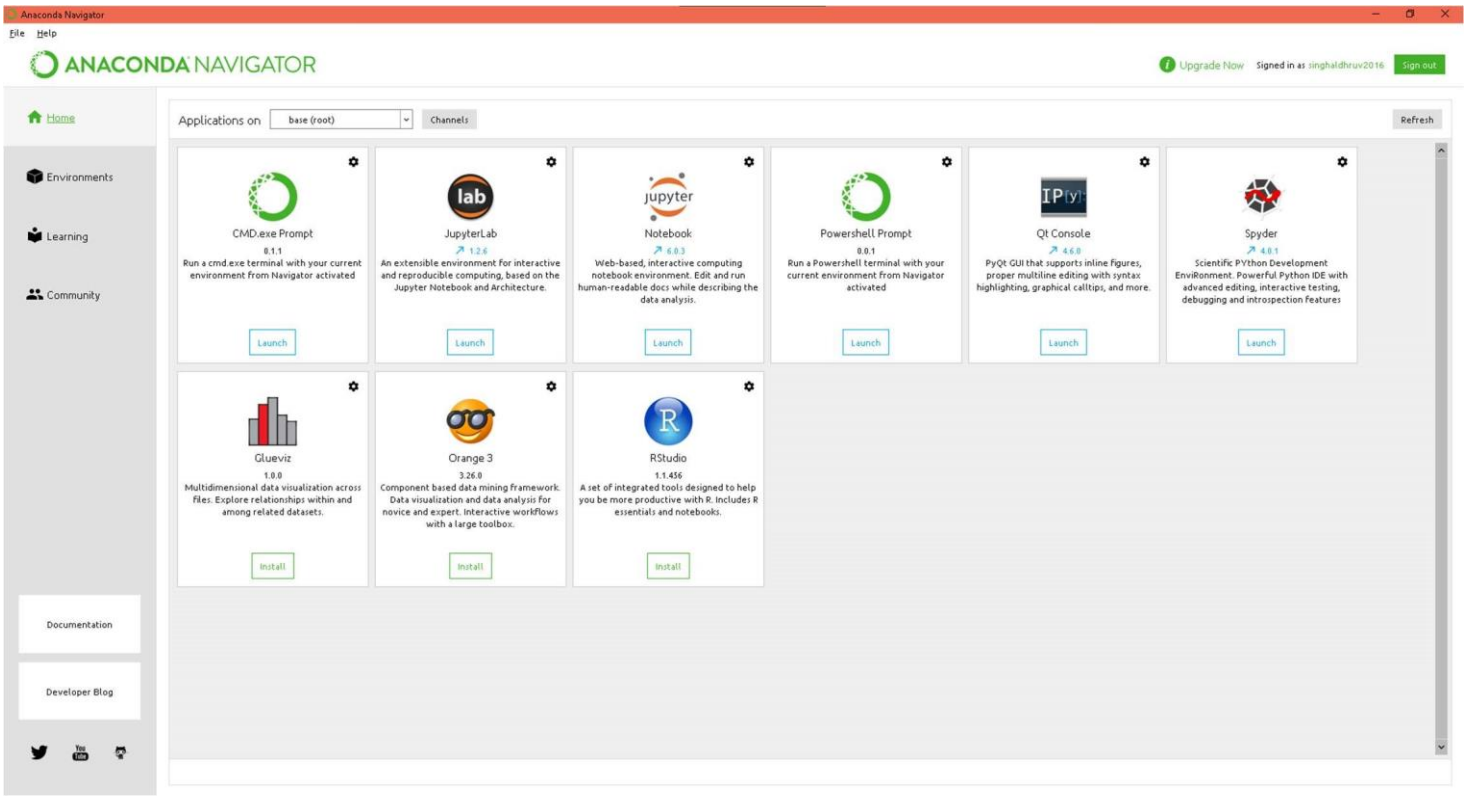


12. After a successful installation you will see the "Thanks for installing Anaconda" dialog box.

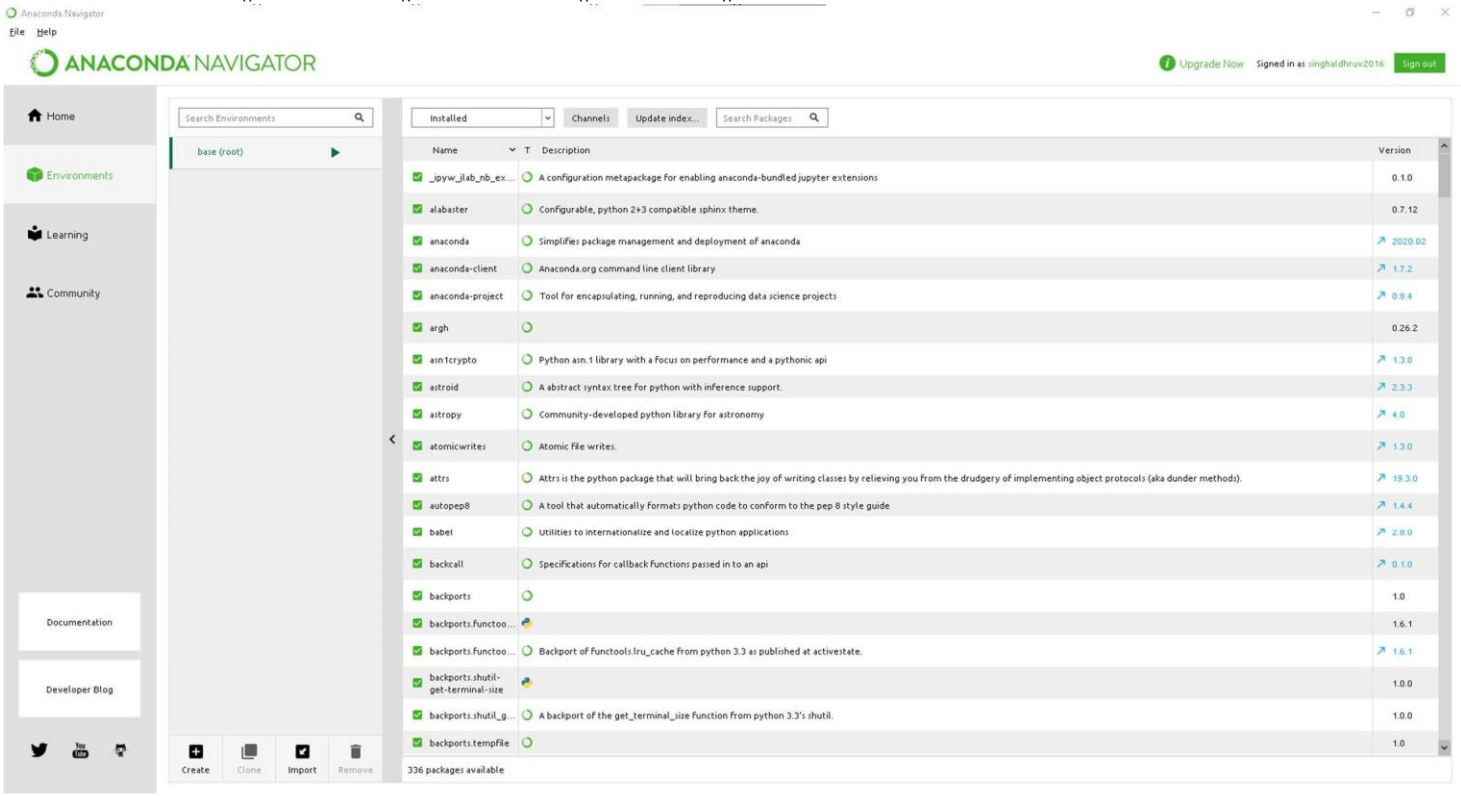
13.Open Anaconda navigator.



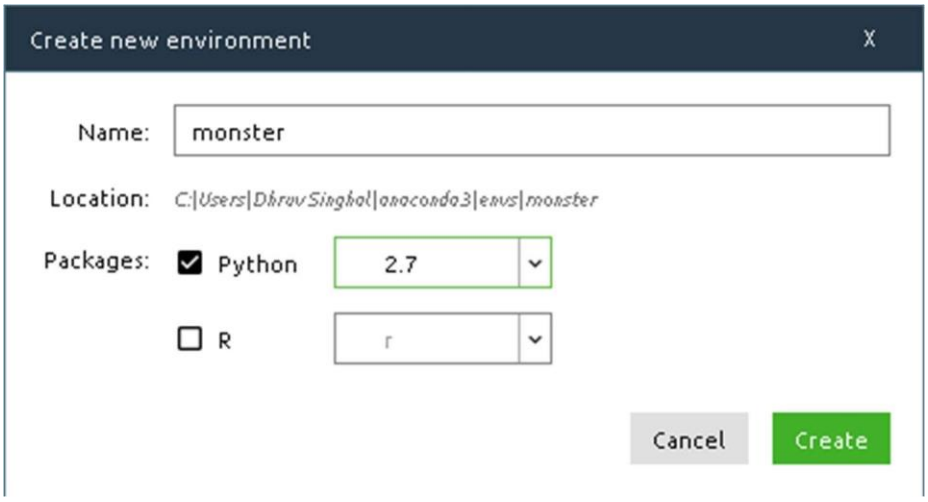
14. This screen will pop up.



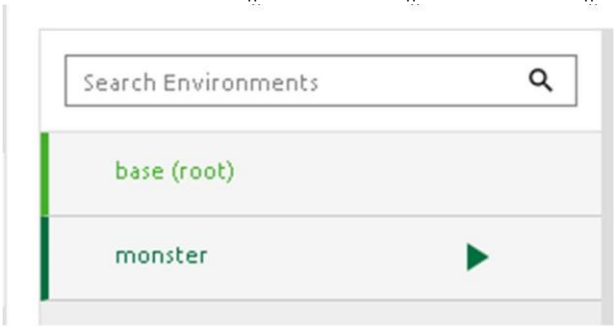
15.Go to “Environments “ from Left Side panel.



16. From Bottom Left click “Create” and choose version and name of the environment.



17. Create environment.



18. Activate Environment from Anaconda Prompt and Launch Spyder(After installing From Anaconda Navigator)

Spyder:

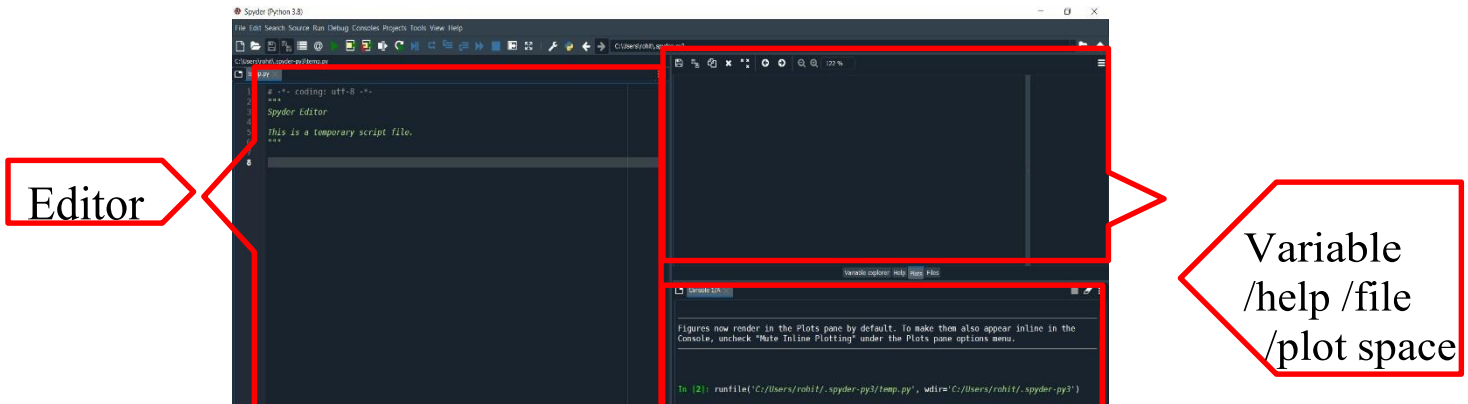
Spyder, the Scientific Python Development Environment, is a free integrated development environment (IDE) that is included with Anaconda.

It includes:

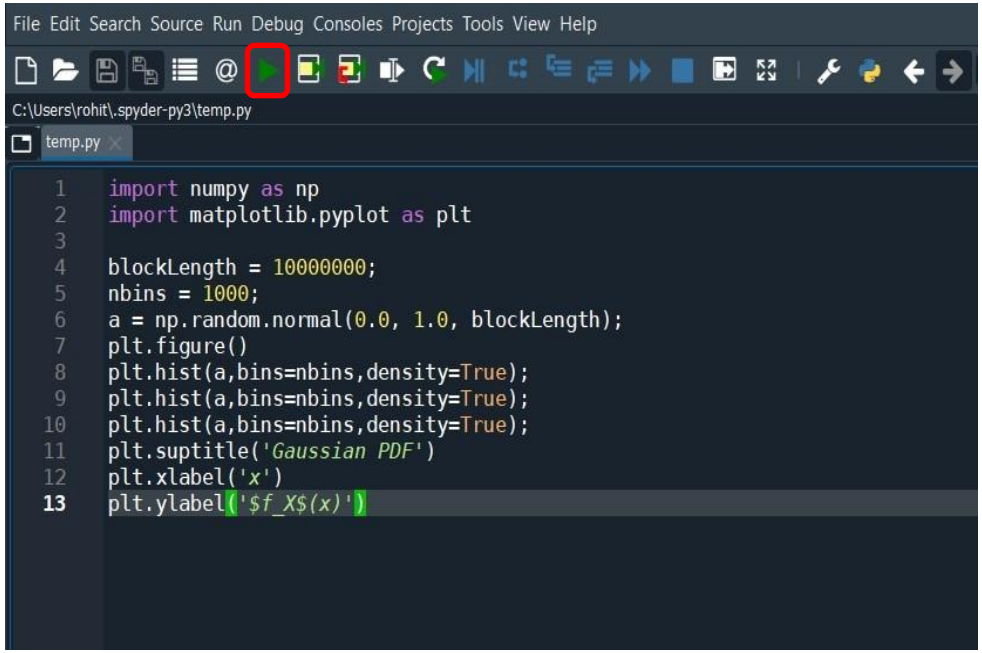
- Editing
- Interactive testing
- Debugging
- Introspection features

Steps for Spyder setup and run a test code:

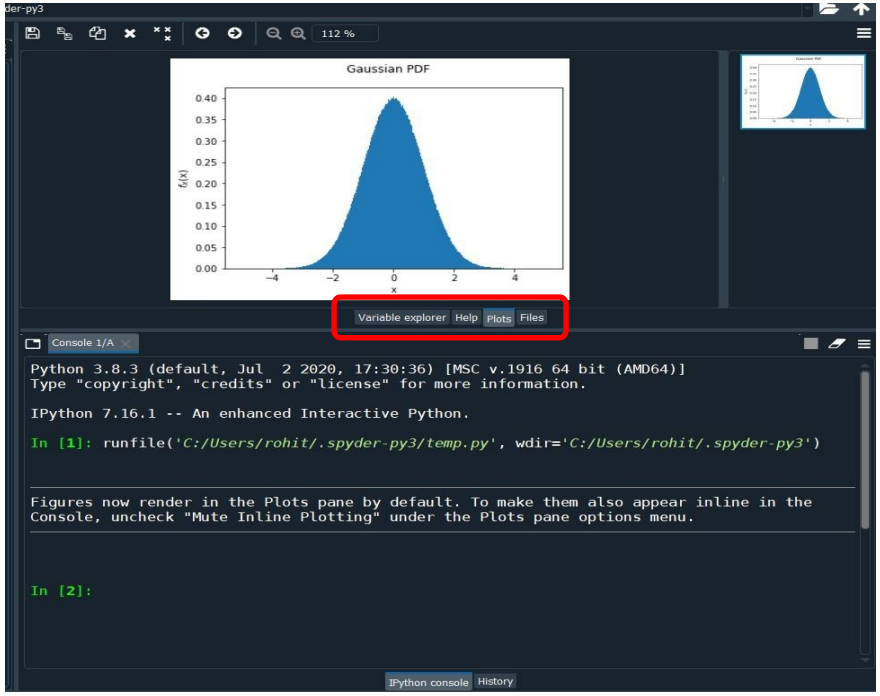
1. In Window search box, type Spyder and press Enter. 2. Spyder IDE opened and you can see a total of 3 area:
    a. Editor
    b. Console
    c. Variable/help/file/plot space



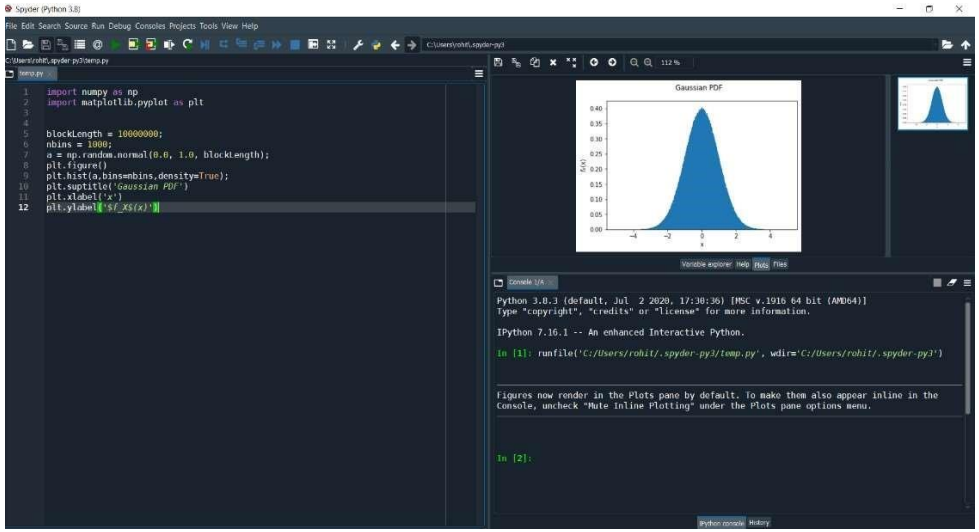
3. Let's write a test code in the Editor and run the code by clicking on Run button.



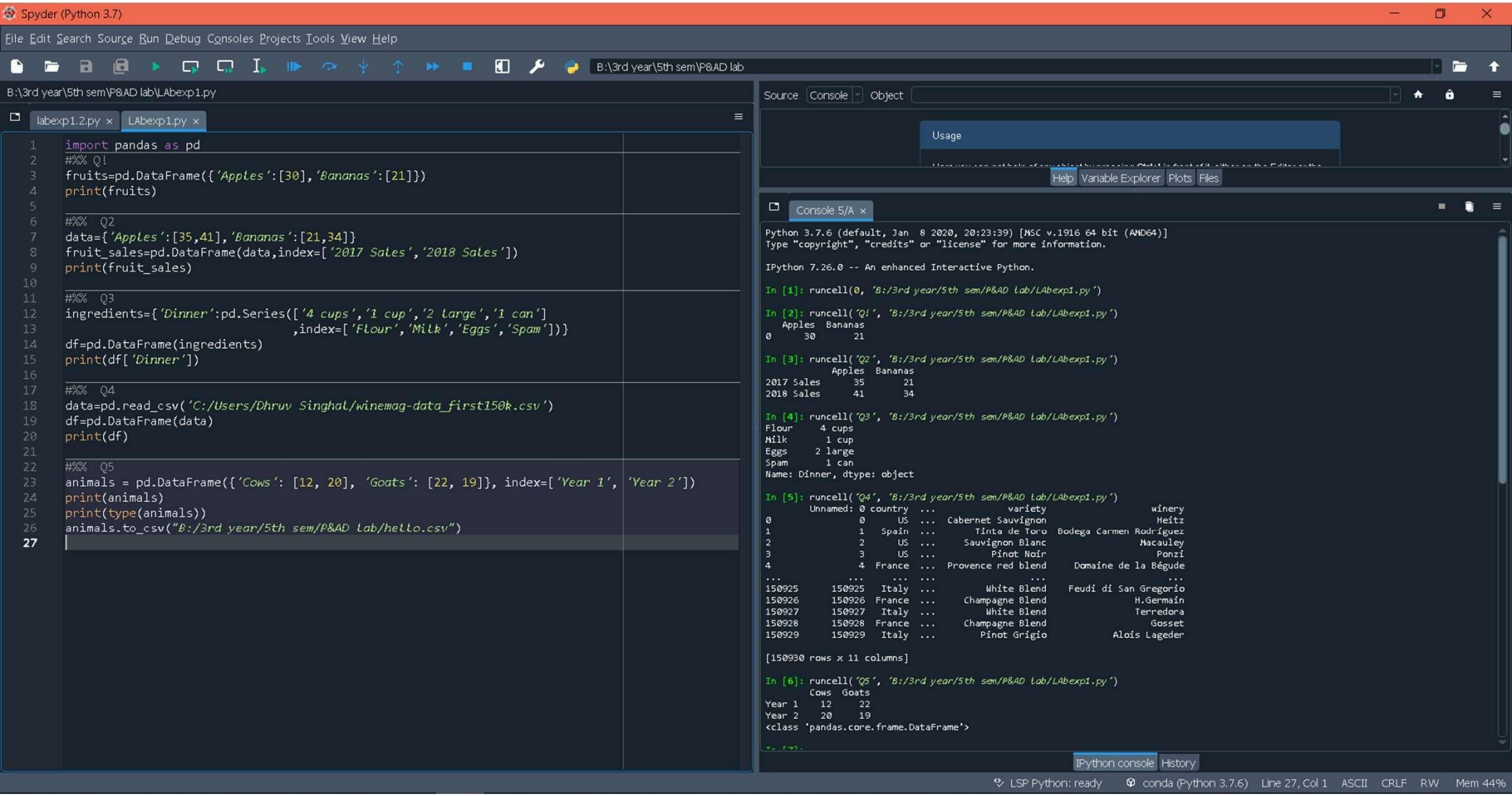
4. You can see the variable, plot, files on right side of IDE by clicking appropriate tabs as highlighted with Red color below.



