Introduction to Python with Breast Cancer Data Set

Python notebook using data from Breast Cancer Wisconsin (Diagnostic) Data Set ·

2,107 views · 10mo ago

Version 1

1 commit

Notebook

Data

Log

Comments

In this tutorial, we aim to perform basic data manipulation with the Breast Cancer Wisconsin (Diagnostic) Data Set provided by UCI.

Firstly, we need to import our data and necessary libraries for our study.

In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt #for data visualizing processes


# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list the files in the input directory

import os
print(os.listdir("../input"))

data = pd.read_csv("../input/data.csv") # you can add data with using pandas library easily

# Any results you write to the current directory are saved as output.
```

```
['data.csv']
```

When you import our data, we want to check information about data. It is possible to do this with a single command as follows:

In [2]:

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 569 entries, 0 to 568
```

```
Data columns (total 33 columns):
id                         569 non-null int
64
diagnosis                  569 non-null obj
ect
radius_mean                569 non-null flo
at64
texture_mean               569 non-null flo
at64
perimeter_mean             569 non-null flo
at64
area_mean                  569 non-null flo
at64
smoothness_mean            569 non-null flo
at64
compactness_mean           569 non-null flo
at64
concavity_mean             569 non-null flo
at64
concave points_mean        569 non-null flo
at64
symmetry_mean              569 non-null flo
at64
fractal_dimension_mean     569 non-null flo
at64
radius_se                  569 non-null flo
at64
texture_se                 569 non-null flo
at64
perimeter_se               569 non-null flo
at64
area_se                    569 non-null flo
at64
smoothness_se              569 non-null flo
at64
compactness_se             569 non-null flo
at64
concavity_se               569 non-null flo
at64
concave points_se          569 non-null flo
at64
symmetry_se                569 non-null flo
at64
fractal_dimension_se       569 non-null flo
at64
radius_worst               569 non-null flo
at64
texture_worst              569 non-null flo
at64
perimeter_worst            569 non-null flo
```

```
at64
area_worst                569 non-null flo
at64
smoothness_worst          569 non-null flo
at64
compactness_worst         569 non-null flo
at64
concavity_worst           569 non-null flo
at64
concave points_worst      569 non-null flo
at64
symmetry_worst            569 non-null flo
at64
fractal_dimension_worst   569 non-null flo
at64
Unnamed: 32               0 non-null float
64
dtypes: float64(31), int64(1), object(1)
memory usage: 146.8+ KB
```

There are a few other commands that can be used to get detailed information about the data. For example; you can reach first 5 rows in data with this command:

In [3]:

```
data.head(5)
```

Out[3]:

	id	diagnosis	radius_mean	texture_mean	perimete
0	842302	M	17.99	10.38	122.80
1	842517	M	20.57	17.77	132.90
2	84300903	M	19.69	21.25	130.00
3	84348301	M	11.42	20.38	77.58
4	84358402	M	20.29	14.34	135.10

Furthermore, you can see all columns with using this command:

In [4]:

```
data.columns
```

Out[4]:

```
Index(['id', 'diagnosis', 'radius_mean', 't
exture_mean', 'perimeter_mean',
       'area_mean', 'smoothness_mean', 'com
pactness_mean', 'concavity_mean',
       'concave points_mean', 'symmetry_mea
n', 'fractal_dimension_mean',
       'radius_se', 'texture_se', 'perimete
r_se', 'area_se', 'smoothness_se',
       'compactness_se', 'concavity_se', 'c
oncave points_se', 'symmetry_se',
       'fractal_dimension_se', 'radius_wors
t', 'texture_worst',
       'perimeter_worst', 'area_worst', 'sm
oothness_worst',
       'compactness_worst', 'concavity_wors
t', 'concave points_worst',
       'symmetry_worst', 'fractal_dimension
_worst', 'Unnamed: 32'],
      dtype='object')
```

Also, we may want to find out about the dependency of our predictions by seeing the correlation between the columns of our data.

The correlation between 2 columns is determined as;

- If there is a correlation value which is close to 1, it is directly proportional,
- If there is a correlation value which is close to -1, it is inversely proportional,
- If there is a correlation value which is equal to 0, there is no relationship between two features.

For example; we can say that, the relationship between raduis_mean and perimeter_mean is directly proportional.

In [5]:

```
data.corr()
```

Out[5]:

	id	radius_mean	texture_mean
id	1.000000	0.074626	0.099770
radius_mean	0.074626	1.000000	0.323782
texture_mean	0.099770	0.323782	1.000000