5. As a whole Spyder screen looks like as below.


Spyder (Python 3.8)
File Edit Search Source Run Debug Consoles Projects Tools View Help
C:\Users\rohit\.spyder-py3\temp.py
temp.py
1 import numpy as np
2 import matplotlib.pyplot as plt
3
4
5 blockLength = 10000000;
6 nbins = 100;
7 a = np.random.normal(0.0, 1.0, blockLength);
8 plt.figure()
9 plt.hist(a,bins=nbins,density=True);
10 plt.suptitle('Gaussian PDF')
11 plt.xlabel('x')
12 plt.ylabel('$f_X(x)')
Gaussian PDF
f(x)
x
Variable explorer Help Plots Files
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/rohit/.spyder-py3/temp.py', wdir='C:/Users/rohit/.spyder-py3')
Figures now render in the Plots pane by default. To make them also appear inline in the Console, uncheck "Mute Inline Plotting" under the Plots pane options menu.
In [2]:
Python console History

Experiment 2



Pandas

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

B:\3rd year\5th sem\P&AD lab

B:\3rd year\5th sem\P&AD lab\LAbexp1.py

labexp1.2.py × LAbexp1.py ×

```
24  print(animals)
25  print(type(animals))
26  animals.to_csv("B:/3rd year/5th sem/P&AD lab/hello.csv")
27
28  #%%
29  reviews=pd.read_csv('C:/Users/Dhruv Singhal/winemag-data_first150k.csv')
30  reviews.head()
31
32  #%%  Q1: Select the description column from reviews and assign the result to the variable desc
33  desc = reviews.description
34  print(desc)
35  #%%
36  #Q2: Select the first value from the description column of reviews , assigning it to the variable
37  first_desc=reviews.description.iloc[0]
38  print(first_desc)
39  #%%
40  #Q3: Select the first row of data from reviews , assigning it to the variable
41  first_row=reviews.iloc[0]
42  print(first_row)
43
44  #%%
45  #Q4: Select the first 10 values from desciption column in review, assigning the result to the variable
46  first_desc=reviews['description'].head(10)
47  print(first_desc)
48
49  #%%
50  #Q5: Select the records with index labels 1,2,3,5,8 assigning the result to variable
51  Sample_reviews=reviews.iloc[[1,2,3,5,8]]
52  print(Sample_reviews)
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
```

Source Editor Object

5

Help Variable Explorer Plots Files

Console 7/A ×

```
In [3]: runcell('Q1: Select the description colum from reviews and assign the result to the variable desc', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
0         This tremendous 100% varietal wine hails from ...
1         Ripe aromas of fig, blackberry and cassis are ...
2         Mac Watson honors the memory of a wine once ma...
3         This spent 20 months in 30% new French oak, an...
4         This is the top wine from La Bégude, named aft...
                                ...
150925    Many people feel Fiano represents southern Ita...
150926    Offers an intriguing nose with ginger, lime an...
150927    This classic example comes from a cru vineyard...
150928    A perfect salmon shade, with scents of peaches...
150929    More Pinot Grigios should taste like this. A r...
Name: description, Length: 150930, dtype: object

In [4]: runcell(8, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
This tremendous 100% varietal wine hails from Oakville and was aged over three years in oak. Juicy red-cherry fruit and a compelling hint of caramel greet the palate, framed by elegant,
fine tannins and a subtle minty tone in the background. Balanced and rewarding from start to finish, it has years ahead of it to develop further nuance. Enjoy 2022-2030.

In [5]: runcell(9, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
Unnamed: 0                                                      0
country                                                        US
description    This tremendous 100% varietal wine hails from ...
designation                                  Martha's Vineyard
points                                                        96
price                                                      235.0
province                                             California
region_1                                           Napa Valley
region_2                                                  Napa
variety                                    Cabernet Sauvignon
winery                                                  Heitz
Name: 9, dtype: object

In [6]: runcell(10, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
0    This tremendous 100% varietal wine hails from ...
1    Ripe aromas of fig, blackberry and cassis are ...
2    Mac Watson honors the memory of a wine once ma...
3    This spent 20 months in 30% new French oak, an...
4    This is the top wine from La Bégude, named aft...
5    Deep, dense and pure from the opening bell, th...
6    Slightly gritty black-fruit aromas include a s...
7    Lush cedary black-fruit aromas are luxe and of...
8    This re-named vineyard was formerly bottled as...
9    The producer sources from two blocks of the vi...
Name: description, dtype: object

In [7]: runcell(11, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')

In [8]: runcell(11, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
   Unnamed: 0 country  ...          variety               winery
1           1   Spain  ...  Tinta de Toro  Bodega Carmen Rodríguez
2           2      US  ...  Sauvignon Blanc               Macauley
3           3      US  ...       Pinot Noir                Ponzi
5           5   Spain  ...  Tinta de Toro             Numanthia
8           8      US  ...       Pinot Noir             Bergström

[5 rows x 11 columns]
```

IPython console History

LSP Python: ready conda (Python 3.7.6) Line 58, Col 1 ASCII CRLF RW Mem 43%

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

B:\3rd year\5th sem\P&AD lab

B:\3rd year\5th sem\P&AD lab\LAbexp1.py

LAbexp1.py

```
#%%
#Q6: Create a variable df containing the country,province,region_1 and region_2 column of records
cols=['country','province','region_1','region_2']
indices=[0,1,10,100]
df=reviews.loc[indices,cols]
print(df)
#%%
#Q7: Create a variable containing the country and variety col of first 100 recors
cols_ine=['country','variety']
df=reviews.loc[:100,cols_ine]
print(df)
#%%
#Q8: create a dataframe italian_wines containing reviews of wine made3 in italy
italian_wine=reviews[reviews['country']=='Italy']
print(italian_wine)
#%%
#Q9:create a dataframe  containing all reviews with at least 95 points (out of 100) for wines from Australia
top_oceania_wines = reviews.loc[reviews.country.isin(['Australia', 'New Zealand']) & (reviews.points >= 95)]
print(top_oceania_wines)
```

Name	Type	Size	Value
cols	list	4	['country', 'province', 'region_1', 'region_2']
cols_ine	list	2	['country', 'variety']

Help | Variable Explorer | Plots | Files

Console 10/A

```
In [13]: runcell(15, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
      Unnamed: 0    country  ...            variety          winery
2148        2148  Australia  ...             Shiraz        Torbreck
2458        2458  Australia  ...     Cabernet-Shiraz     Hickinbotham
3033        3033  Australia  ... Cabernet Sauvignon         Penfolds
3044        3044  Australia  ...             Shiraz        Henschke
3047        3047  Australia  ...          Red Blend       Heartland
...          ...        ...  ...                ...             ...
122779    122779  Australia  ...             Shiraz         Standish
127614    127614  Australia  ...             Shiraz        Henschke
137383    137383  Australia  ...              Syrah  Clarendon Hills
150562    150562  Australia  ...             Shiraz         Penfolds
150563    150563  Australia  ...             Shiraz         Penfolds

[82 rows x 11 columns]

In [14]: runcell(13, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
   country              variety
0       US   Cabernet Sauvignon
1    Spain        Tinta de Toro
2       US      Sauvignon Blanc
3       US           Pinot Noir
4   France   Provence red blend
..     ...                  ...
96      US           Chardonnay
97      US   Cabernet Sauvignon
98  France        Merlot-Malbec
99  France  Ugni Blanc-Colombard
100     US             Viognier

[101 rows x 2 columns]

In [15]: runcell(14, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
      Unnamed: 0 country  ...          variety                winery
10            10   Italy  ...         Friulano      Bongo del Tiglio
32            32   Italy  ...       Sangiovese       Abbadia Ardenga
35            35   Italy  ...       Sangiovese              Carillon
37            37   Italy  ...       Sangiovese           Avignonesi
38            38   Italy  ...       Sangiovese      Casina di Cornia
...          ...     ...  ...              ...                 ...
150920    150920   Italy  ... Champagne Blend             Letrari
150922    150922   Italy  ...            Tocai     Ronchi di Manzano
150925    150925   Italy  ...       White Blend  Feudi di San Gregorio
150927    150927   Italy  ...       White Blend           Terredora
150929    150929   Italy  ...      Pinot Grigio         Alois Lageder

[23478 rows x 11 columns]

In [16]: runcell(15, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
      Unnamed: 0    country  ...            variety          winery
2148        2148  Australia  ...             Shiraz        Torbreck
2458        2458  Australia  ...     Cabernet-Shiraz     Hickinbotham
3033        3033  Australia  ... Cabernet Sauvignon         Penfolds
3044        3044  Australia  ...             Shiraz        Henschke
3047        3047  Australia  ...          Red Blend       Heartland
...          ...        ...  ...                ...             ...
122779    122779  Australia  ...             Shiraz         Standish
```

IPython console | History

LSP Python: ready | conda (Python 3.7.6) | Line 89, Col 1 | ASCII | CRLF | RW | Mem 46%

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

B:\3rd year\5th sem\P&AD lab

B:\3rd year\5th sem\P&AD lab\LAbexp1.py

LAbexp1.py

```
113
114
115 #%% What is the median of the points column in the reviews DataFrame?
116 med_points=reviews['points'].median()
117 print(med_points)
118
119 #%%What countries are represented in the dataset? (Your answer should not include any duplicat
120 countries=reviews.country.unique()
121 print(countries)
122
123 #%% How often does each country appear in the dataset? Create a Series reviews_per_country map
124
125 reviews_per_country=reviews.country.value_counts()
126 print(reviews_per_country)
127
128 #%% Create variable centered_price containing a version of the price column with the mean pric
129
130 centered_price = reviews.price- reviews.price.mean()
131 print(centered_price)
132
133 #%% I'm an economical wine buyer. Which wine is the "best bargain"? Create a variable bargain_
134 bargain_idx=(reviews.points/reviews.price).idxmax()
135 bargain_wine = reviews.loc[bargain_idx]
136 print(bargain_wine)
137
138 #%% There are only so many words you can use when describing a bottle of wine. Is a wine more
139 n_trop = reviews['description'].map(lambda desc: "tropical" in desc ).sum()
140 n_fruity = reviews['description'].map(lambda desc: "fruity" in desc ).sum()
141 descriptor_counts=pd.Series([n_trop,n_fruity],index=['tropical','fruity'])
142 print(descriptor_counts)
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
```

Source Editor Object

No documentation available

Help Variable Explorer Plots Files

Console 12/A

```
88.0

In [4]: runcell('What countries are represented in the dataset? (Your answer should not include any duplicates.)', 'B:/3rd year/5th sem/P&AD
lab/LAbexp1.py')
['US' 'Spain' 'France' 'Italy' 'New Zealand' 'Bulgaria' 'Argentina'
 'Australia' 'Portugal' 'Israel' 'South Africa' 'Greece' 'Chile' 'Morocco'
 'Romania' 'Germany' 'Canada' 'Moldova' 'Hungary' 'Austria' 'Croatia'
 'Slovenia' nan 'India' 'Turkey' 'Macedonia' 'Lebanon' 'Serbia' 'Uruguay'
 'Switzerland' 'Albania' 'Bosnia and Herzegovina' 'Brazil' 'Cyprus'
 'Lithuania' 'Japan' 'China' 'South Korea' 'Ukraine' 'England' 'Mexico'
 'Georgia' 'Montenegro' 'Luxembourg' 'Slovakia' 'Czech Republic' 'Egypt'
 'Tunisia' 'US-France']

In [5]: runcell('How often does each country appear in the dataset? Create a Series reviews_per_country mapping countries to the count of
reviews of wines from that country.', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
US                          62397
Italy                       23478
France                      21098
Spain                        8268
Chile                        5816
Argentina                    5631
Portugal                     5322
Australia                    4957
New Zealand                  3320
Austria                      3057
Germany                      2452
South Africa                 2258
Greece                        884
Israel                        630
Hungary                       231
Canada                        196
Romania                       139
Slovenia                       94
Uruguay                        92
Croatia                        89
Bulgaria                       77
Moldova                        71
Mexico                         63
Turkey                         52
Georgia                        43
Lebanon                        37
Cyprus                         31
Brazil                         25
Macedonia                      16
Serbia                         14
Morocco                        12
```

IPython console History

LSP Python: ready conda (Python 3.7.6) Line 147, Col 1 ASCII CRLF RW Mem 44%

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

B:\3rd year\5th sem\P&AD lab

B:\3rd year\5th sem\P&AD lab\LAbexp1.py

LAbexp1.py

```
#%%What is the best wine I can buy for a given amount of money? Create a Series whose index is wine prices an
best_rating_per_price = reviews.groupby('price')['points'].max().sort_index()
print(best_rating_per_price)

#%% What are the minimum and maximum prices for each variety of wine? Create a DataFrame whose index is the v
price_extremes = reviews.groupby('variety').price.agg([min, max])
print(price_extremes)

#%%  What are the most expensive wine varieties? Create a variable sorted_varieties containing a copy of the
sorted_varieties = price_extremes.sort_values(by=['min', 'max'], ascending=False)
print(sorted_varieties)

#%% What combination of countries and varieties are most common? Create a Series whose index is a MultiIndexc
country_variety_counts = reviews.groupby(['country', 'variety']).size().sort_values(ascending=False)
print(country_variety_counts)
```

Name	Type	Size	Value
best_rating_per_price	Series	(357,)	Series object of pandas.core.series module
cols	list	4	['country', 'province', 'region_1', 'region_2']

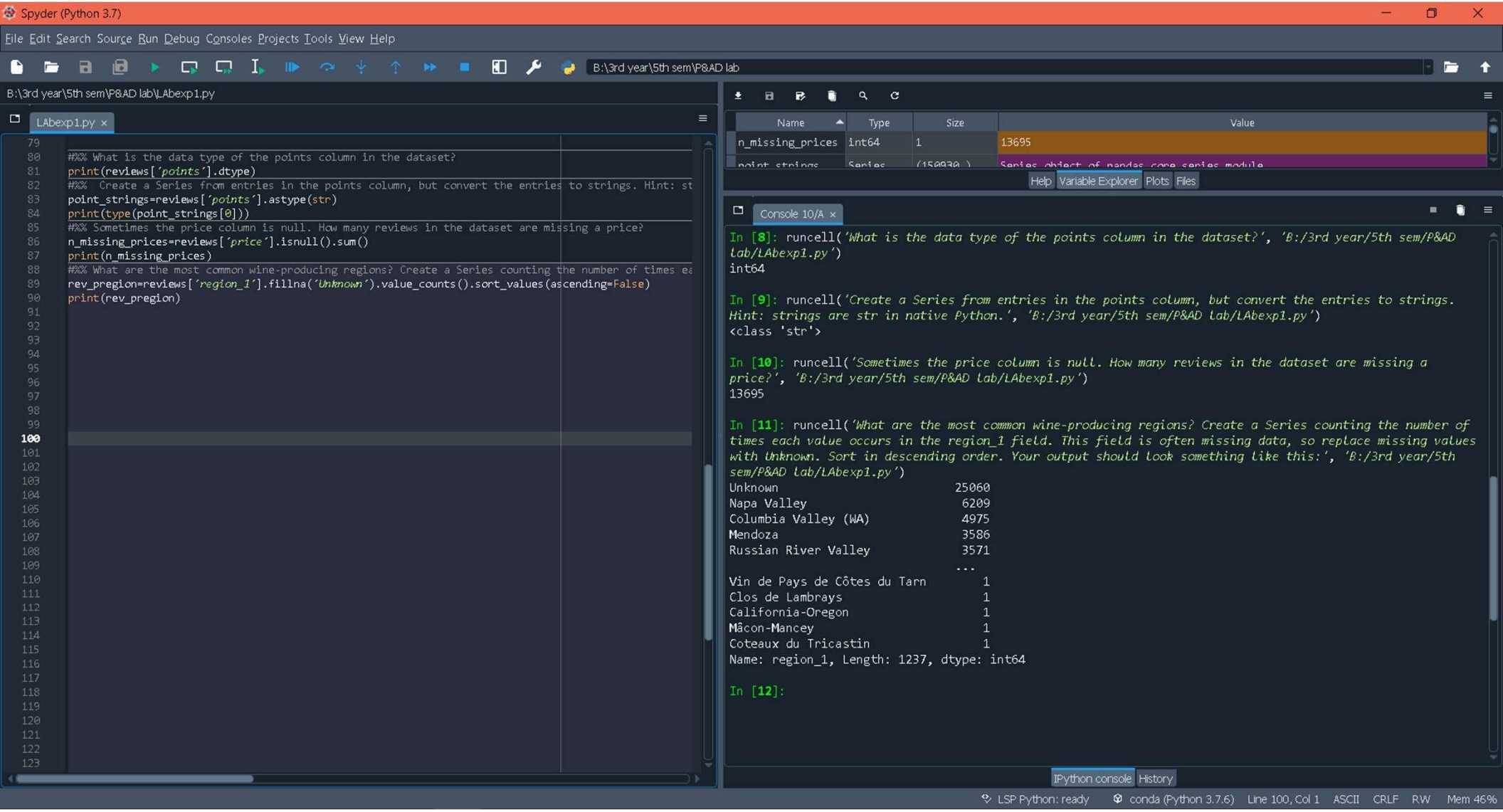
Help Variable Explorer Plots Files

Console 10/A

```
2013.0    91
2300.0    99
Name: points, Length: 357, dtype: int64

In [18]: runcell('What are the minimum and maximum prices for each variety of wine? Create a DataFrame whose index is the variety category from the dataset and whose values are the min and max values
thereof.', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
                          min    max
variety
Agiorgitiko               8.0   65.0
Aglianico                 6.0  130.0
Aidani                   27.0   27.0
Airen                     8.0   10.0
Albana                    8.0   66.0
...                       ...    ...
Zierfandler-Rotgipfler   20.0   25.0
Zinfandel                 4.0  100.0
Zlahtina                 13.0   17.0
Zweigelt                  9.0   70.0
Žilavka                  13.0   15.0

[632 rows x 2 columns]

In [19]: runcell('What are the most expensive wine varieties? Create a variable sorted_varieties containing a copy of the dataframe from the previous question where varieties are sorted in descending
order based on minimum price, then on maximum price (to break ties).', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
                                 min    max
variety
Cabernet-Shiraz                150.0  150.0
Mazuelo                         92.0  105.0
Carignan-Syrah                  80.0   80.0
Syrah-Cabernet Franc            69.0   69.0
Nasco                           65.0   65.0
...                              ...    ...
Rabigato                         NaN    NaN
Sacy                             NaN    NaN
Sauvignon Blanc-Sauvignon Gris   NaN    NaN
Terret Blanc                     NaN    NaN
Zelen                            NaN    NaN

[632 rows x 2 columns]

In [20]: runcell('What combination of countries and varieties are most common? Create a Series whose index is a MultiIndexof {country, variety} pairs. For example, a pinot noir produced in the US
should map to {"US", "Pinot Noir"}. Sort the values in the Series in descending order based on wine count.', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
country      variety
US           Pinot Noir                  10340
             Cabernet Sauvignon           9178
             Chardonnay                   8127
France       Bordeaux-style Red Blend     4960
US           Syrah                        4274
                                          ...
France       Pied de Perdrix                 1
Tunisia      White Blend                     1
             Rosé                            1
Switzerland  White Blend                     1
US           Carignan-Grenache               1
Length: 1475, dtype: int64

In [21]:
```

IPython console History

LSP Python: ready conda (Python 3.7.6) Line 122, Col 1 ASCII CRLF RW Mem 45%


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
B:\3rd year\5th sem\P&AD lab
B:\3rd year\5th sem\P&AD lab\LAbexp1.py
LAbexp1.py
#%% What is the data type of the points column in the dataset?
print(reviews['points'].dtype)
#%% Create a Series from entries in the points column, but convert the entries to strings. Hint: st
point_strings=reviews['points'].astype(str)
print(type(point_strings[0]))
#%% Sometimes the price column is null. How many reviews in the dataset are missing a price?
n_missing_prices=reviews['price'].isnull().sum()
print(n_missing_prices)
#%% What are the most common wine-producing regions? Create a Series counting the number of times ea
rev_pregion=reviews['region_1'].fillna('Unknown').value_counts().sort_values(ascending=False)
print(rev_pregion)
Name
Type
Size
Value
n_missing_prices
int64
1
13695
Help
Variable Explorer
Plots
Files
Console 10/A
In [8]: runcell('What is the data type of the points column in the dataset?', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
int64
In [9]: runcell('Create a Series from entries in the points column, but convert the entries to strings. Hint: strings are str in native Python.', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
<class 'str'>
In [10]: runcell('Sometimes the price column is null. How many reviews in the dataset are missing a price?', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
13695
In [11]: runcell('What are the most common wine-producing regions? Create a Series counting the number of times each value occurs in the region_1 field. This field is often missing data, so replace missing values with Unknown. Sort in descending order. Your output should look something like this:', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
Unknown 25060
Napa Valley 6209
Columbia Valley (WA) 4975
Mendoza 3586
Russian River Valley 3571
...
Vin de Pays de Côtes du Tarn 1
Clos de Lambrays 1
California-Oregon 1
Mâcon-Mancey 1
Coteaux du Tricastin 1
Name: region_1, Length: 1237, dtype: int64
In [12]:
IPython console
History
LSP Python: ready
conda (Python 3.7.6)
Line 100, Col 1
ASCII
CRLF
RW
Mem 46%

Spyder (Python 3.7)

File Edit Search Source Run Debug Consoles Projects Tools View Help

B:\3rd year\5th sem\P&AD lab

B:\3rd year\5th sem\P&AD lab\LAbexp1.py

LAbexp1.py

```
#%% region_1 and region_2 are pretty uninformative names for locale columns in the dataset. Create a
renamed=reviews.rename(columns=dict(region_1='region',region_2='Locale'))
print(renamed.head())
#%%  Set the index name in the dataset to wines.
reindexed=reviews.rename_axis('wine',axis=0)
print(reindexed.head())
```