perimeter_mean	0.073159	0.997855	0.329533
area_mean	0.096893	0.987357	0.321086
smoothness_mean	-0.012968	0.170581	-0.023389
compactness_mean	0.000096	0.506124	0.236702
concavity_mean	0.050080	0.676764	0.302418
concave points_mean	0.044158	0.822529	0.293464
symmetry_mean	-0.022114	0.147741	0.071401
fractal_dimension_mean	-0.052511	-0.311631	-0.076437
radius_se	0.143048	0.679090	0.275869
texture_se	-0.007526	-0.097317	0.386358
perimeter_se	0.137331	0.674172	0.281673
area_se	0.177742	0.735864	0.259845
smoothness_se	0.096781	-0.222600	0.006614
compactness_se	0.033961	0.206000	0.191975
concavity_se	0.055239	0.194204	0.143293
concave points_se	0.078768	0.376169	0.163851
symmetry_se	-0.017306	-0.104321	0.009127
fractal_dimension_se	0.025725	-0.042641	0.054458
radius_worst	0.082405	0.969539	0.352573
texture_worst	0.064720	0.297008	0.912045
perimeter_worst	0.079986	0.965137	0.358040
area_worst	0.107187	0.941082	0.343546
smoothness_worst	0.010338	0.119616	0.077503
compactness_worst	-0.002968	0.413463	0.277830
concavity_worst	0.023203	0.526911	0.301025
concave points_worst	0.035174	0.744214	0.295316
symmetry_worst	-0.044224	0.163953	0.105008
fractal_dimension_worst	-0.029866	0.007066	0.119205
Unnamed: 32	NaN	NaN	NaN

After we have reached our data information, we may also want to access information visually on the whole data.

We can easily visualize (plot) our data with **Matplotlib** library in Python.

The **Matplotlib** library contains various graphic styles such as Line, Histogram, Scatter etc. For this reason, we can choose graphic styles that will be suitable for efficiency.

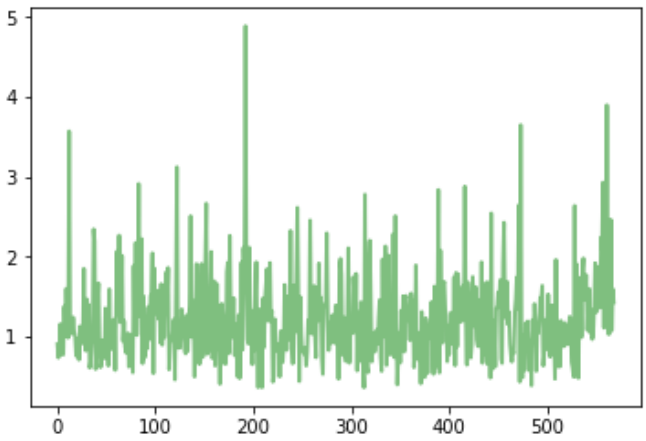
Let's try some plot drawings.

In [6]:

```
data.texture_se.plot(kind='line', color='green', label=
'texture_se', linewidth=2, alpha=0.5, grid=False, linest
yle='-')
```

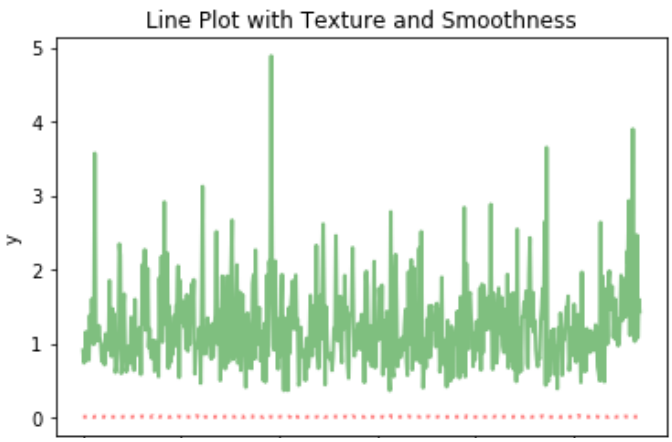
Out[6]:

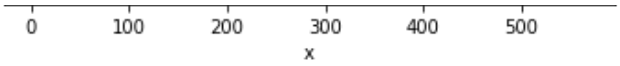
```
<matplotlib.axes._subplots.AxesSubplot at 0
x7fb66549b2e8>
```



In [7]:

```
data.texture_se.plot(kind='line', color='green', label=
'texture_se', linewidth=2, alpha=0.5, grid=False, linest
yle='-')
data.smoothness_se.plot(kind='line', color='red', label=
'smoothness_se', linewidth=2, alpha=0.5, grid=False, lin
estyle=':')
plt.xlabel('x')              # label = name of label
plt.ylabel('y')
plt.title('Line Plot with Texture and Smoothness')
# title = title of plot
plt.show()
```