Source Editor Object

arr

Help Variable Explorer Plots Files

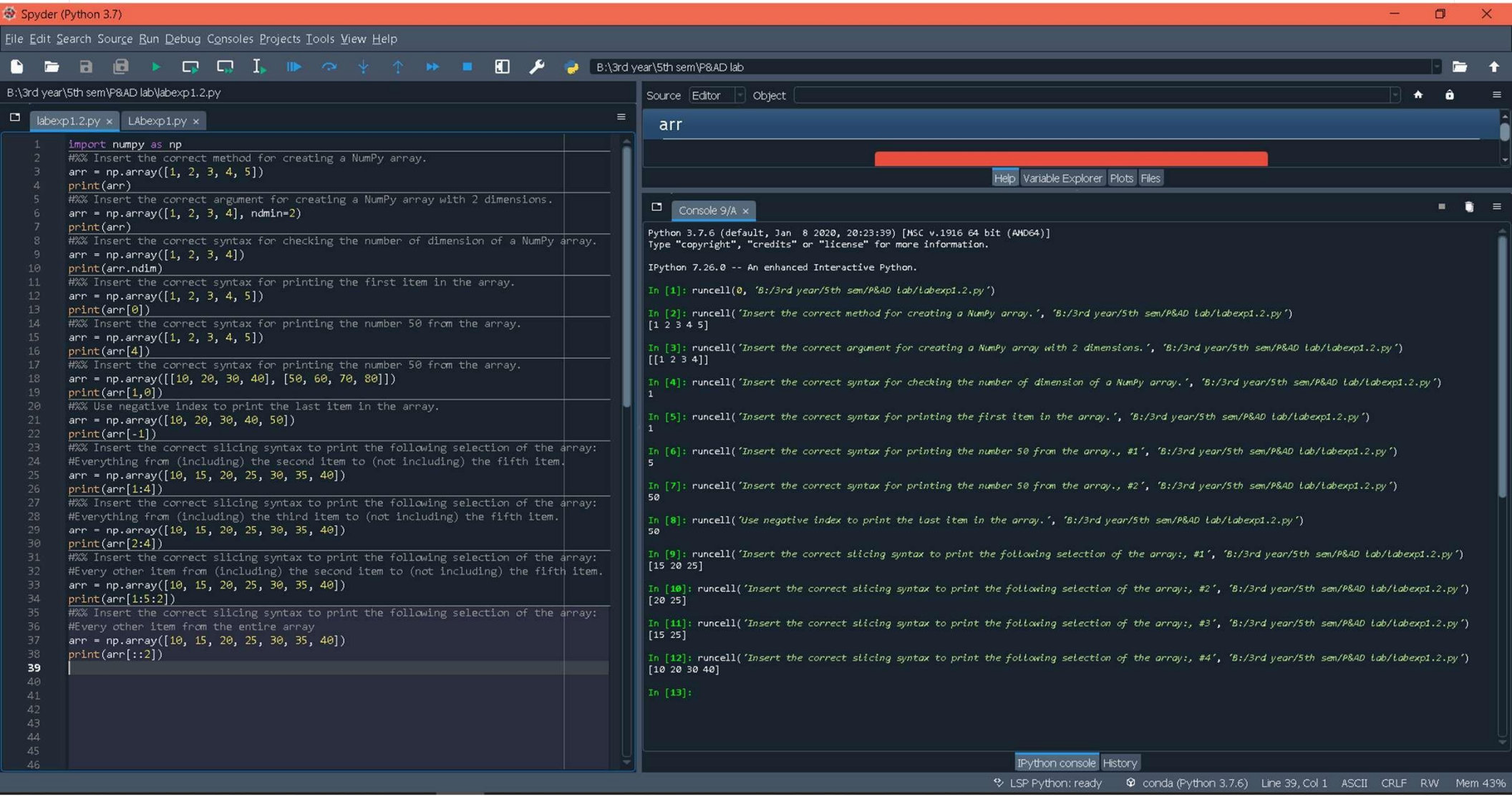
Console 10/A

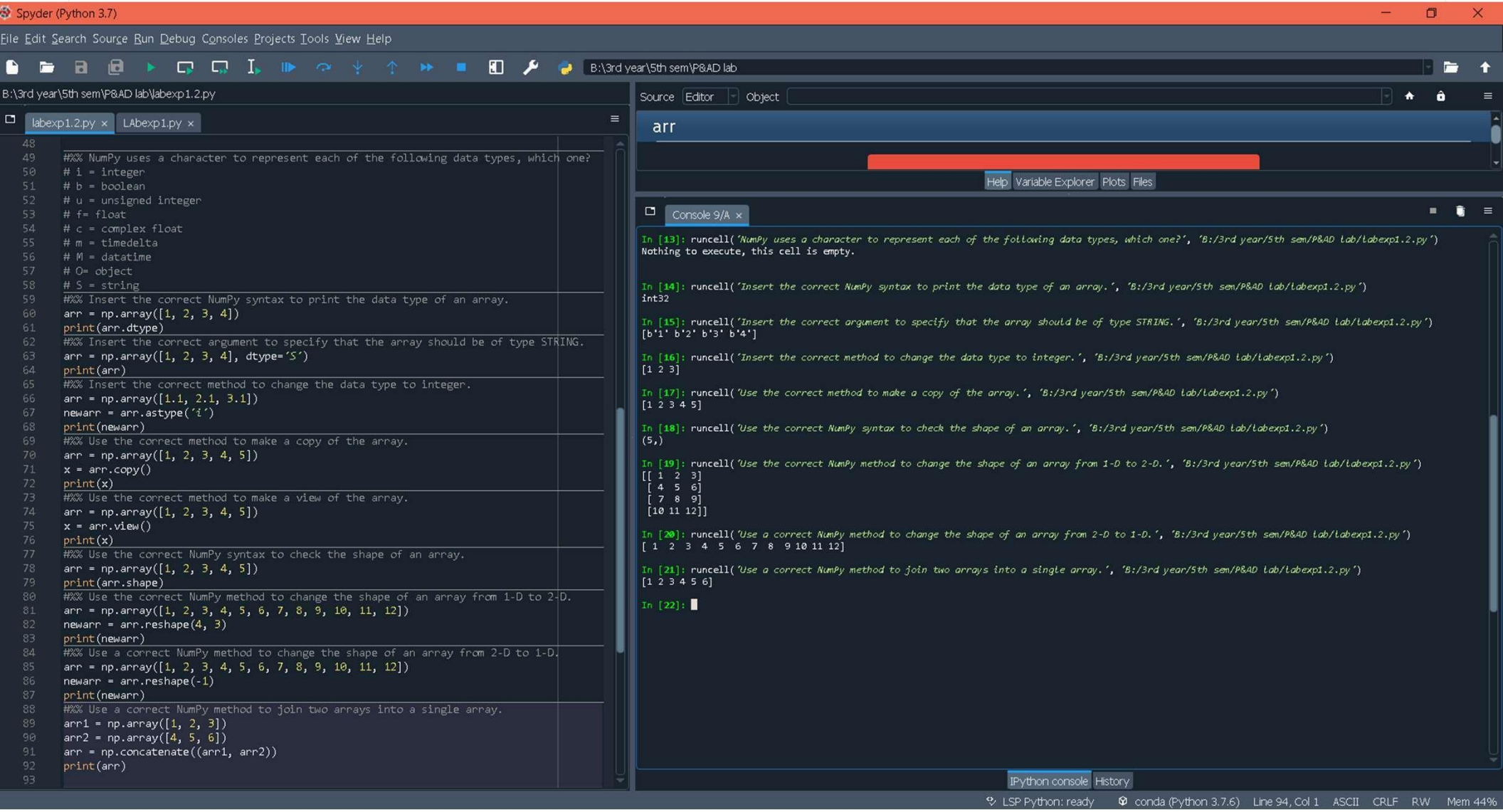
```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')

In [2]: runcell(6, 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')

In [3]: runcell('region_1 and region_2 are pretty uninformative names for locale columns in the dataset.
Create a copy of reviews with these columns renamed to region and locale, respectively.', 'B:/3rd year/5th
sem/P&AD lab/LAbexp1.py')
   Unnamed: 0 country  ...            variety                winery
0           0      US  ...  Cabernet Sauvignon                Heitz
1           1   Spain  ...       Tinta de Toro  Bodega Carmen Rodríguez
2           2      US  ...     Sauvignon Blanc              Macauley
3           3      US  ...          Pinot Noir                Ponzi
4           4  France  ...  Provence red blend  Domaine de la Bégude

[5 rows x 11 columns]

In [4]: runcell('Set the index name in the dataset to wines.', 'B:/3rd year/5th sem/P&AD lab/LAbexp1.py')
      Unnamed: 0 country  ...            variety                winery
wine                      ...
0              0      US  ...  Cabernet Sauvignon                Heitz
1              1   Spain  ...       Tinta de Toro  Bodega Carmen Rodríguez
2              2      US  ...     Sauvignon Blanc              Macauley
3              3      US  ...          Pinot Noir                Ponzi
4              4  France  ...  Provence red blend  Domaine de la Bégude

[5 rows x 11 columns]

In [5]:
```

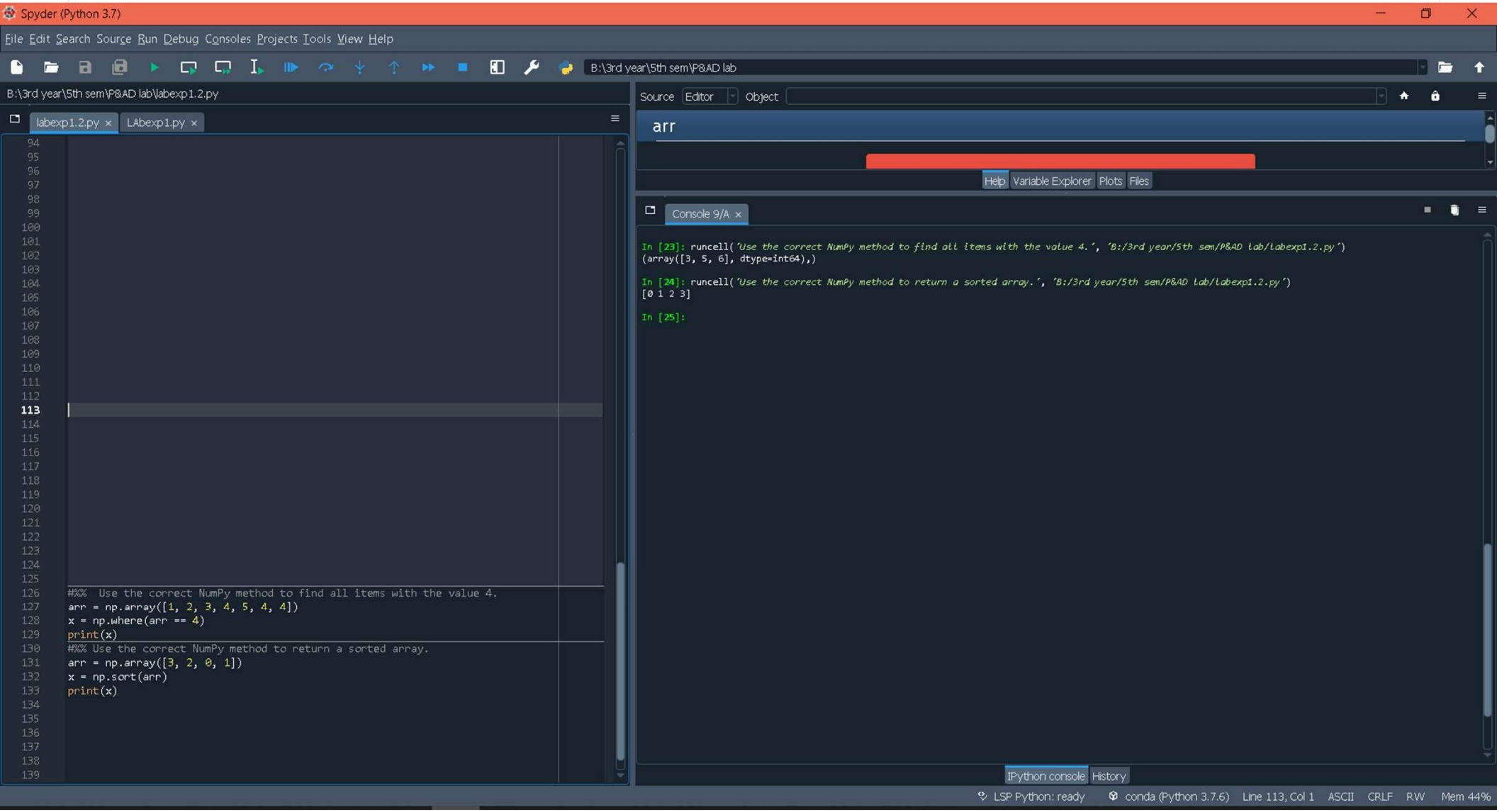
IPython console History

LSP Python: ready conda (Python 3.7.6) Line 97, Col 1 ASCII CRLF RW Mem 45%

NumPy


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
B:\3rd year\5th sem\P&AD lab
B:\3rd year\5th sem\P&AD lab\labexp1.2.py
labexp1.2.py
LAbexp1.py
import numpy as np
#%% Insert the correct method for creating a NumPy array.
arr = np.array([1, 2, 3, 4, 5])
print(arr)
#%% Insert the correct argument for creating a NumPy array with 2 dimensions.
arr = np.array([1, 2, 3, 4], ndmin=2)
print(arr)
#%% Insert the correct syntax for checking the number of dimension of a NumPy array.
arr = np.array([1, 2, 3, 4])
print(arr.ndim)
#%% Insert the correct syntax for printing the first item in the array.
arr = np.array([1, 2, 3, 4, 5])
print(arr[0])
#%% Insert the correct syntax for printing the number 50 from the array.
arr = np.array([1, 2, 3, 4, 5])
print(arr[4])
#%% Insert the correct syntax for printing the number 50 from the array.
arr = np.array([[10, 20, 30, 40], [50, 60, 70, 80]])
print(arr[1,0])
#%% Use negative index to print the last item in the array.
arr = np.array([10, 20, 30, 40, 50])
print(arr[-1])
#%% Insert the correct slicing syntax to print the following selection of the array:
#Everything from (including) the second item to (not including) the fifth item.
arr = np.array([10, 15, 20, 25, 30, 35, 40])
print(arr[1:4])
#%% Insert the correct slicing syntax to print the following selection of the array:
#Everything from (including) the third item to (not including) the fifth item.
arr = np.array([10, 15, 20, 25, 30, 35, 40])
print(arr[2:4])
#%% Insert the correct slicing syntax to print the following selection of the array:
#Every other item from (including) the second item to (not including) the fifth item.
arr = np.array([10, 15, 20, 25, 30, 35, 40])
print(arr[1:5:2])
#%% Insert the correct slicing syntax to print the following selection of the array:
#Every other item from the entire array
arr = np.array([10, 15, 20, 25, 30, 35, 40])
print(arr[::2])
Source Editor Object
arr
Help Variable Explorer Plots Files
Console 9/A
Python 3.7.6 (default, Jan 8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.26.0 -- An enhanced Interactive Python.
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
In [2]: runcell('Insert the correct method for creating a NumPy array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[1 2 3 4 5]
In [3]: runcell('Insert the correct argument for creating a NumPy array with 2 dimensions.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[[1 2 3 4]]
In [4]: runcell('Insert the correct syntax for checking the number of dimension of a NumPy array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
1
In [5]: runcell('Insert the correct syntax for printing the first item in the array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
1
In [6]: runcell('Insert the correct syntax for printing the number 50 from the array., #1', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
5
In [7]: runcell('Insert the correct syntax for printing the number 50 from the array., #2', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
50
In [8]: runcell('Use negative index to print the last item in the array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
50
In [9]: runcell('Insert the correct slicing syntax to print the following selection of the array:, #1', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[15 20 25]
In [10]: runcell('Insert the correct slicing syntax to print the following selection of the array:, #2', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[20 25]
In [11]: runcell('Insert the correct slicing syntax to print the following selection of the array:, #3', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[15 25]
In [12]: runcell('Insert the correct slicing syntax to print the following selection of the array:, #4', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[10 20 30 40]
In [13]:
IPython console History
LSP Python: ready
conda (Python 3.7.6)
Line 39, Col 1
ASCII
CRLF
RW
Mem 43%


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
B:\3rd year\5th sem\P&AD lab
B:\3rd year\5th sem\P&AD lab\labexp1.2.py
labexp1.2.py
LAbexp1.py
#%% NumPy uses a character to represent each of the following data types, which one?
i = integer
b = boolean
u = unsigned integer
f= float
c = complex float
m = timedelta
M = datatime
O= object
S = string
#%% Insert the correct NumPy syntax to print the data type of an array.
arr = np.array([1, 2, 3, 4])
print(arr.dtype)
#%% Insert the correct argument to specify that the array should be of type STRING.
arr = np.array([1, 2, 3, 4], dtype='S')
print(arr)
#%% Insert the correct method to change the data type to integer.
arr = np.array([1.1, 2.1, 3.1])
newarr = arr.astype('i')
print(newarr)
#%% Use the correct method to make a copy of the array.
arr = np.array([1, 2, 3, 4, 5])
x = arr.copy()
print(x)
#%% Use the correct method to make a view of the array.
arr = np.array([1, 2, 3, 4, 5])
x = arr.view()
print(x)
#%% Use the correct NumPy syntax to check the shape of an array.
arr = np.array([1, 2, 3, 4, 5])
print(arr.shape)
#%% Use the correct NumPy method to change the shape of an array from 1-D to 2-D.
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(4, 3)
print(newarr)
#%% Use a correct NumPy method to change the shape of an array from 2-D to 1-D.
arr = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12])
newarr = arr.reshape(-1)
print(newarr)
#%% Use a correct NumPy method to join two arrays into a single array.
arr1 = np.array([1, 2, 3])
arr2 = np.array([4, 5, 6])
arr = np.concatenate((arr1, arr2))
print(arr)
Source Editor Object
arr
Help Variable Explorer Plots Files
Console 9/A
In [13]: runcell('NumPy uses a character to represent each of the following data types, which one?', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
Nothing to execute, this cell is empty.
In [14]: runcell('Insert the correct NumPy syntax to print the data type of an array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
int32
In [15]: runcell('Insert the correct argument to specify that the array should be of type STRING.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[b'1' b'2' b'3' b'4']
In [16]: runcell('Insert the correct method to change the data type to integer.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[1 2 3]
In [17]: runcell('Use the correct method to make a copy of the array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[1 2 3 4 5]
In [18]: runcell('Use the correct NumPy syntax to check the shape of an array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
(5,)
In [19]: runcell('Use the correct NumPy method to change the shape of an array from 1-D to 2-D.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[[1 2 3]
[4 5 6]
[7 8 9]
[10 11 12]]
In [20]: runcell('Use a correct NumPy method to change the shape of an array from 2-D to 1-D.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[1 2 3 4 5 6 7 8 9 10 11 12]
In [21]: runcell('Use a correct NumPy method to join two arrays into a single array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[1 2 3 4 5 6]
In [22]:
IPython console History
LSP Python: ready
conda (Python 3.7.6)
Line 94, Col 1
ASCII
CRLF
RW
Mem 44%


Spyder (Python 3.7)
File Edit Search Source Run Debug Consoles Projects Tools View Help
B:\3rd year\5th sem\P&AD lab
B:\3rd year\5th sem\P&AD lab\labexp1.2.py
labexp1.2.py
LAbexp1.py
#%% Use the correct NumPy method to find all items with the value 4.
arr = np.array([1, 2, 3, 4, 5, 4, 4])
x = np.where(arr == 4)
print(x)
#%% Use the correct NumPy method to return a sorted array.
arr = np.array([3, 2, 0, 1])
x = np.sort(arr)
print(x)
Source Editor Object
arr
Help Variable Explorer Plots Files
Console 9/A
In [23]: runcell('Use the correct NumPy method to find all items with the value 4.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
(array([3, 5, 6], dtype=int64),)
In [24]: runcell('Use the correct NumPy method to return a sorted array.', 'B:/3rd year/5th sem/P&AD lab/labexp1.2.py')
[0 1 2 3]
In [25]:
IPython console History
LSP Python: ready
conda (Python 3.7.6)
Line 113, Col 1
ASCII
CRLF
RW
Mem 44%

Experiment 3

AIM:

To perform Linear Regression (OLS Model)

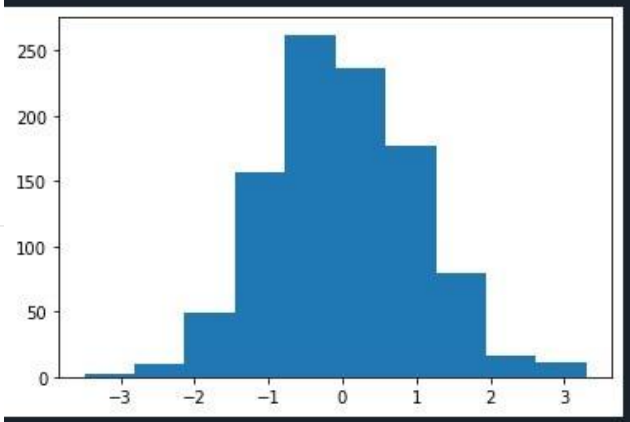
CODE:

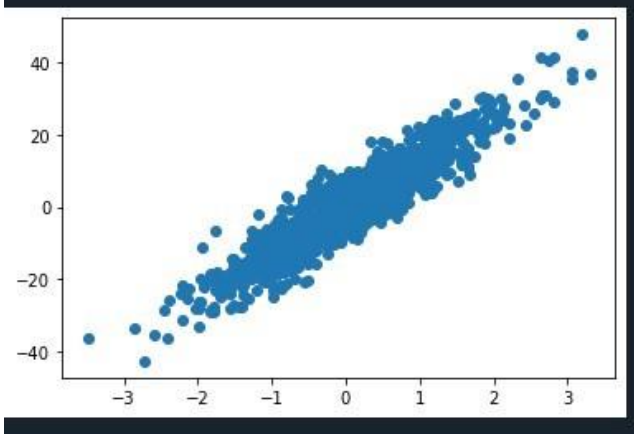
```
#%%
# import required libraries
from sklearn.datasets import make_regression
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error ,mean_absolute_error
import numpy as np
import seaborn as sns
from math import sqrt
import warnings
warnings.filterwarnings('ignore')
#%%
#generate dataset for regression
X,y = make_regression(n_samples=1000, n_features = 1, shuffle = True, noise = 5,random_state=86)
#%%
#Statistics
stdX = np.std(X)
meanX = np.mean(X)

stdY = np.std(y)
meanY = np.mean(y)
print(meanX)
print(stdX)
print(meanY)
print(stdY)
#%%
# plot generated data
plt.hist(X)
plt.show()
#%%
plt.scatter(X,y)
plt.show()
#%%
# As mean of this dataset is already near to 0, so standard scaling is not required
#%%
# Preprocessing
#%%
plt.hist(X)
plt.show()
#%%
#Plot for data after preprocessing
plt.scatter(X,y)
plt.show()
#%%
# Train Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, random_state=42)
#%%
sns.distplot(y_train)
plt.show()
#%%
sns.distplot(y_test)
plt.show()
#%%
#Performing Linear Regression || Using Traditional Method
```

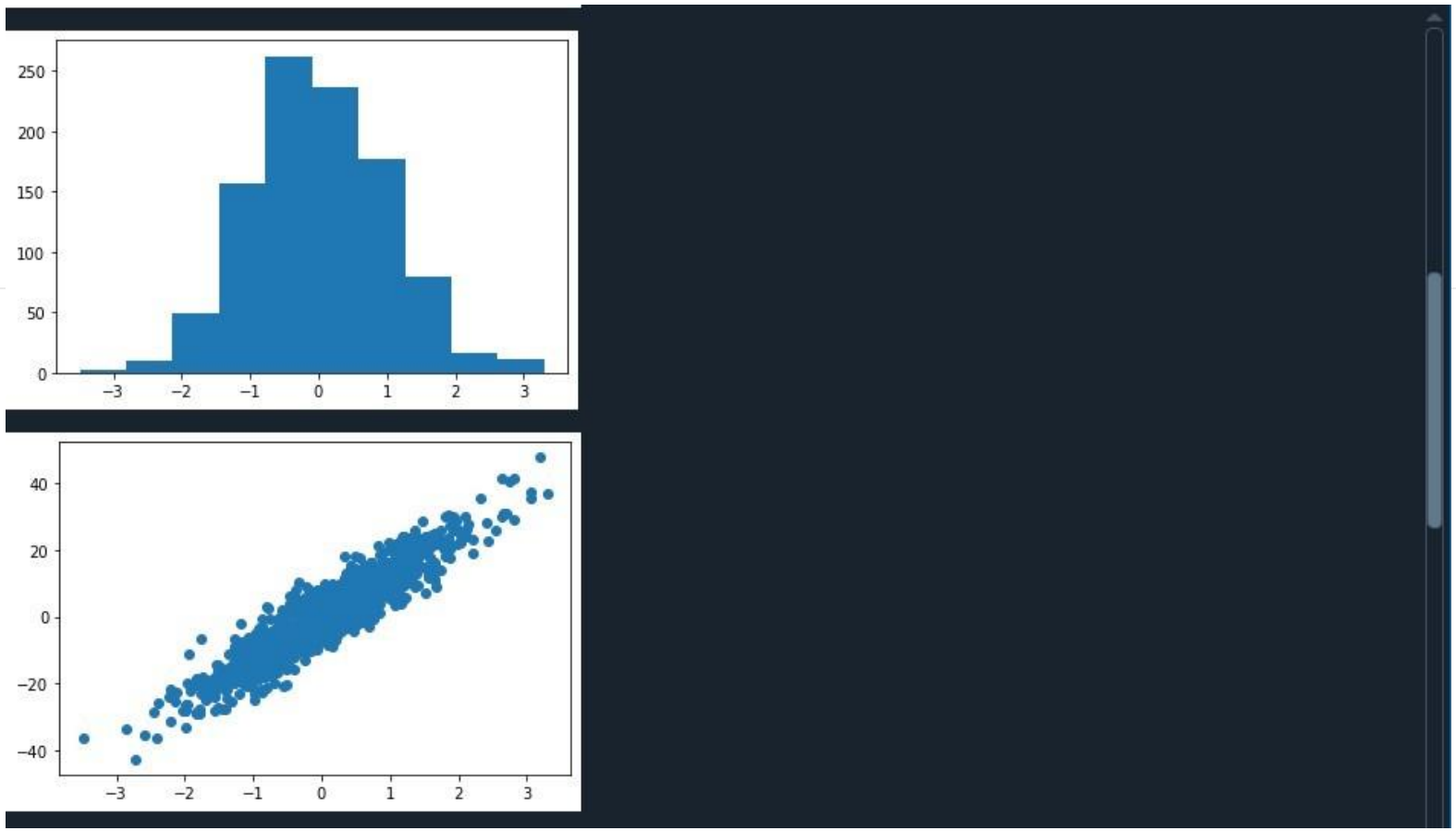
```
#Initializing variables
n = len(X_train)
num = 0
denom = 0
#%%
#Calculating slope & intercept
for i in range(n):
    num += (X_train[i]-meanX) * (y_train[i]-meanY)
    denom += (X_train[i] - meanX)**2
m = num/denom
c = meanY - (m*meanX)
print(m, ', ', c)

min_X = 1 - np.min(X_train)
max_X = 1 - np.max(X_train)
print(min_X, ', ', max_X)
#%%
#calculating the values of x & y
x = np.linspace(min_X, max_X,1000)
y = (m*x)+c
#%%  Plot Regression Line
plt.scatter(X_train,y_train,color='b')
plt.plot(x,y,color='r')
plt.title('Simple Linear Regression')
plt.xlabel('X')
plt.ylabel('Y')
#%%
#calculating error
y_pred = []
sum_pred = 0
sum_actual = 0
for i in range(len(X_test)):
    val = (m*X_test[i]+c)
    y_pred.append(val)
mse = mean_squared_error(y_test, y_pred)
print("MSE", mse)
r2 = sqrt(mse)
print("R2",r2)
mae = mean_absolute_error(y_test, y_pred)
print("MAE", mae)
#%%
# Perform Linear Regression || Using sklearn
# Build Model
LRmodel = LinearRegression()
LRmodel.fit(X_train, y_train)
#%%
# Error Calculation
print("-------------------------------------")
y_pred1 = LRmodel.predict(X_test)
mse1 = mean_squared_error(y_test, y_pred1)
print("MSE with sklearn", mse1)
r2new = sqrt(mse1)
print("R2 with sklearn",r2new)
mae1 = mean_absolute_error(y_test, y_pred1)
print("MAE with sklearn", mae1)
```

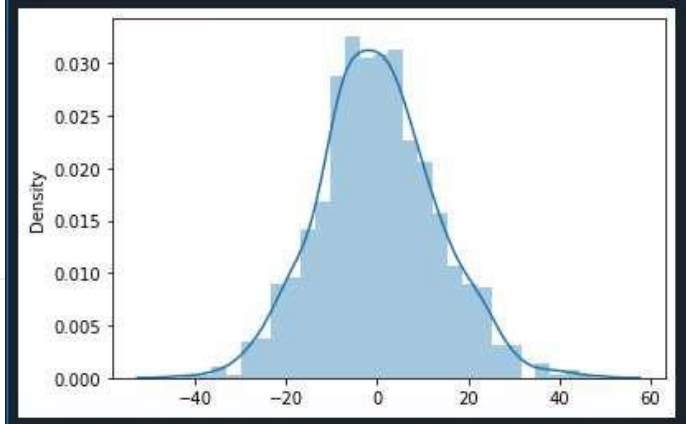
Output:

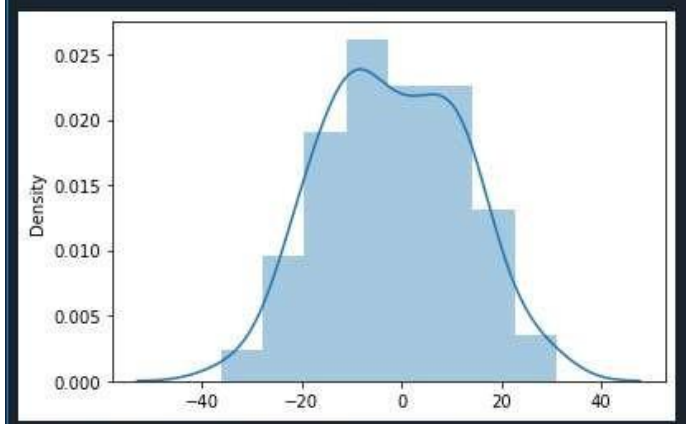
```
In [1]: runfile('B:/3rd year/5th sem/P&AD/LinearRegression.py', wdir='B:/3rd year/5th sem/P&AD')
0.007610682926160853
0.9942474804457474
-0.05835119770320271
13.08071673806697
```



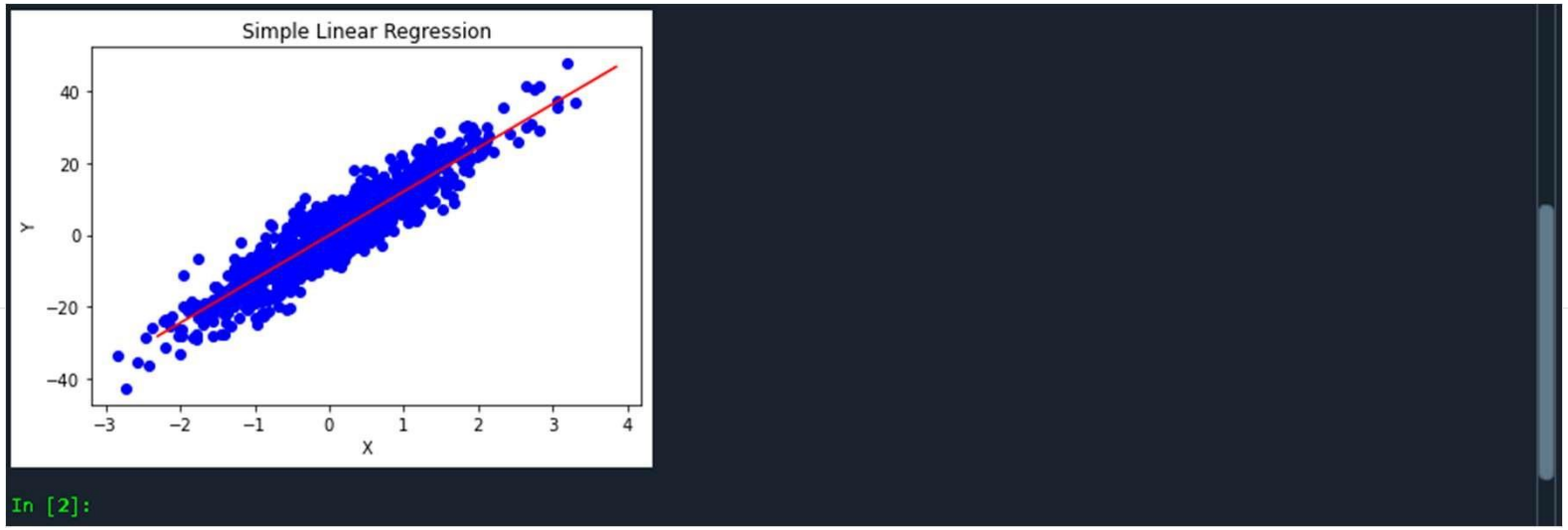



250
200
150
100
50
0
−3
−2
−1
0
1
2
3
40
20
0
−20
−40
−3
−2
−1
0
1
2
3






```
[12.21409321] , [-0.15130879]
3.8487665053713127 , -2.303677618501437
MSE 20.304967238809418
R2 4.506103332016413
MAE 3.611272677585965
-------------------------------------
MSE with sklearn 20.33856770371379
R2 with sklearn 4.509830119163447
MAE with sklearn 3.610421754936301
```




```
In [2]:
```

Experiment 4

AIM:

To perform Logistic Regression

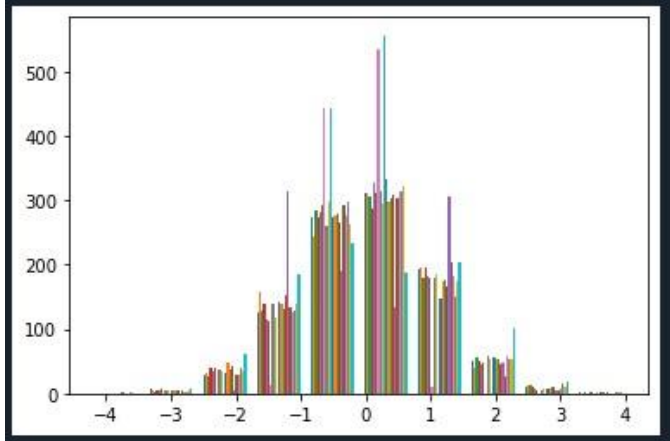
CODE:

```
"""
@author: Dhruv Singhal
"""

#%%
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
import seaborn as sns
import pandas as pd
from sklearn.metrics import confusion_matrix
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
#%%

#generate dataset
X,y = make_classification(n_samples=1000,n_classes=2)
X=pd.DataFrame(X)
y=pd.Series(y,name=20)
print("X values are:",X.head())
print("Y values are:",y.head())

#%% Visualization

plt.hist(X)
plt.show()

#%%
sns.distplot(y)
plt.show()

#%%
sns.distplot(X)
plt.show()

#%% feature extractionpreprocessing, correlation matrix
_,graph=plt.subplots(figsize=(15,10))
sns.heatmap(X.corr(),annot=True,ax=graph,square=True)
plt.show()

#%%
df=pd.merge(X,y,right_index=True,left_index=True)
print(df.head())
print(df.corr()[[20]].abs().sort_values(by=20,ascending=False))
```

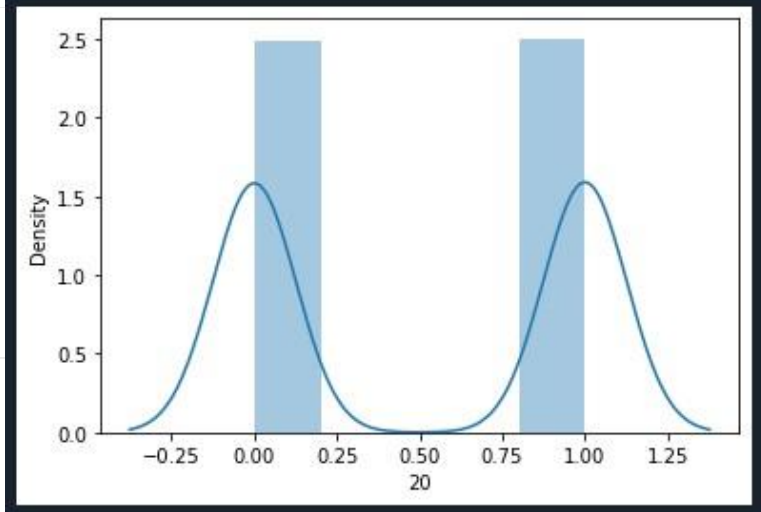
```
#%% columns with high correlation will be dropped
# creating useful data
datasetX=df[[1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19]]
datasetY=df[20]
print(datasetY.head())

#%%
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.15,random_state=42)

model=LogisticRegression()
model.fit(X_train,Y_train)
print(model.classes_)

Y_pred=model.predict(X_test)
print(Y_pred)

print("train Accuracy:",model.score(X_train,Y_train))
print("test Accuracy:",model.score(X_test,Y_test))

print(confusion_matrix(Y_test,Y_pred))
```

OUTPUT:

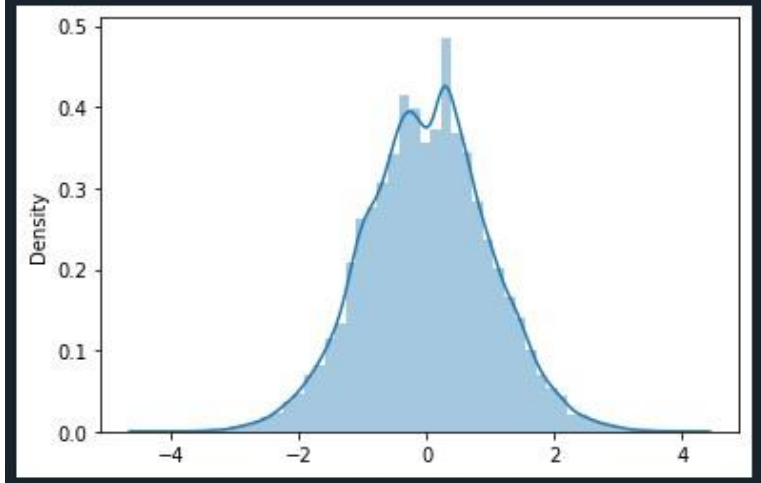
```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: runcell(0, 'C:/Users/Dhruv Singhal/untitled0.py')

In [2]: runcell(1, 'C:/Users/Dhruv Singhal/untitled0.py')

In [3]: runcell(2, 'C:/Users/Dhruv Singhal/untitled0.py')
X values are:          0         1         2   ...        17        18        19
0 -1.259436 -0.372492  0.054760  ... -0.035846 -1.462309 -1.140185
1 -0.800033  0.478840 -0.629627  ...  0.170002  0.445090 -1.299911
2 -0.770212 -0.741631  1.075089  ...  0.075282 -1.240537 -0.914384
3  1.583047 -0.705226  0.710976  ... -1.260828  0.947186 -0.973737
4  0.431166  0.902277  2.205464  ...  0.333836  0.417141  1.348560

[5 rows x 20 columns]
Y values are: 0    0
1    0
2    1
3    0
4    1
Name: 20, dtype: int32

In [4]: runcell('Visualization', 'C:/Users/Dhruv Singhal/untitled0.py')
```



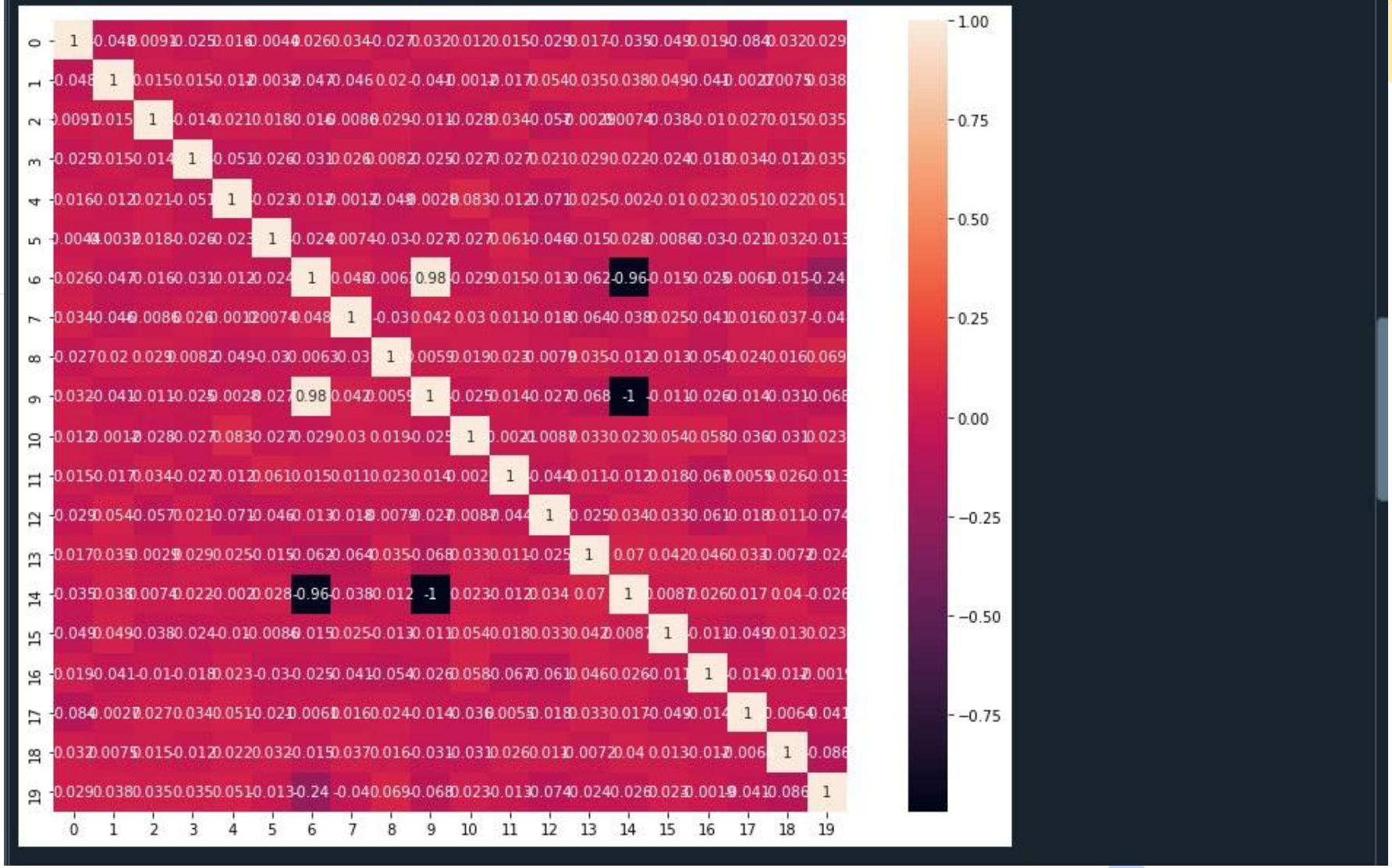
```
In [5]: runcell(4, 'C:/Users/Dhruv Singhal/untitled0.py')
```




```
In [6]: runcell(5, 'C:/Users/Dhruv Singhal/untitled0.py')
```



```
In [7]: runcell('feature extractionpreprocessing, correlation matrix', 'C:/Users/Dhruv Singhal/untitled0.py')
```



```
In [8]: runcell(7, 'C:/Users/Dhruv Singhal/untitled0.py')
          0         1         2         3   ...        17        18        19  20
0 -1.259436 -0.372492  0.054760 -2.045250  ... -0.035846 -1.462309 -1.140185   0
1 -0.800033  0.478840 -0.629627 -0.773224  ...  0.170002  0.445090 -1.299911   0
2 -0.770212 -0.741631  1.075089 -1.506821  ...  0.075282 -1.240537 -0.914384   1
3  1.583047 -0.705226  0.710976 -1.241509  ... -1.260828  0.947186 -0.973737   0
4  0.431166  0.902277  2.205464 -0.400889  ...  0.333836  0.417141  1.348560   1

[5 rows x 21 columns]
           20
20  1.000000
14  0.907235
9   0.906191
6   0.882873
13  0.067238
12  0.050728
1   0.046343
15  0.036825
3   0.032795
7   0.029817
0   0.029335
11  0.028842
16  0.026296
18  0.018547
5   0.018465
4   0.015409
10  0.011313
19  0.007832
17  0.005624
8   0.001292
2   0.000811

In [9]: runcell('columns with high correlation will be dropped', 'C:/Users/Dhruv Singhal/untitled0.py')
0    0
1    0
2    1
3    0
4    1
Name: 20, dtype: int32
```

```
In [10]: runcell(9, 'C:/Users/Dhruv Singhal/untitled0.py')
[0 1]
[1 0 1 0 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 0 0 0 1 0 1 1 1 0 0 0 0 1 1 1 1 0
 0 1 1 0 1 1 0 1 0 0 1 1 0 0 1 1 1 1 0 0 0 0 0 1 1 1 1 1 0 1 0 0 0 1 0 0 0
 0 0 1 0 1 0 0 0 0 1 1 0 0 0 1 0 1 1 1 0 1 1 1 0 0 0 0 1 1 1 0 0 0 1 0 1 0
 0 0 0 0 1 1 0 0 1 0 0 0 1 0 1 1 0 1 1 0 1 0 1 0 1 0 0 0 0 1 0 1 0 0 0 1 1
 0 0]
train Accuracy: 0.9776470588235294
test Accuracy: 0.9733333333333334
[[82  0]
 [ 4 64]]

In [11]:
```

Experiment 5

```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.datasets import make_regression

#%%
# generate  dataset
X,y = make_regression(n_samples=100, n_features=5,noise=50)

#%%
#plot the graph
plt.scatter(X[:,2],y)

#%%
#Train test Split
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3) #30%test and 70% train dataset.

#%%
#implement Polynomial Regression
poly = PolynomialFeatures(degree = 3)  # degree of polynomial=3
X_poly = poly.fit_transform(X_train)
```

```
#%%
#implement through ridgeCV
from sklearn.linear_model import RidgeCV
model01=RidgeCV(alphas=[1e-3,1e-2,1e-1,1])
model01.fit(X_poly,y_train)

#%%
#prediction
y_pred_rid=model01.predict(X_poly_test)

#%%
#calculate rmse ,log error and R^2 score
rms_rid=mean_squared_error(y_test, y_pred_rid)
print("RMSE Error: ",rms_rid)
log_rid=mean_squared_log_error(abs(y_test), abs(y_pred_rid))
print("Log Error: ",log_rid)
r2_sc_ridge=r2_score(y_test,y_pred_rid)
print("R^2 Score: ",r2_sc_ridge)

#%%
#implement cross validation model
from sklearn.model_selection import KFold,cross_val_score
folds = KFold(n_splits = 5, shuffle = True, random_state = 100)
scores = cross_val_score(model, X_train, y_train, scoring='neg_mean_absolute_error', cv=folds)
print(np.average(scores) )
```

Experiment 7

```
# -*- coding: utf-8 -*-
"""
Created on Mon Sep 20 15:33:41 2021

@author: Dhruv Singhal
"""

#%%
import numpy as np
import matplotlib.pyplot  as plt
#import pandas as pd
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#%%
from sklearn.datasets import make_regression
x,y=make_regression(n_samples=10000,n_features=5,noise=30)
sns.distplot(y)

#%%
plt.hist(x)
#%%
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,train_size=0.15,random_state=42)
from sklearn.metrics import r2_score,mean_squared_error
#%%
from sklearn.linear_model import LinearRegression
lin=LinearRegression()
lin.fit(x_train,y_train)
y_pred=lin.predict(x_test)
print("r2 score without tuning",r2_score(y_test,y_pred))
print("RMSE without tuning",np.sqrt(mean_squared_error(y_test,y_pred)))

#%%  Hyperparameter Tuning From Here
#%%
from sklearn.model_selection import GridSearchCV
tuned_parameters = [{'fit_intercept': ['True'], 'normalize': ['True']},
                    {'fit_intercept': ['False'], 'normalize': ['True']},
                    {'fit_intercept': ['True'], 'normalize': ['False']},
                    {'fit_intercept': ['False'], 'normalize': ['False']}

                    ]
clf=GridSearchCV(LinearRegression(),tuned_parameters,scoring=('r2'))
clf.fit(x_train,y_train)
```

```
print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Best Score:",clf.best_score_)
z=clf.cv_results_

#%% kfold cross validation with hyperparameter tuning
from sklearn.model_selection import KFold
k = 5
kf = KFold(n_splits=k, random_state=None)
model = GridSearchCV(LinearRegression(),tuned_parameters,scoring=('r2'))

for train_index , test_index in kf.split(x):
    X_train , X_test = x[train_index,:],x[test_index,:]
    y_train , y_test = y[train_index] , y[test_index]

    model.fit(X_train,y_train)
    pred_values = model.predict(X_test)
print("Best parameters set found on development set:")
print()
print(model.best_params_)
print()
print("Best Score:",model.best_score_)
z2=model.cv_results_
```

Experiment 6

CODE:

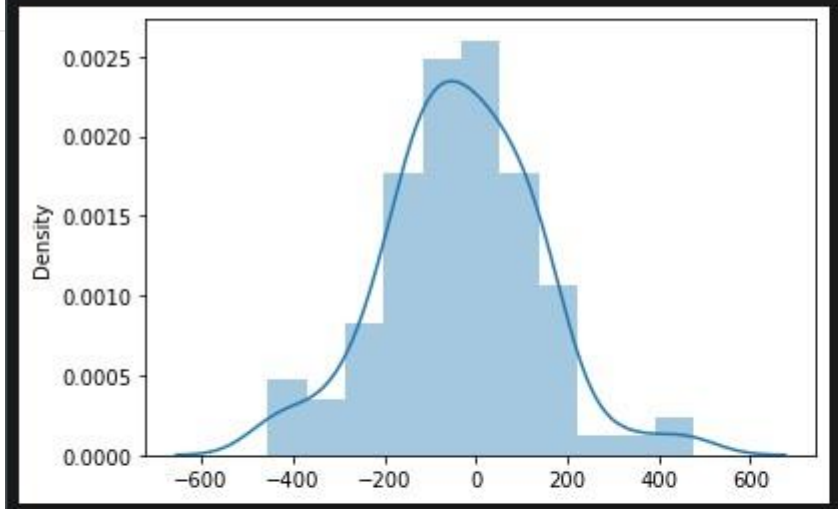
```python
# -*- coding: utf-8 -*-
"""

@author: Dhruv Singhal
"""

#%%
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import make_regression
#%%
X,y=make_regression(n_samples=100,n_features=10,n_informative=4,random_state=0)
sns.distplot(y)

#%%
from sklearn.decomposition import PCA
pca=PCA(n_components=5)
principalComponents=pca.fit_transform(X)
X_new=pd.DataFrame(data=principalComponents,columns=('PC1','PC2','PC3','PC4','PC5'))
print(X_new.head())
#%%
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
X_train , X_test , Y_train , Y_test = train_test_split(X_new, y, train_size=0.20,random_state=0)
lin=LinearRegression()
model=lin.fit(X_train,Y_train)
y_pred=model.predict(X_test)
#%%
print('Coefficients: ', model.coef_)
print('Variance score: {}'.format(model.score(X_test, Y_test)))
print("Test Score",model.score(X_train,Y_train))
print("Test Score",model.score(X_test,Y_test))
#%%
#training
plt.scatter(model.predict(X_train),model.predict(X_train) - Y_train,color = "green", s = 10, label = 'Train data')
#testing
plt.scatter(model.predict(X_test),model.predict(X_test) - Y_test,color = "orange", s = 10, label = 'Testing data')


#%%
import math
from sklearn.metrics import mean_squared_error
print("RMSE",math.sqrt(mean_squared_error(Y_test,y_pred)))
from sklearn.metrics import r2_score
print("R^2",r2_score(Y_test,y_pred))


#%%
print(np.max(Y_train),np.min(Y_train))

#%%
plt.figure(figsize=(20,10))
plt.plot(y_pred)
plt.plot(Y_test)
```

```
57
58
59
60
61  #%%
62  X_train2 , X_test2 , Y_train2 , Y_test2 = train_test_split(X, y, train_size=0.20,random_state=0)
63  lin=LinearRegression()
64  model2=lin.fit(X_train2,Y_train2)
65  y_pred2=model.predict(X_test)
66  #%%
67  print('Coefficients: ', model2.coef_)
68  print('Variance score: {}'.format(model2.score(X_test2, Y_test2)))
69  print("Test Score",model2.score(X_train2,Y_train2))
70  print("Test Score",model2.score(X_test2,Y_test2))
71
72  #%%
73  import math
74  from sklearn.metrics import mean_squared_error
75  print("RMSE",math.sqrt(mean_squared_error(Y_test2,y_pred2)))
76  from sklearn.metrics import r2_score
77  print("R^2",r2_score(Y_test2,y_pred2))
78
79
80  #%%
81  print(np.max(Y_train2),np.min(Y_train2))
82
83  #%%
84  plt.figure(figsize=(20,10))
85  plt.plot(y_pred2)
86  plt.plot(Y_test2)
87
88
89
90
```

OUTPUT:

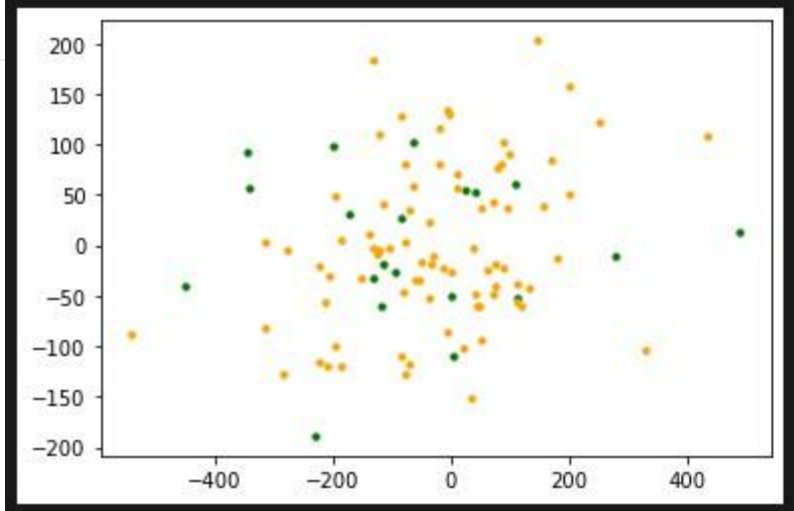
Console 1/A ×

```
Python 3.7.6 (default, Jan  8 2020, 20:23:39) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/PCA.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/PCA.py')

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/PCA.py')
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\seaborn\distributions.py:2619: FutureWarning: `distplot` is a deprecated
function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```



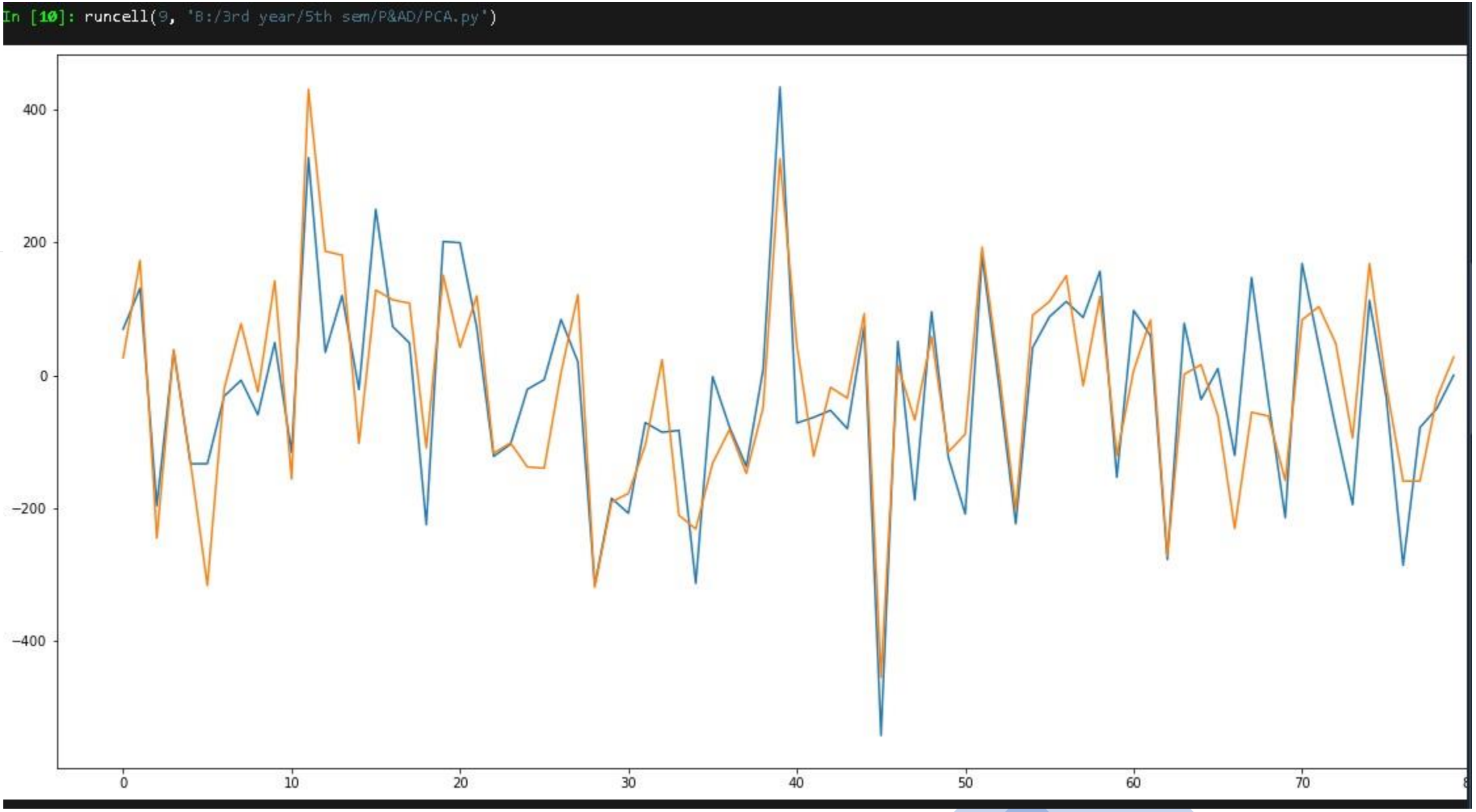

```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/PCA.py')
        PC1       PC2       PC3       PC4       PC5
0  2.689187  0.188731  0.553922  0.677884  0.069860
1 -2.718170 -2.513221  0.564428  0.308534  0.148173
2 -0.150486  3.214459 -0.133615 -1.756744  0.593628
3 -1.597885 -2.291277 -0.211204 -0.593296 -0.201198
4 -0.621526 -0.492717  0.588593  0.407506 -0.956159

In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/PCA.py')
```

```
In [6]: runcell(5, 'B:/3rd year/5th sem/P&AD/PCA.py')

Output from spyder call 'get_namespace_view':
Coefficients:  [-14.02798066  29.78262828 -44.53078748 120.20573166 -94.01743173]
Variance score: 0.715357937146422
Test Score 0.8948220992728023
Test Score 0.715357937146422

In [7]: runcell(6, 'B:/3rd year/5th sem/P&AD/PCA.py')
```



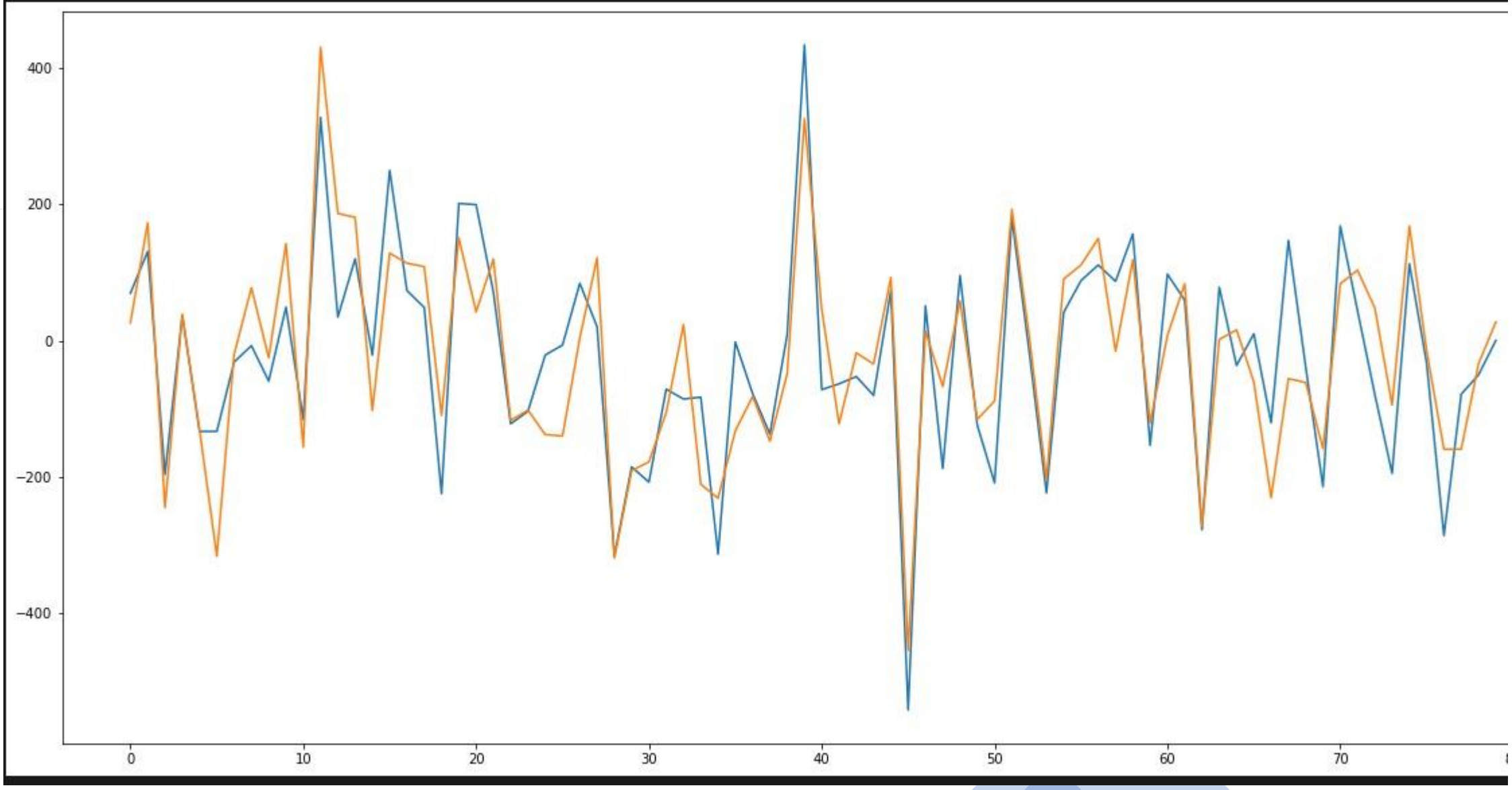

```
In [8]: runcell(7, 'B:/3rd year/5th sem/P&AD/PCA.py')
RMSE 79.41012724129493
R^2 0.715357937146422

In [9]: runcell(8, 'B:/3rd year/5th sem/P&AD/PCA.py')
476.0496266907876 -436.9209738094973

In [10]: runcell(9, 'B:/3rd year/5th sem/P&AD/PCA.py')
```

```
In [10]: runcell(9, 'B:/3rd year/5th sem/P&AD/PCA.py')
```



```
In [11]: runcell(10, 'B:/3rd year/5th sem/P&AD/PCA.py')

In [12]: runcell(11, 'B:/3rd year/5th sem/P&AD/PCA.py')
Coefficients:  [-1.05459713e-14  9.83472758e-14 -7.33134857e-14  7.00528623e+01
  8.83077597e+01  9.66575107e+01  8.21903908e+01  3.77754737e-14
  4.10586745e-14 -4.61401623e-14]
Variance score: 1.0
Test Score 1.0
Test Score 1.0

In [13]: runcell(12, 'B:/3rd year/5th sem/P&AD/PCA.py')
RMSE 79.41012724129493
R^2 0.715357937146422

In [14]: runcell(13, 'B:/3rd year/5th sem/P&AD/PCA.py')
476.0496266907876 -436.9209738094973

In [15]: runcell(14, 'B:/3rd year/5th sem/P&AD/PCA.py')
```

```
In [15]: runcell(14, 'B:/3rd year/5th sem/P&AD/PCA.py')
```



Experiment 8

Hyperparameter

Tuning and Nested Cross validation on

Linear Regression

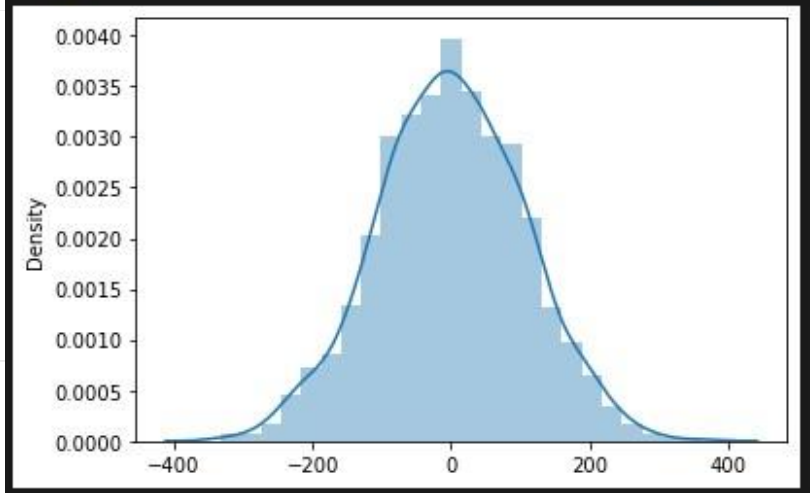
```
import numpy as np
import matplotlib.pyplot  as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
#%%
from sklearn.datasets import make_regression
x,y=make_regression(n_samples=1000,n_features=5,noise=20)
sns.distplot(y)

#%%
plt.hist(x)
#%%
x_train = x
y_train =y

#%%
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.model_selection import KFold
from sklearn.linear_model import Ridge
NUM_TRIALS = 30


tuned_parameters = [{'solver' : ['svd', 'lsqr'],'fit_intercept': ['True'],'normalize': ['False']},
                    {'solver' : ['sag', 'cholesky'],'fit_intercept': ['False'],'normalize': ['true']}]

score = 'r2'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)

# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LinearRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = Ridge(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= Ridge(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()


score_difference = non_nested_scores - nested_scores

print("Average difference of {:6f} with std. dev. of {:6f}."
      .format(score_difference.mean(), score_difference.std()))
```

CODE:

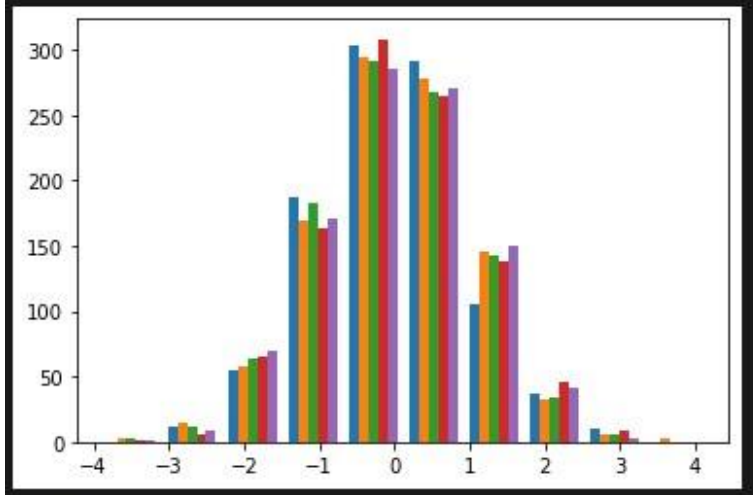
OUTPUT:

Console 3/A

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp9.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp9.py')
```




```
In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/exp9.py')
```




```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/exp9.py')

In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/exp9.py')
Average difference of -0.003284 with std. dev. of 0.001567.
```

Hyperparameter Tuning and Nested Cross validation on

Logistic Regression

```python
import numpy as np
from sklearn.datasets import make_classification
from sklearn import linear_model
#from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
#%%
# Generating Data
X, y = make_classification(n_samples = 1000, n_features = 5, n_classes = 2)
x_train = X
y_train = y
#%%
NUM_TRIALS = 30

tuned_parameters = [{ 'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
 'C' : np.logspace(-4, 4, 20),
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']
   }]

score = 'accuracy'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LogisticRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = linear_model.LogisticRegression(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    print(model.best_params_)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= linear_model.LogisticRegression(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()
score_difference = non_nested_scores - nested_scores
print("Average difference of {:6f} with std. dev. of {:6f}.".format(score_difference.mean(), score_difference.std()))

#%%
```

CODE:

OUTPUT:

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/LogisticTuning.py')
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
{'C': 0.2335721469090l212, 'penalty': 'l1', 'solver': 'liblinear'}
Average difference of 0.001900 with std. dev. of 0.001719.
```

Hyperparameter

Tuning and Nested Cross validation on

Logistic Regression

```python
import numpy as np
#from sklearn.datasets import make_classification
from sklearn import linear_model
#from matplotlib import pyplot as plt
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
import warnings
warnings.filterwarnings('ignore')
#%%
# Generating Data
from sklearn.datasets import load_breast_cancer

data = load_breast_cancer()

X = data.data

y = data.target
x_train = X
y_train = y
#%%
NUM_TRIALS = 30


tuned_parameters = [{ 'penalty' : ['l1', 'l2', 'elasticnet', 'none'],
    'solver' : ['lbfgs','newton-cg','liblinear','sag','saga']
   }]
score = 'accuracy'
non_nested_scores = np.zeros(NUM_TRIALS)
nested_scores = np.zeros(NUM_TRIALS)
# Loop for each trial
for i in range(NUM_TRIALS):

    # model= GridSearchCV(linear_model.LogisticRegression(), tuned_parameters, scoring= score)
    inner_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    outer_cv = KFold(n_splits=4, shuffle=True, random_state=i)
    model= GridSearchCV(estimator = linear_model.LogisticRegression(), param_grid = tuned_parameters, scoring = score)
    model.fit(x_train, y_train)
    print(model.best_params_)
    non_nested_scores[i] = model.best_score_


    # Nested CV with parameter optimization
    model = GridSearchCV(estimator= linear_model.LogisticRegression(), param_grid = tuned_parameters, cv=inner_cv, scoring= score)
    nested_score = cross_val_score(model, X=x_train, y=y_train, cv=outer_cv)
    nested_scores[i] = nested_score.mean()
score_difference = non_nested_scores - nested_scores
print("Average difference of {:6f} with std. dev. of {:6f}.".format(score_difference.mean(), score_difference.std()))
```

(Breast Cancer Dataset)

CODE:

OUTPUT:

```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/Dataset(Logistic_Hyper_Tuning).py')
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
{'penalty': 'none', 'solver': 'newton-cg'}
Average difference of 0.009804 with std. dev. of 0.006161.
```

Experiment 9

CODE:

```python
import matplotlib.pyplot as plt
import numpy as np


def plot_gpr_samples(gpr_model, n_samples, ax):
    """Plot samples drawn from the Gaussian process model.

    If the Gaussian process model is not trained then the drawn samples are
    drawn from the prior distribution. Otherwise, the samples are drawn from
    the posterior distribution. Be aware that a sample here corresponds to a
    function. """

    x = np.linspace(0, 5, 100)
    X = x.reshape(-1, 1)

    y_mean, y_std = gpr_model.predict(X, return_std=True)
    y_samples = gpr_model.sample_y(X, n_samples)

    for idx, single_prior in enumerate(y_samples.T):
        ax.plot(
            x,
            single_prior,
            linestyle="--",
            alpha=0.7,
            label=f"Sampled function #{idx + 1}",
        )
    ax.plot(x, y_mean, color="black", label="Mean")
    ax.fill_between(
        x,
        y_mean - y_std,
        y_mean + y_std,
        alpha=0.1,
        color="black",
        label=r"$\pm$ 1 std. dev.",
    )
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_ylim([-3, 3])



rng = np.random.RandomState(4)
X_train = rng.uniform(0, 5, 10).reshape(-1, 1)
y_train = np.sin((X_train[:, 0] - 2.5) ** 2)
n_samples = 5


### RBF Kernel

from sklearn.gaussian_process import GaussianProcessRegressor
from sklearn.gaussian_process.kernels import RBF

kernel = 1.0 * RBF(length_scale=1.0, length_scale_bounds=(1e-1, 10.0))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
```

```python

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Radial Basis Function kernel", fontsize=18)
plt.tight_layout()


print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)




### Rational Quadratic Kernel

from sklearn.gaussian_process.kernels import RationalQuadratic

kernel = 1.0 * RationalQuadratic(length_scale=1.0, alpha=0.1, alpha_bounds=(1e-5, 1e15))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Rational Quadratic kernel", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
```

```
#%%
### ExpSineSquared


from sklearn.gaussian_process.kernels import ExpSineSquared


kernel = 1.0 * ExpSineSquared(length_scale=1,periodicity=1,length_scale_bounds=(1e-5, 1e15),periodicity_bounds=(1e-5, 1e15))
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("ExpSineSquared", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%
### Matern


from sklearn.gaussian_process.kernels import Matern


kernel = 1.0 * Matern(length_scale=1,length_scale_bounds=(1e-5, 1e15),nu=1.5)
gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Matern", fontsize=18)
plt.tight_layout()
```

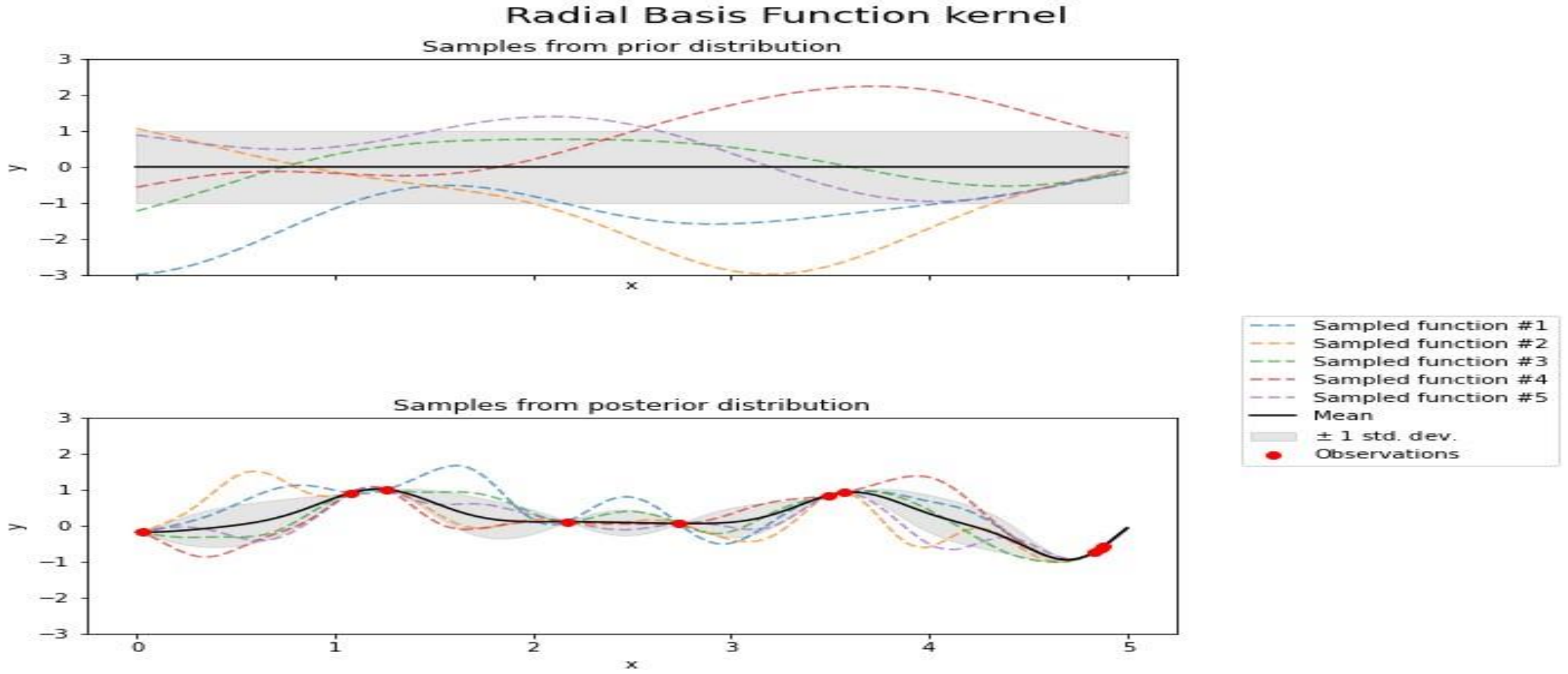
```
166
167 print(f"Kernel parameters before fit:\n{kernel})")
168 print(
169     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
170     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
171 )
172
173 #%%
174 ### WhiteKernel
175
176
177 from sklearn.gaussian_process.kernels import WhiteKernel
178
179
180 kernel = 1.0 * WhiteKernel(noise_level=1,noise_level_bounds=(1e-5, 1e15))
181 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
182
183 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
184
185 # plot prior
186 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
187 axs[0].set_title("Samples from prior distribution")
188
189 # plot posterior
190 gpr.fit(X_train, y_train)
191 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
192 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
193 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
194 axs[1].set_title("Samples from posterior distribution")
195
196 fig.suptitle("WhiteKernel", fontsize=18)
197 plt.tight_layout()
198
199 print(f"Kernel parameters before fit:\n{kernel})")
200 print(
201     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
202     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
203 )
204
205 #%%
206
207 ### Exponentiation
208
209
210 from sklearn.gaussian_process.kernels import Exponentiation
211
212
213 kernel = 1.0 * Exponentiation(RationalQuadratic(), exponent=2)
214 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
215
216 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
217
218 # plot prior
219 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
220 axs[0].set_title("Samples from prior distribution")
221
```

```
221
222 # plot posterior
223 gpr.fit(X_train, y_train)
224 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
225 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
226 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
227 axs[1].set_title("Samples from posterior distribution")
228
229 fig.suptitle("Exponentiation", fontsize=18)
230 plt.tight_layout()
231
232 print(f"Kernel parameters before fit:\n{kernel})")
233 print(
234     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
235     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
236 )
237 #%%
238 ### DotProduct
239
240
241 from sklearn.gaussian_process.kernels import DotProduct
242
243
244 kernel = 1.0 * DotProduct(sigma_0=1,sigma_0_bounds=(1e-5, 1e5))
245 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
246
247 fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))
248
249 # plot prior
250 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
251 axs[0].set_title("Samples from prior distribution")
252
253 # plot posterior
254 gpr.fit(X_train, y_train)
255 plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
256 axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
257 axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
258 axs[1].set_title("Samples from posterior distribution")
259
260 fig.suptitle("DotProduct", fontsize=18)
261 plt.tight_layout()
262
263 print(f"Kernel parameters before fit:\n{kernel})")
264 print(
265     f"Kernel parameters after fit: \n{gpr.kernel_} \n"
266     f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
267 )
268 #%%
269 ### Sum kernel
270
271
272 from sklearn.gaussian_process.kernels import Sum
273
274
275 kernel = 1.0 * Sum(DotProduct(),WhiteKernel())
276 gpr = GaussianProcessRegressor(kernel=kernel, random_state=0)
```

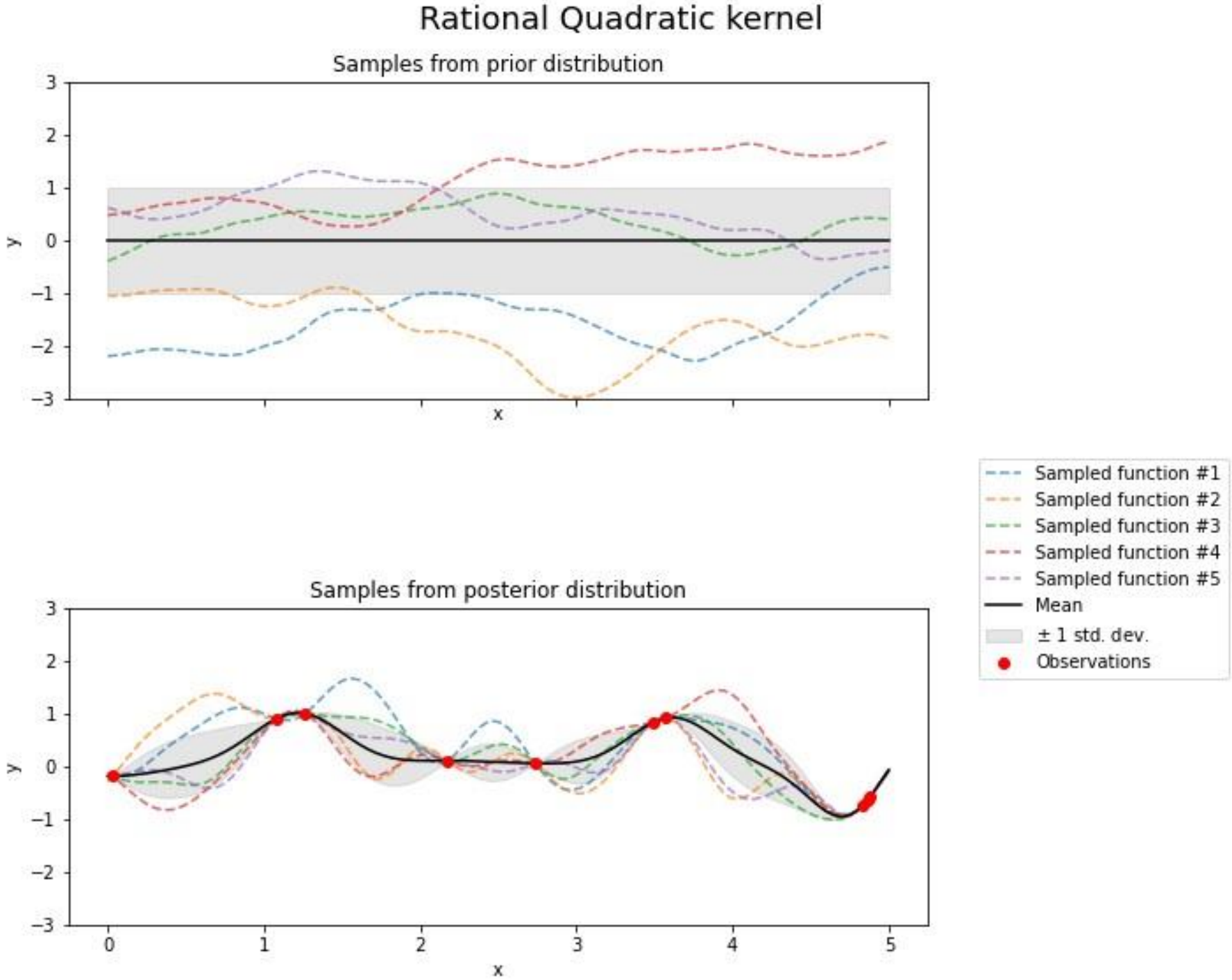
```python

fig, axs = plt.subplots(nrows=2, sharex=True, sharey=True, figsize=(10, 8))

# plot prior
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[0])
axs[0].set_title("Samples from prior distribution")

# plot posterior
gpr.fit(X_train, y_train)
plot_gpr_samples(gpr, n_samples=n_samples, ax=axs[1])
axs[1].scatter(X_train[:, 0], y_train, color="red", zorder=10, label="Observations")
axs[1].legend(bbox_to_anchor=(1.05, 1.5), loc="upper left")
axs[1].set_title("Samples from posterior distribution")

fig.suptitle("Sum kernel", fontsize=18)
plt.tight_layout()

print(f"Kernel parameters before fit:\n{kernel})")
print(
    f"Kernel parameters after fit: \n{gpr.kernel_} \n"
    f"Log-likelihood: {gpr.log_marginal_likelihood(gpr.kernel_.theta):.3f}"
)
#%%

```

OUTPU

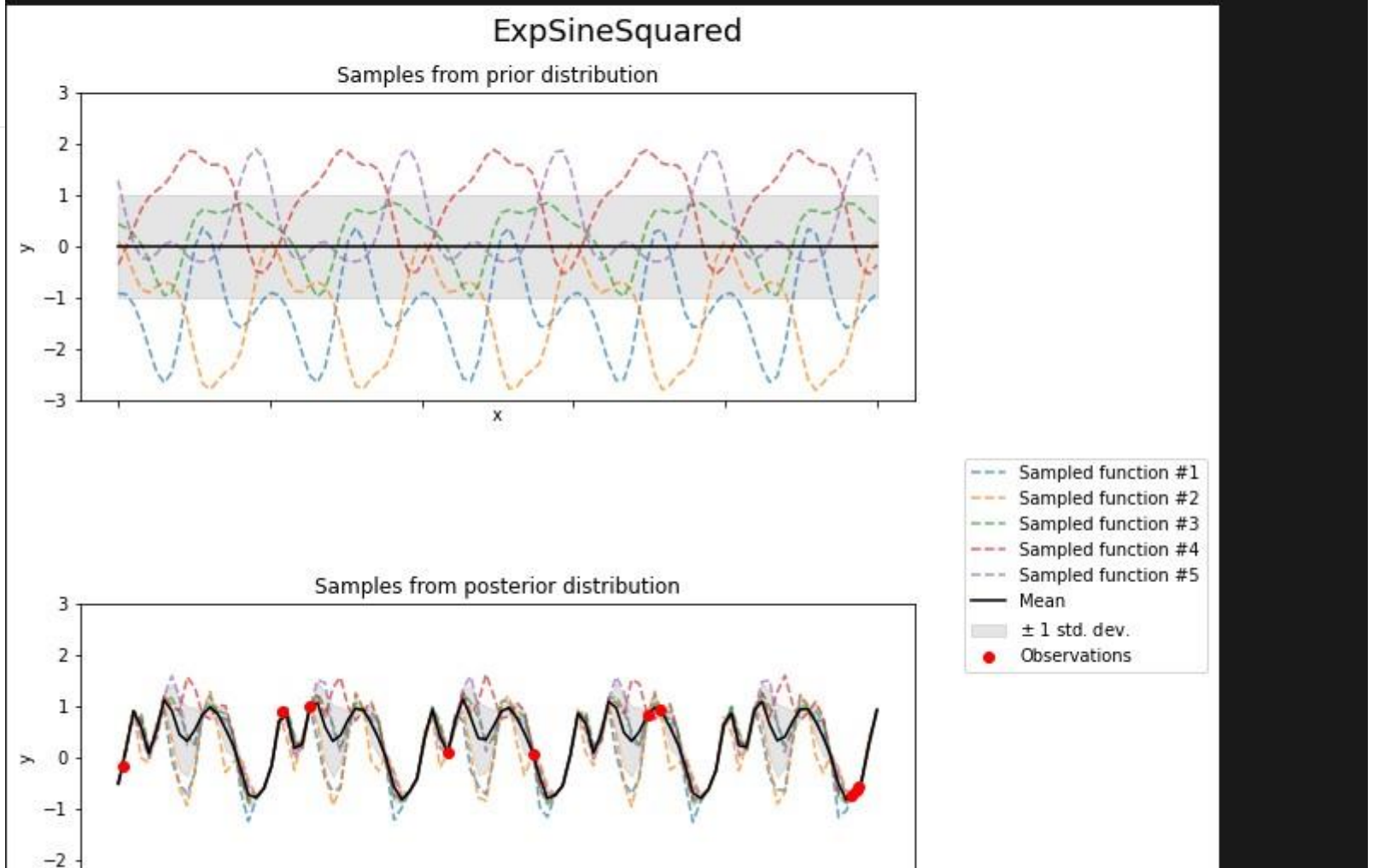
```
In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * RBF(length_scale=1))+++
Kernel parameters after fit:
0.594**2 * RBF(length_scale=0.279)
Log-likelihood: -0.067
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\_gpr.py:506: ConvergenceWarning: lbfgs failed to converge (status=2):
ABNORMAL_TERMINATION_IN_LNSRCH.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
  _check_optimize_result("lbfgs", opt_res)
Kernel parameters before fit:
1**2 * RationalQuadratic(alpha=0.1, length_scale=1))
Kernel parameters after fit:
0.594**2 * RationalQuadratic(alpha=1.78e+06, length_scale=0.279)
Log-likelihood: -0.067
```



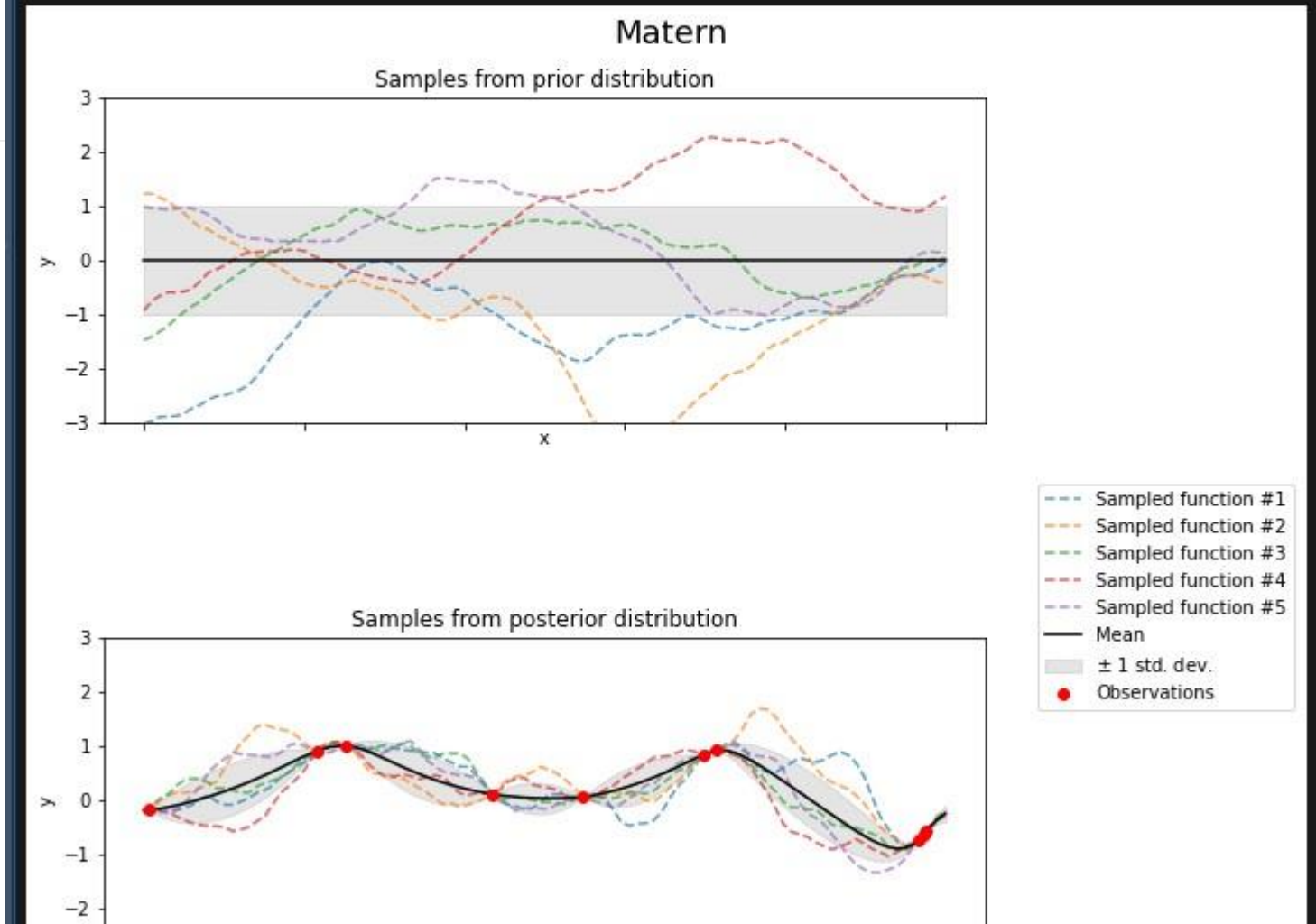
T:


Rational Quadratic kernel
Samples from prior distribution
Samples from posterior distribution
x
y
Sampled function #1
Sampled function #2
Sampled function #3
Sampled function #4
Sampled function #5
Mean
± 1 std. dev.
Observations

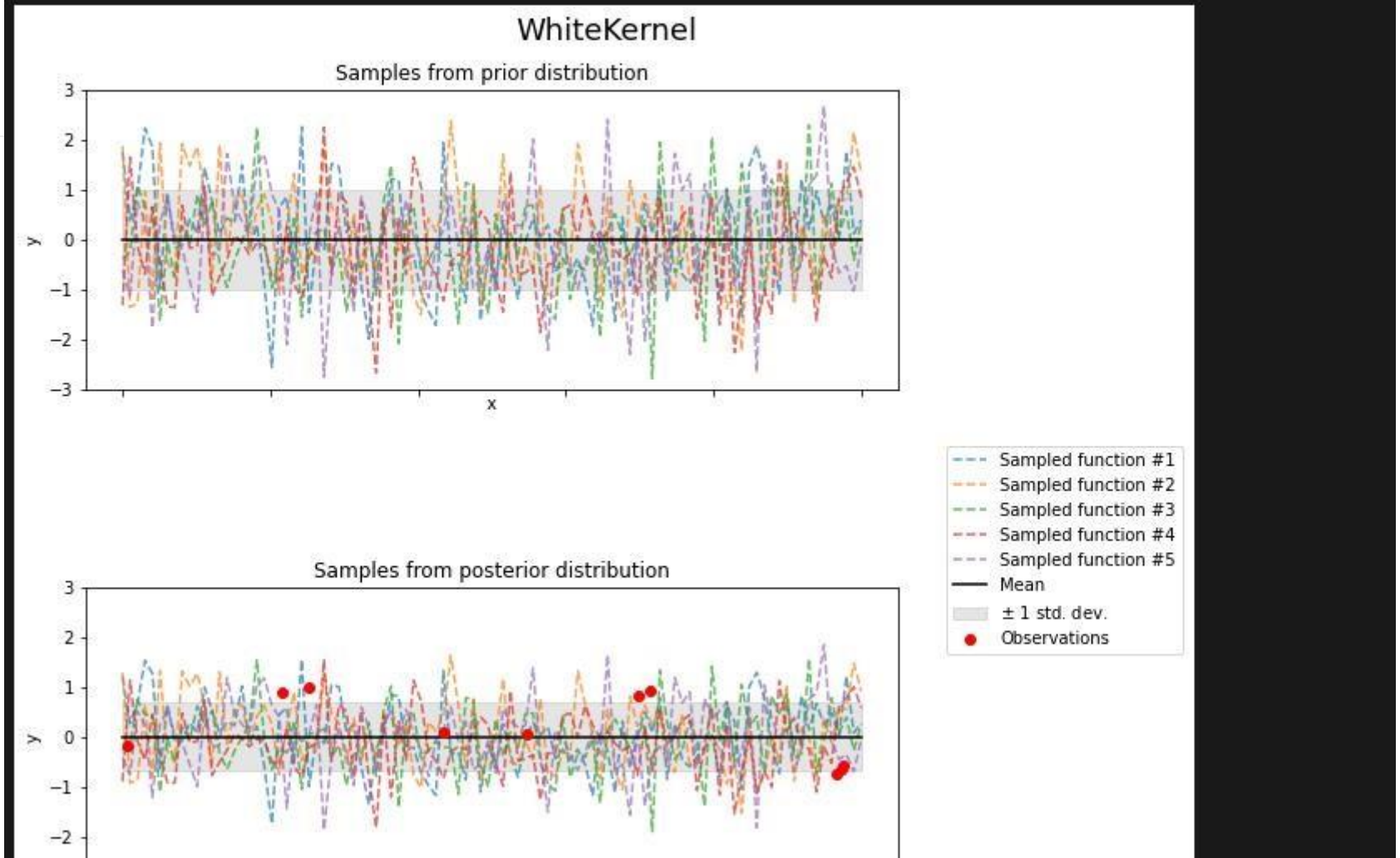
```
In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * ExpSineSquared(length_scale=1, periodicity=1))
Kernel parameters after fit:
0.736**2 * ExpSineSquared(length_scale=0.438, periodicity=0.978)
Log-likelihood: -5.478
```



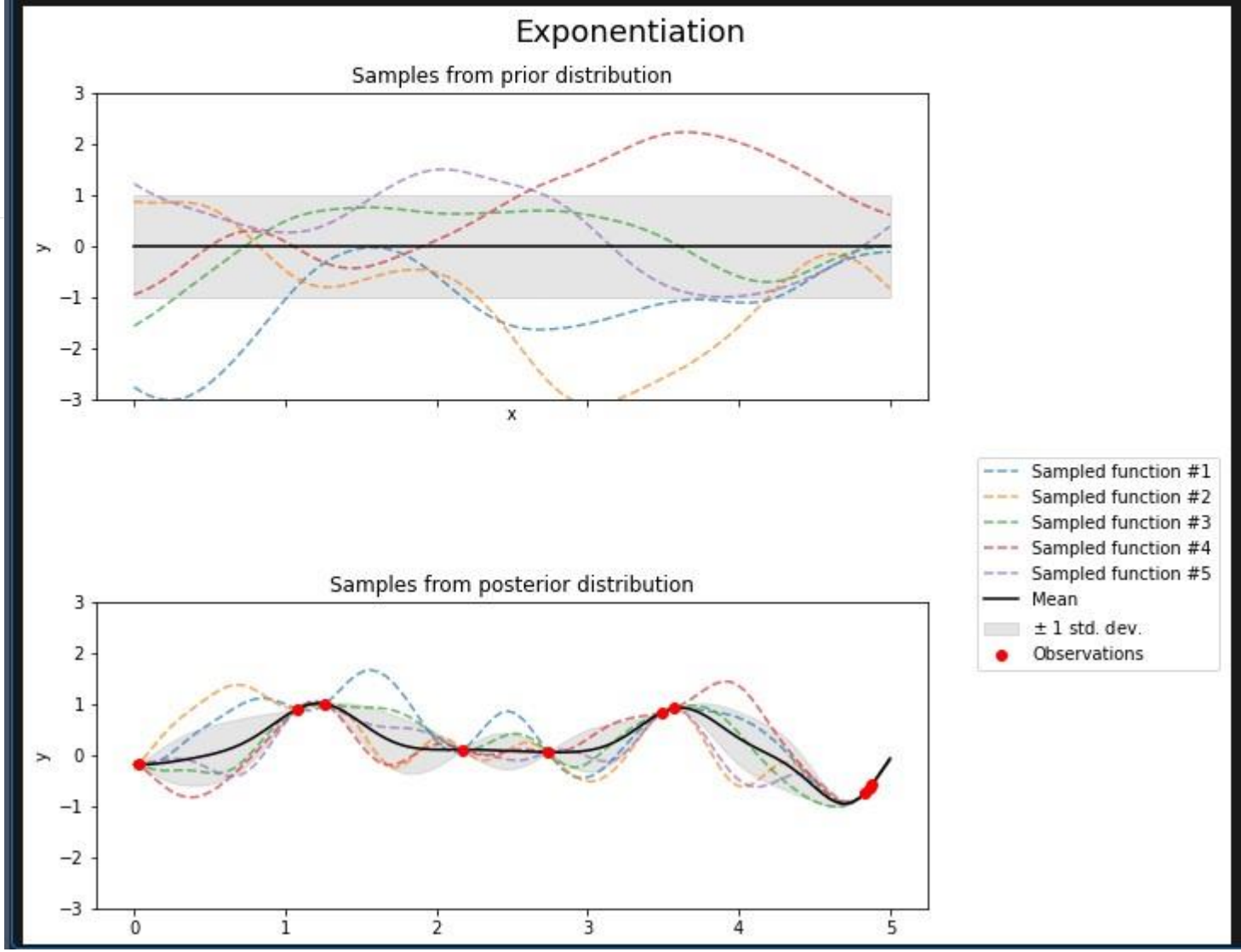
```
In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * Matern(length_scale=1, nu=1.5))
Kernel parameters after fit:
0.609**2 * Matern(length_scale=0.484, nu=1.5)
Log-likelihood: -1.185
```



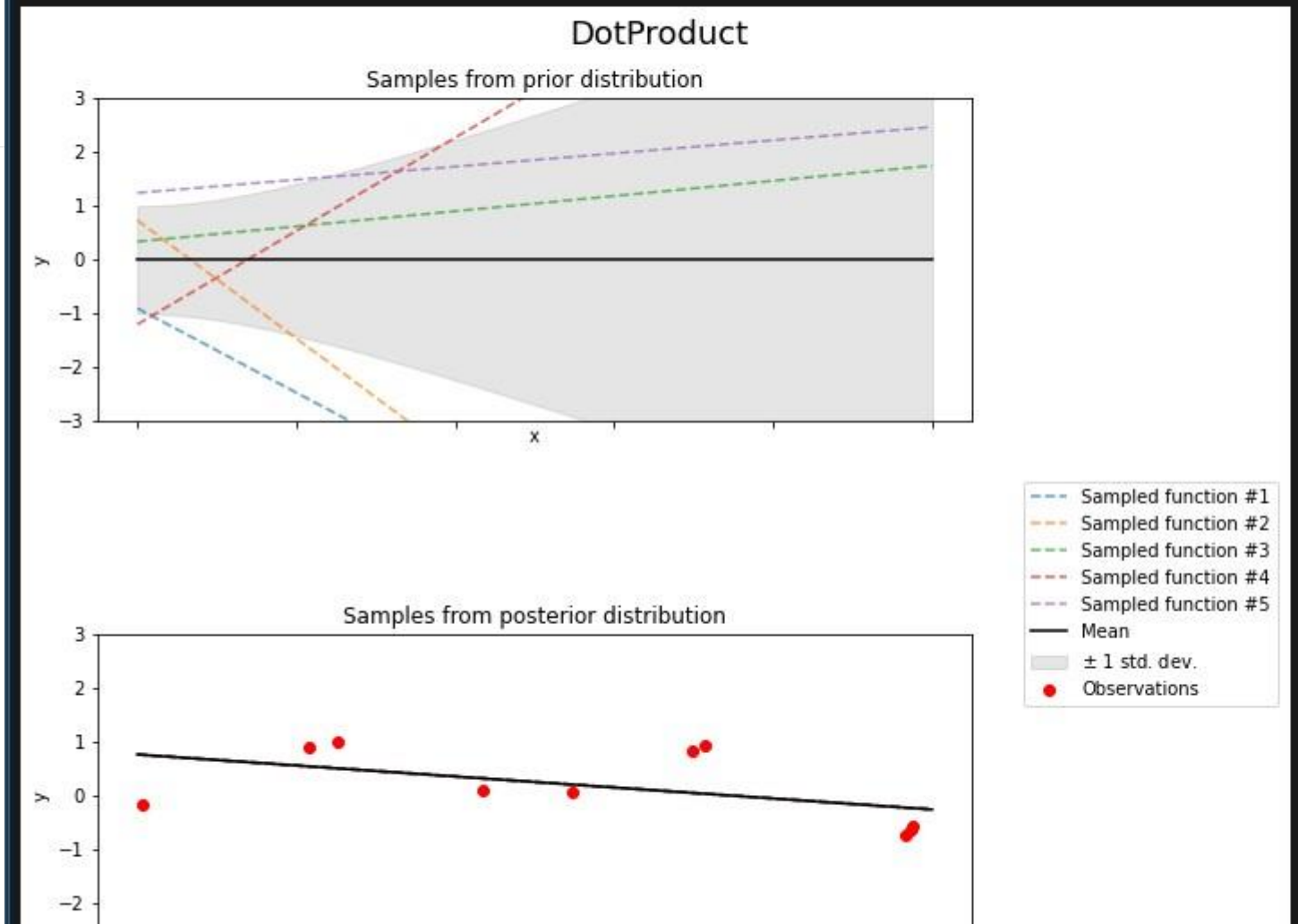
```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * WhiteKernel(noise_level=1))
Kernel parameters after fit:
0.827**2 * WhiteKernel(noise_level=0.684)
Log-likelihood: -10.386
```



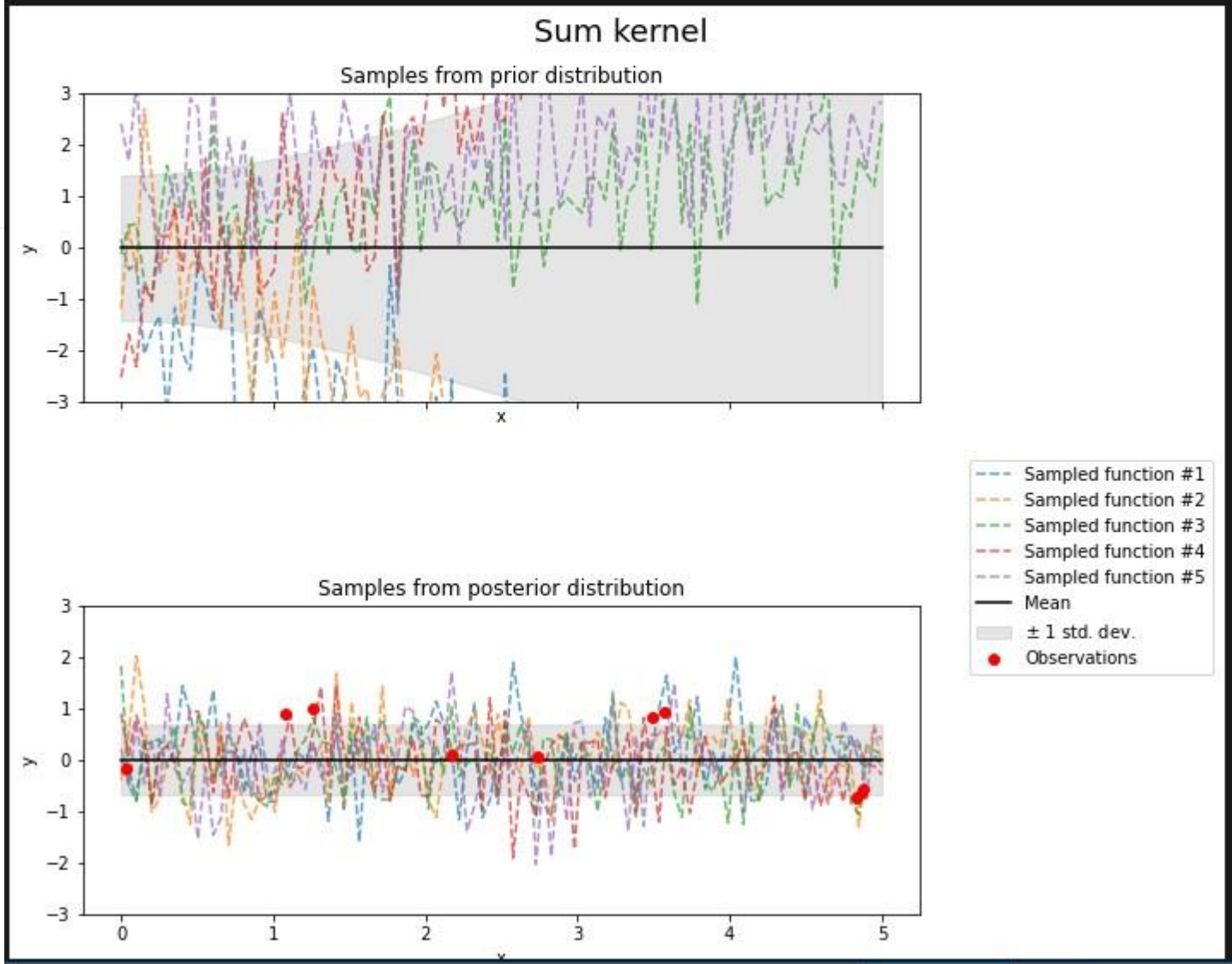
```
In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/exp9.py')
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\kernels.py:418: ConvergenceWarning: The optimal value found for dimension 0 of
parameter k2__kernel__alpha is close to the specified upper bound 100000.0. Increasing the bound and calling fit again may find a better value.
  ConvergenceWarning)
Kernel parameters before fit:
1**2 * RationalQuadratic(alpha=1, length_scale=1) ** 2)
Kernel parameters after fit:
0.594**2 * RationalQuadratic(alpha=1e+05, length_scale=0.394) ** 2
Log-likelihood: -0.067
```



```
In [6]: runcell(5, 'B:/3rd year/5th sem/P&AD/exp9.py')
Kernel parameters before fit:
1**2 * DotProduct(sigma_0=1))
Kernel parameters after fit:
0.999**2 * DotProduct(sigma_0=0.999)
Log-likelihood: -16319044829.997
```



```
In [7]: runcell(6, 'B:/3rd year/5th sem/P&AD/exp9.py')
C:\Users\Dhruv Singhal\anaconda3\lib\site-packages\sklearn\gaussian_process\kernels.py:409: ConvergenceWarning: The optimal value found for dimension 0 of
parameter k1__constant_value is close to the specified lower bound 1e-05. Decreasing the bound and calling fit again may find a better value.
  ConvergenceWarning)
Kernel parameters before fit:
1**2 * DotProduct(sigma_0=1) + WhiteKernel(noise_level=1))
Kernel parameters after fit:
0.00316**2 * DotProduct(sigma_0=0.339) + WhiteKernel(noise_level=4.67e+04)
Log-likelihood: -10.387
```



Experiment 10

CODE:

```python
import numpy as np

x1 = np. random. random( (100, 1))
y= 4 + 3*x1 + np. random. randn(100, 1)

x0 = np. ones ( (100, 1))
X = np. concatenate( (x0, x1), axis = 1)


#%%
temp1 = np.linalg.inv(np.dot (X.T, X))
temp2 = np.dot(temp1,X.T)
w = np. dot(temp2, y)
print("-------------------------------")
print("Least squares method(Direct) Single Input")
print("-------------------------------")
print("W0",w[0])
print("W1",w[1])


#%%
import numpy as np
x1 = np.random. random( (100, 3))
X=np.c_[np.ones((100,1)),x1]
a=[[4,5,8],
   [8,5,7],
   [7,6,3],
   [1,3,8]]
W=np.array(a)
y1=np.dot(X,W)
temp1 = np.linalg.inv(np.dot (X.T, X))
temp2 = np.dot(temp1,X.T)
w = np. dot(temp2, y1)
print("-------------------------------")
print("Least squares method(Direct) Multiple Input")
print("-------------------------------")
print("W1's are:\n" ,w)

```

```
#%%
X_ = np.random. random( (100, 3))
y1= 4 + 3*X_ + np. random. randn(100, 1)
y2= 5 + 2*X_ + np. random. randn(100, 1)
y3= 3 + 6*X_ + np. random. randn(100, 1)
y4= 7 + 9*X_ + np. random. randn(100, 1)
Xwb=np.c_[np.ones((100,1)),X_]
W_=Xwb.T.dot(Xwb)
tp1 = np.linalg.inv(np.dot (Xwb.T, Xwb))
tp2 = np.dot(tp1,Xwb.T)
W1 = np. dot(tp2, y1)
W2 = np. dot(tp2, y2)
W3 = np. dot(tp2, y3)
W4 = np. dot(tp2, y4)


print("------------Modified-------------------")
print("Least squares method(Direct) Multiple Input")
print("------------------------------")
print("W1:\n",W1)
print("W2:\n",W2)
print("W3:\n",W3)
print("W4:\n",W4)
#%%
print(np.concatenate((W1,W2,W3,W4)))
```

OUTPUT:

```
IPython 7.26.0 -- An enhanced Interactive Python.

In [1]: runcell(0, 'B:/3rd year/5th sem/P&AD/exp10.py')

In [2]: runcell(1, 'B:/3rd year/5th sem/P&AD/exp10.py')
------------------------------
Least squares method(Direct) Single Input
------------------------------
W0 [3.6047836]
W1 [3.50268276]

In [3]: runcell(2, 'B:/3rd year/5th sem/P&AD/exp10.py')
------------------------------
Least squares method(Direct) Multiple Input
------------------------------
W1's are:
 [[4. 5. 8.]
 [8. 5. 7.]
 [7. 6. 3.]
 [1. 3. 8.]]
```

```
In [4]: runcell(3, 'B:/3rd year/5th sem/P&AD/exp10.py')
------------Modified-----------------
Least squares method(Direct) Multiple Input
------------------------------
W1:
 [[ 3.87767253  3.87767253  3.87767253]
 [ 3.05152315  0.05152315  0.05152315]
 [ 0.3952086   3.3952086   0.3952086 ]
 [-0.24948184 -0.24948184  2.75051816]]
W2:
 [[ 5.67017869  5.67017869  5.67017869]
 [ 1.25767771 -0.74232229 -0.74232229]
 [-0.28164296  1.71835704 -0.28164296]
 [-0.2510575  -0.2510575   1.7489425 ]]
W3:
 [[ 3.00627306  3.00627306  3.00627306]
 [ 6.48435087  0.48435087  0.48435087]
 [-0.09223401  5.90776599 -0.09223401]
 [-0.3093984  -0.3093984   5.6906016 ]]
W4:
 [[ 6.98119803  6.98119803  6.98119803]
 [ 9.46229368  0.46229368  0.46229368]
 [-0.05479141  8.94520859 -0.05479141]
 [-0.61903883 -0.61903883  8.38096117]]

In [5]: runcell(4, 'B:/3rd year/5th sem/P&AD/exp10.py')
[[ 3.87767253  3.87767253  3.87767253]
 [ 3.05152315  0.05152315  0.05152315]
 [ 0.3952086   3.3952086   0.3952086 ]
 [-0.24948184 -0.24948184  2.75051816]
 [ 5.67017869  5.67017869  5.67017869]
 [ 1.25767771 -0.74232229 -0.74232229]
 [-0.28164296  1.71835704 -0.28164296]
 [-0.2510575  -0.2510575   1.7489425 ]
 [ 3.00627306  3.00627306  3.00627306]
 [ 6.48435087  0.48435087  0.48435087]
 [-0.09223401  5.90776599 -0.09223401]
 [-0.3093984  -0.3093984   5.6906016 ]
 [ 6.98119803  6.98119803  6.98119803]
 [ 9.46229368  0.46229368  0.46229368]
 [-0.05479141  8.94520859 -0.05479141]
 [-0.61903883 -0.61903883  8.38096117]]
```

Experiment 11

Different Anomaly Detection Algorithms on 2D Dataset

Importing required libraries

```
I [3] import tim
      impor numpy a n
      impor matplotli
      impor matplotlib.pyplo a pl
      fro sklearn impor sv
      fro sklearn.datasets impor make_moo make_blob
      fro sklearn.covariance impor EllipticEnvelo
      fro sklearn.ensemble impor IsolationFore
      fro sklearn.neighbors impor LocalOutlierFact
```

Data

for this experiment we have use make_moons, make_blobs datsets which are already in sklearn.datasets

sklearn.datasets.make_blobs(n_samples=100, n_features=2, *, centers=None, cluster_std=1.0, center_box=(- 10.0, 10.0), shuffle=True, random_state=None, return_centers=False) Generate isotropic Gaussian blobs for clustering.

sklearn.datasets.make_moons(n_samples=100, *, shuffle=True, noise=None, random_state=None) Make two interleaving half circles. A simple toy dataset to visualize clustering and classification algorithms.

Preprocessing & Visualization

for this experiment we do not need preprocessing as we need to visualize different anomaly detection algorithms

Model Building

Covariance Matrix

This metric assesses how many standard deviations σ away x_i is from μ, thereby being dimensionless. An extreme observation has a large distance from the center of a distribution. It is useful in predictive maintenance as it enables finding unusual behavior of a system; upcoming down-times and malfunction can be targeted and identified, which is particularly important since finding faults in equipment, machines, and devices early can reduce the extent of the damage.

One Class SVM

One Class SVM mode is trained in only one class, referred to as the normal class. The model learns all the features and patterns of the normal class . When a new observation is introduced to the model, then based on its learning, the One Class SVM detects if the new observation deviates from the normal behaviour then it classifies it as an oulier else the new observation will identified as an inlier. One Class SVM is based on Support Vector Machines where the binary classes are separated by a non-linear hyper plane

Local Outlier Factor

Local outlier factor (LOF) is an algorithm used for Unsupervised outlier detection. It produces an anomaly score that represents data points which are outliers in the data set. It does this

by measuring the local density deviation of a given data point with respect to the data points near it.

Isolation Forest

Isolation Forest is based on the Decision Tree algorithm. It isolates the outliers by randomly selecting a feature from the given set of features and then randomly selecting a split value between the max and min values of that feature. This random partitioning of features will produce shorter paths in trees for the anomalous data points, thus distinguishing them from the rest of the data.

Compile & Train

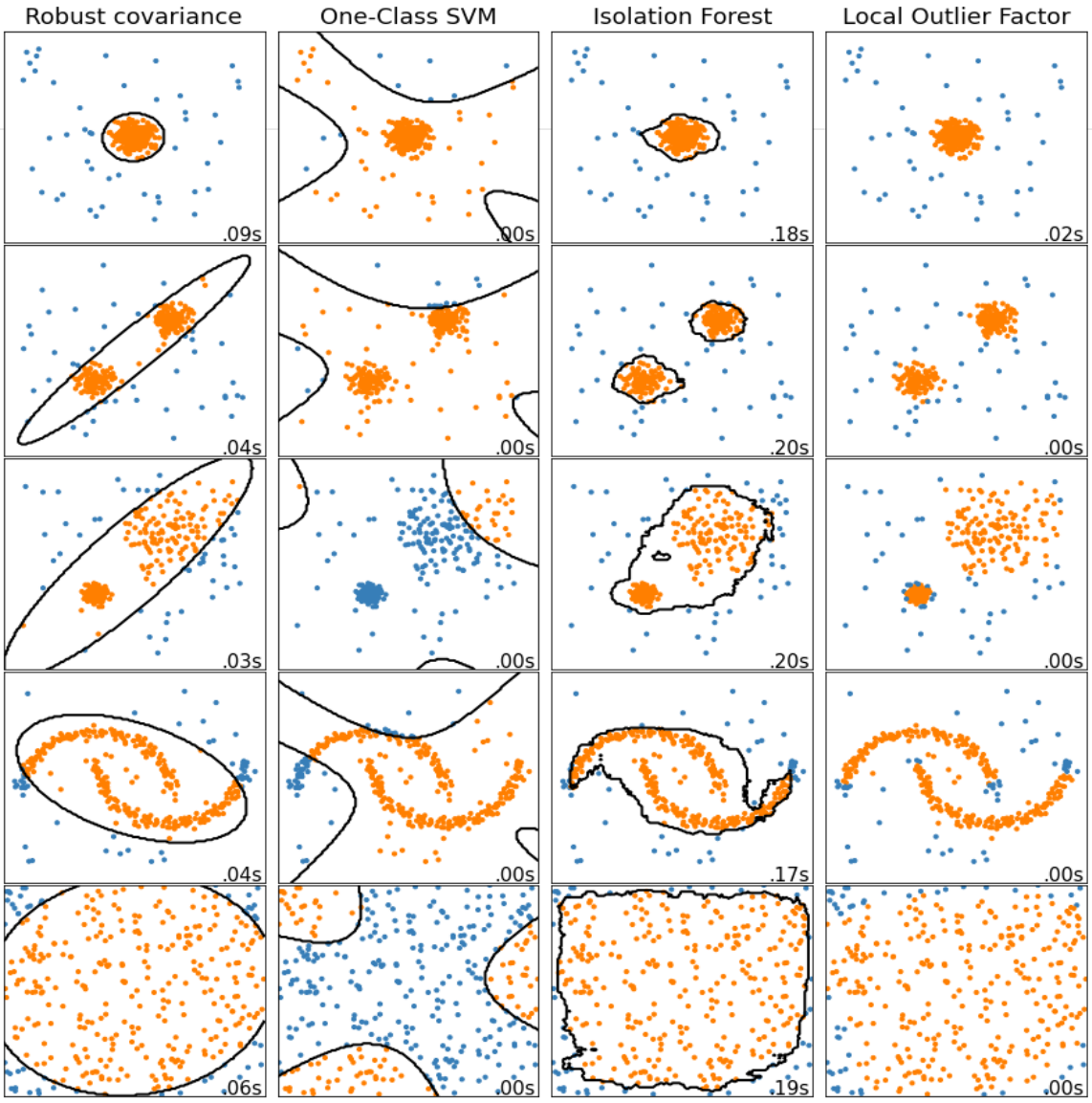
Altering the hyperparameters to see the effect of the hyperparameter on the working of the algorithm

In [4]:
```
matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
n_samples = 300
outliers_fraction = 0.15
n_outliers = int(outliers_fraction * n_samples)
n_inliers = n_samples - n_outliers
anomaly_algorithms = [
    ("Robust covariance", EllipticEnvelope(contamination=outliers_fraction)),
    ("One-Class SVM", svm.OneClassSVM(nu=outliers_fraction, kernel="poly",
                                      gamma=0.1)),
    ("Isolation Forest", IsolationForest(contamination=outliers_fraction,
                                         random_state=42)),
    ("Local Outlier Factor", LocalOutlierFactor(
        n_neighbors=35, contamination=outliers_fraction))]

# Define datasets
blobs_params = dict(random_state=0, n_samples=n_inliers, n_features=2)
datasets = [
    make_blobs(centers=[[0, 0], [0, 0]], cluster_std=0.5,
               **blobs_params)[0],
    make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[0.5, 0.5],
               **blobs_params)[0],
    make_blobs(centers=[[2, 2], [-2, -2]], cluster_std=[1.5, .3],
               **blobs_params)[0],
    4. * (make_moons(n_samples=n_samples, noise=.05, random_state=0)[0] -
          np.array([0.5, 0.25])),
    14. * (np.random.RandomState(42).rand(n_samples, 2) - 0.5)]
matplotlib.rcParams['contour.negative_linestyle'] = 'solid'
# Compare given classifiers under given settings
xx, yy = np.meshgrid(np.linspace(-7, 7, 150),
                     np.linspace(-7, 7, 150))

plt.figure(figsize=(len(anomaly_algorithms) * 2 + 4, 12.5))
plt.subplots_adjust(left=.02, right=.98, bottom=.001, top=.96, wspace=.05,
                    hspace=.01)

plot_num = 1
rng = np.random.RandomState(42)

for i_dataset, X in enumerate(datasets):
    # Add outliers
    X = np.concatenate([X, rng.uniform(low=-6, high=6, size=(n_outliers, 2))],
                       axis=0)

    for name, algorithm in anomaly_algorithms:
        t0 = time.time()
        algorithm.fit(X)
        t1 = time.time()
        plt.subplot(len(datasets), len(anomaly_algorithms), plot_num)
        if i_dataset == 0:
            plt.title(name, size=18)

        # fit the data and tag outliers
        if name == "Local Outlier Factor":
            y_pred = algorithm.fit_predict(X)
        else:
            y_pred = algorithm.fit(X).predict(X)

        # plot the levels lines and the points
        if name != "Local Outlier Factor":  # LOF does not implement predict
            Z = algorithm.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)
            plt.contour(xx, yy, Z, levels=[0], linewidths=2, colors='black')

        colors = np.array(['#377eb8', '#ff7f00'])
        plt.scatter(X[:, 0], X[:, 1], s=10, color=colors[(y_pred + 1) // 2])

        plt.xlim(-7, 7)
        plt.ylim(-7, 7)
        plt.xticks(())
        plt.yticks(())
        plt.text(.99, .01, ('%.2fs' % (t1 - t0)).lstrip('0'),
                 transform=plt.gca().transAxes, size=15,
                 horizontalalignment='right')
        plot_num += 1
```

Result

By this experiment i came to know the working of different anomaly detection algorithms and how they differ from each other in working and in time perspective how they find anomaly

In [ ]:



Experiment 12

Support Vector Machines

Importing required libraries

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.datasets import make_classification
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
```

Creating Data

Creating Dataset with 5000 sample points and 2 features and 2 classes and number of informative features are 2 and number of redundant features are 0 and number of repeated features are 0 and random_state is 50 and difference between 2 class is set as 3.5

In [2]:
```
X, y= make_classification(n_samples=5000,n_features=2,n_classes=2
,n_informative=2,n_redundant=0,n_repeated=0,random_state=0,class_se
p=2.5)
```

Preprocessing & Visualization

since we have created dataset using make_classfiocation , it by default make dataset's mean tending to 0 and Standard Deviation to 1 which doesn't require any preprocessing.

Model Building

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

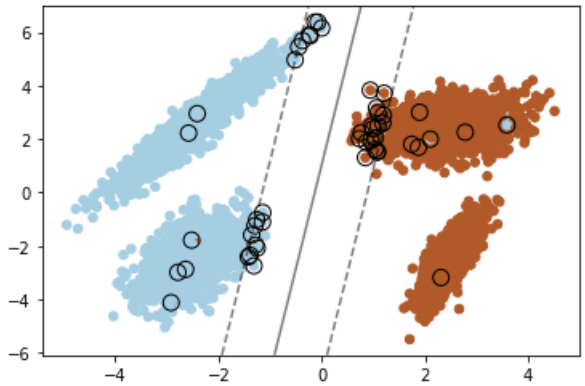
SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine. In this case we have called the in built function svm.SVC

Compile & Train

Firstly splitting the input dataset into training and test parts following which training set is passed through svm.SVC and is fitted to it. and predicting over the test test to check the model performance.

In [3]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20,random_state=42)
clf = svm.SVC(kernel="linear", C=10)
clf.fit(X_train,Y_train)
Y_pred=clf.predict(X_test)
print("train Accuracy:",clf.score(X_train,Y_train))
print("test Accuracy:",clf.score(X_test,Y_test))
plt.scatter(X[:, 0], X[:, 1], c=y, s=30, cmap=plt.cm.Paired)


# plot the decision function

ax = plt.gca()
xlim = ax.get_xlim()
ylim = ax.get_ylim()

# create grid to evaluate model
xx = np.linspace(xlim[0], xlim[1], 30)
yy = np.linspace(ylim[0], ylim[1], 30)
YY, XX = np.meshgrid(yy, xx)
xy = np.vstack([XX.ravel(), YY.ravel()]).T
Z = clf.decision_function(xy).reshape(XX.shape)

# plot decision boundary and margins
ax.contour(
    XX, YY, Z, colors="k", levels=[-1, 0, 1], alpha=0.5, linestyles=["--", "-", "--"]
)
# plot support vectors
ax.scatter(
    clf.support_vectors_[:, 0],
    clf.support_vectors_[:, 1],
    s=100,
    linewidth=1,
    facecolors="none",
    edgecolors="k",
)

plt.show()
```

```
train Accuracy: 0.99675
test Accuracy: 0.996
```



Result

since training is giving 99.67% accuracy and in testing it is giving 99.60% accuracy so to check if model is overfitting or not or to improve the validation accuracy more and make more generalized model . So to overcome this we are using k fold cross validation and hyperparameter tuning to improve the overall score and generate generalizesd model and knowing which hyperparameter combination gives best overall score

In [4]:
```
#Cross Validation without Hyper parameter Tuning
clf= svm.SVC(kernel='linear',C=2)
kf=KFold(n_splits=70)
score=cross_val_score(clf, X, y, cv=kf)
print("Average Cross Validation score ( Accuracy) :{}".format(score.mean()))
```

```
Average Cross Validation score ( Accuracy) :0.9965906550413592
```

In [5]:
```
# Hyper parameter tuning on SVM
tuned_parameters=[{'kernel':["linear"], 'C':[100,10,20,40,50]},
                  {'kernel':["rbf"], 'C':[90,40,50,100,90]},

                  ]
clf=GridSearchCV(svm.SVC(),tuned_parameters,scoring=('accuracy'))
clf.fit(X,y)
print("Best parameters set found on development set:")
print()
print(clf.best_params_)
print()
print("Best Score:",clf.best_score_)
```

```
Best parameters set found on development set:

{'C': 100, 'kernel': 'linear'}
Best Score: 0.9966000000000002
```

In [6]:
```
#%% kfold cross validation with hyperparameter tuning
k = 5
kf = KFold(n_splits=k, random_state=None)
model= GridSearchCV(svm.SVC(),tuned_parameters,scoring=('accuracy'))

for train_index,test_index in kf.split(X):
    X_train, X_test = X[train_index],X[test_index]
    y_train, y_test = y[train_index], y[test_index]

    model.fit(X_train,y_train)
    pred_values=model.predict(X_test)
print("Best parameters set found on development set:")
print()
print(model.best_params_)
print()
print("Best Score:",model.best_score_)
```

```
Best parameters set found on development set:

{'C': 100, 'kernel': 'linear'}

Best Score: 0.9964999999999999
```

Now the overall score
is 99.649% accuracy

In [ ]:

Experiment 13

Random Forest Classfier

Importing required libraries

In [1]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
```

Data creation

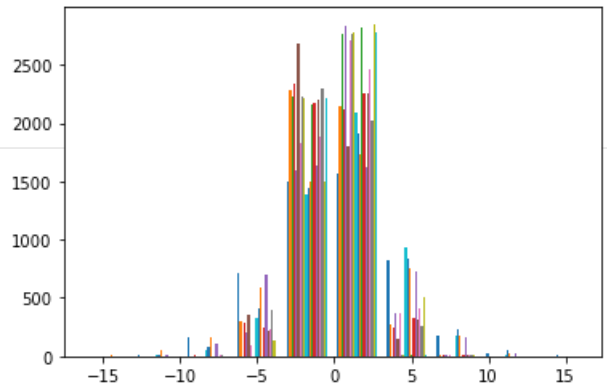
Creating Dataset using make_classfication with 5000 sample points and 5 classes with 20 number of features in which 10 are informative features and 5 are redundant features and shuffling is turned ON

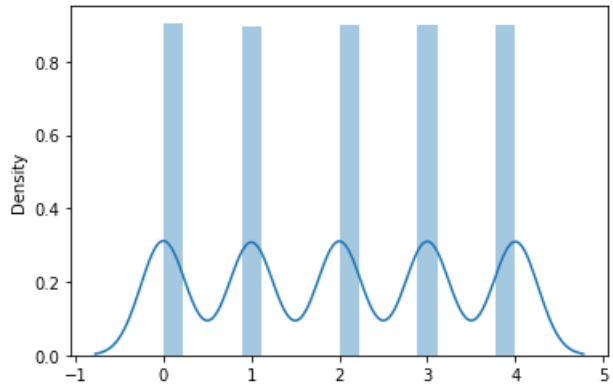
In [2]:
```
X,y = make_classification(n_samples=5000,n_classes=5,n_features=20,
n_informative=10,n_redundant=5,shuffle=True,random_state=0)
X=pd.DataFrame(X)
y=pd.DataFrame(y)
```

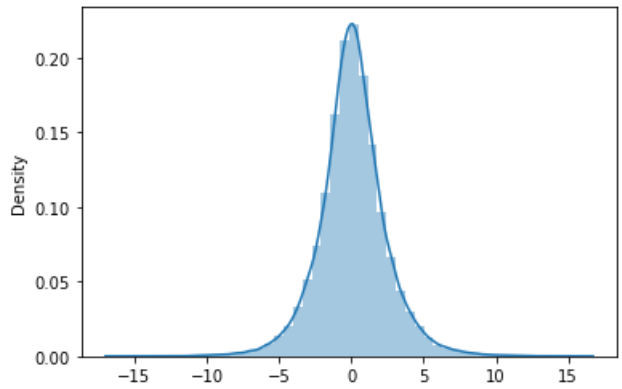
Preprocessing & Visualization

since we have created dataset using make_classfiocation , it by default make dataset's mean tending to 0 and Standard Deviation to 1 which doesn't require any preprocessing. For Visualization varition of values of X(with 20 features) and y's distplot to observe y's vartion of values and distplot of X to check if any pre processing is to be required

In [3]:
```
plt.hist(X)
plt.show()
sns.distplot(y)
plt.show()
sns.distplot(X)
plt.show()
```







feature extraction preprocessing, correlation matrix to check if any pre processing is to be required

In [4]:
```
_,graph=plt.subplots(figsize=(15,10))
sns.heatmap(X.corr(),annot=True,ax=graph,square=True)
plt.show()
```

Model Building

Random forest classifier creates a set of decision trees from a randomly selected subset of the training set. It is basically a set of decision trees from a randomly selected subset of the training set and then It collects the votes from different decision trees to decide the final prediction. Assuming dataset has “m” features, the random forest will randomly choose “k” features where k < m. Now, the algorithm will calculate the root node among the k features by picking a node that has the highest information gain. After that, the algorithm splits the node into child nodes and repeats this process “n” times. Now we have a forest with n trees. Finally, perform bootstrapping, ie, combine the results of all the decision trees present in the forest.

We have directly used the in built function of sklearn library i.e., from sklearn.ensemble import RandomForestClassifier which directly imports this classifier

Compile & Train

Firstly splitting the input dataset into training and test parts following which training set is passed through RandomForestClassifier and is fitted to it. and predicting over the test test to check the model performance.

In [5]:
```
X_train,X_test,Y_train,Y_test=train_test_split(X,y,test_size=0.20,random_state=42)

model=RandomForestClassifier()
model.fit(X_train,Y_train)
Y_pred=model.predict(X_test)
print("train Accuracy:",model.score(X_train,Y_train))
print("test Accuracy:",model.score(X_test,Y_test))
```

```
tr
ai
n
Ac
cu
ra
cy
:
1.
0
te
st
Ac
cu
ra
cy
:
0.
79
```

Result

since training is giving 100% accuracy and in testing it is giving 79% accuracy so it's clearly overfitting so to overcome this we are using k fold cross validation and hyperparameter tuning to improve the overall score and generate generalizesd model and knowing which hyperparameter combination gives best overall score

In [6]:
```
#Cross Validation without Hyper
parameter tuning from
sklearn.model_selection import KFold
from sklearn.model_selection import
cross_val_score clf =
RandomForestClassifier(max_depth=10,
random_state=0) kf=KFold(n_splits=7)
score=cross_val_score(clf, X, y,
scoring='accuracy',cv=kf) print("Cross
Validation Scores are {}".format(score))
```

```
print("Average Cross Validation score
:{}".format(score.mean()))

Cross Validation Scores are [0.78461538 0.77062937 0.80672269 0.82633053 0.78431373
0.78011204
 0.79411765]
Average Cross Validation score :0.7924059134143168
```

In [7]:
```
from sklearn.model_selection import GridSearchCV tuned_parameters =
[{'n_estimators':[10,20,40,100],'criterion':['gini', 'entropy'],
'max_features':['auto', 'sqrt', 'log2'],'bootstrap':[True,False]}]
clf=GridSearchCV(RandomForestClassifier(),tuned_parameters,scoring=('
accuracy'),verbose=3) clf.fit(X,y)
print("Best parameters set found on
development set:") print()
print(clf.best
_params_)
print()
print("Best
Score:",clf.be
st_score_)
```

```
=   0.3s
[CV 1/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=40;,
score=0.804 total time
=   0.7s
[CV 2/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=40;,
score=0.805 total time
=   0.7s
[CV 3/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=40;,
score=0.813 total time
=   0.7s
[CV 4/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=40;,
score=0.794 total time
=   0.8s
[CV 5/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=40;,
score=0.804 total time
=   0.8s
[CV 1/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=100;,
score=0.810 total tim e=   2.0s
[CV 2/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=100;,
score=0.827 total tim e=   1.9s
[CV 3/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=100;,
score=0.837 total tim e=   2.0s
[CV 4/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=100;,
score=0.805 total tim e=   2.0s
[CV 5/5] END bootstrap=False, criterion=gini, max_features=auto, n_estimators=100;,
score=0.821 total tim
     2
```

Best parameters set found on development set:

{'bootstrap': False, 'criterion': 'gini', 'max_features': 'log2', 'n_estimators': 100}

Best Score: 0.8214 Now the overall score is 82.14% accuracy which is above the testing accuracy without cross validation

Experiment 14

DBSCAN Implementation

Importing Required Libraries

In [1]:
```
from sklearn.cluster import DBSCAN
import pandas as pd
import matplotlib.pyplot as plt
import warnings
from sklearn.preprocessing import StandardScaler
warnings.filterwarnings('ignore')
```

Data

for this experiment we have use make_blobs datsets which are already in sklearn.datasets

sklearn.datasets.make_blobs(n_samples=100, n_features=2, *, centers=None, cluster_std=1.0, center_box=(- 10.0, 10.0), shuffle=True, random_state=None, return_centers=False) Generate isotropic Gaussian blobs for clustering.

In [2]:
```
from sklearn.datasets import make_blobs
```

In [3]:
```
X,y=make_blobs(n_samples=1000, n_features=2, centers=[[0.5, 2], [-1, -1], [1.5, -1]],
               cluster_std=0.5, center_box=(- 10.0, 10.0), shuffle=True, random_state=42)
```

In [4]:
```
X.shape
```

Out[4]: (1000, 2)

In [5]:
```

```

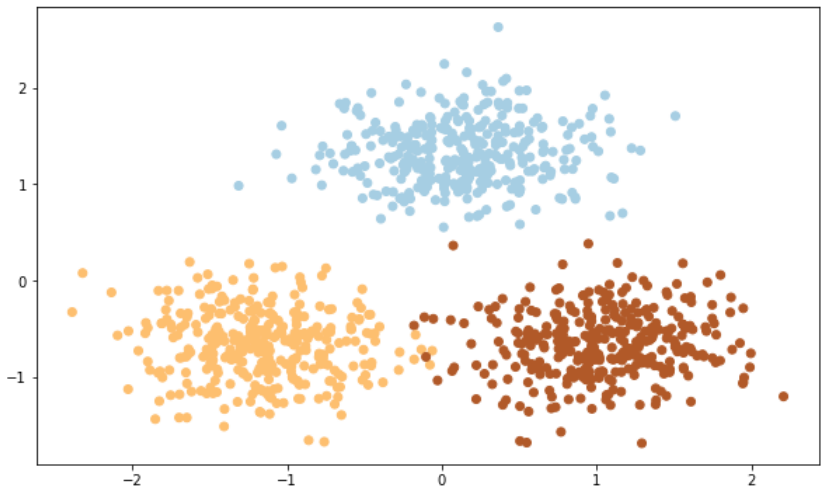
```
y.shape
```

Out[5]: (1000,)

Data Preprocessing and Visualization

In [6]:
```
X = StandardScaler().fit_transform(X)
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1], c=y, cmap='Paired')
```

Out[6]: <matplotlib.collections.PathCollection at 0x1bc149f3388>



Model Building

In [7]:
```
db = DBSCAN(eps=0.45, min_samples=50)
db.fit(X)
y_pred = db.fit_predict(X)
plt.figure(figsize=(10,6))
plt.scatter(X[:,0], X[:,1],c=y_pred, cmap='Paired')
```

Out[7]: <matplotlib.collections.PathCollection at 0x1bc16dcd608>