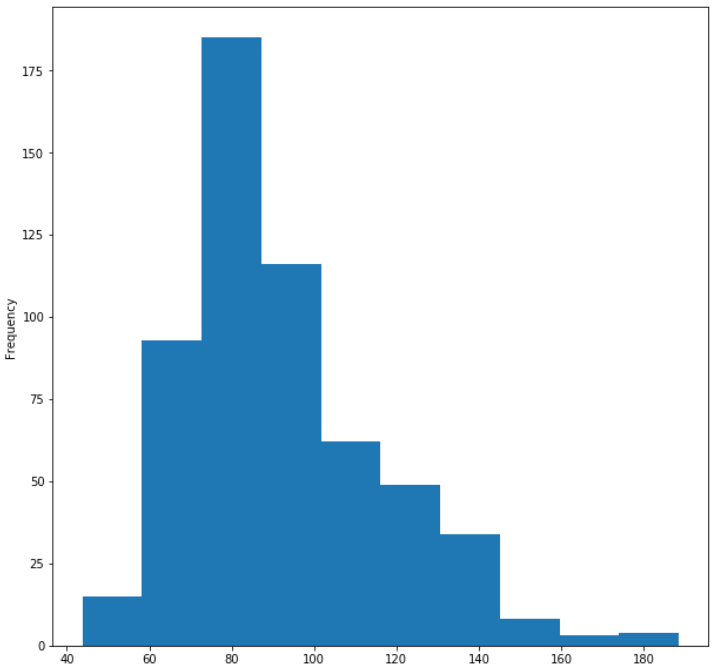


In [8]:

```
data.perimeter_mean.plot(kind='hist', bins=10, figsize=(
10,10))
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0
x7fb6653eee10>
```

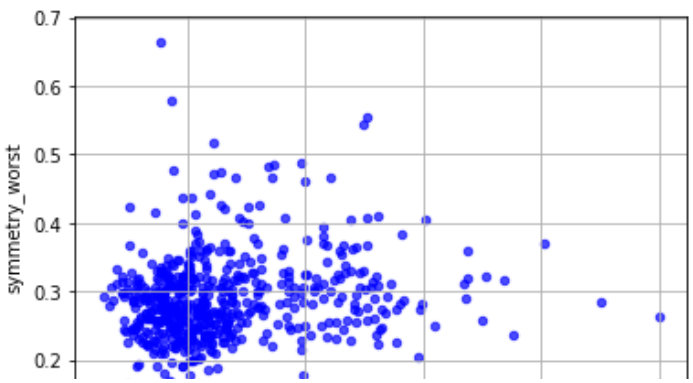


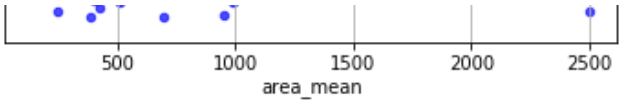
In [9]:

```
data.plot(kind='Scatter', x='area_mean', y='symmetry_wor
st', alpha=0.7, color='blue', grid=True)
```

Out[9]:

```
<matplotlib.axes._subplots.AxesSubplot at 0
x7fb6653fe7b8>
```



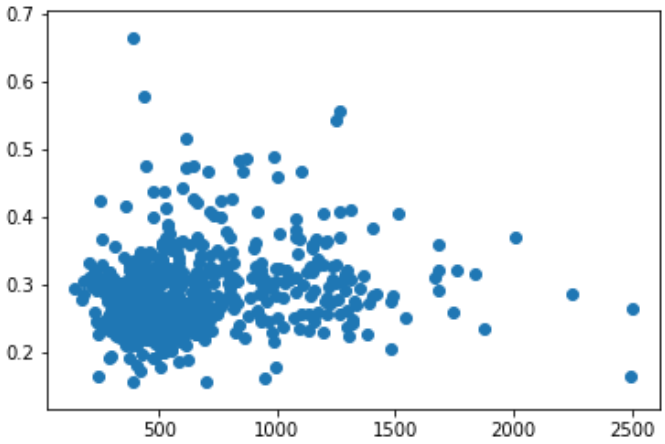


In [10]:

```
plt.scatter(data.area_mean, data.symmetry_worst)
```

Out[10]:

```
<matplotlib.collections.PathCollection at 0
x7fb6642fdf28>
```



As you can see above, we can make the same plot drawing with different commands and variables.

There is some variables for plot customizations.

- kind determines the plot type.
- alpha determines the opacity of the points.
- color determines the color of the points.
- label determines the flag of the points.
- grid determines creates squares on plot.
- figsize determines plot size.

After learning how to visualize the data, now we learn how to reach the desired data specifically.

Dictionary data type contains key and value pairs for every entry. We use the Dictionary because it works faster than the list structure.

Let's look at Dictionary syntax and processes.

In [11]:

```
dictionary = {'type': 'malignant', 'area': 'breast'}
print(dictionary)
print("\n")
print(dictionary.keys()) #prints keys
print(dictionary.values()) #prints values

dictionary['type'] = 'benign' #update dictionary
print("\n")
print(dictionary.keys())
print(dictionary.values())

del dictionary['area'] #deletes area's key & value
print("\n")
print(dictionary.keys())
print(dictionary.values())
```

```
{'type': 'malignant', 'area': 'breast'}


dict_keys(['type', 'area'])
dict_values(['malignant', 'breast'])


dict_keys(['type', 'area'])
dict_values(['benign', 'breast'])


dict_keys(['type'])
dict_values(['benign'])
```

There are some operators that we use to perform mathematical comparison and filtering operations on our data.

Now, we will mention about these operators.

In [12]:

```
malignant = data['diagnosis'] =='M'
data[malignant]
```

Out[12]:

	id	diagnosis	radius_mean	texture_mean	perim
0	842302	M	17.99	10.38	122.8
1	842517	M	20.57	17.77	132.9
2	84300903	M	19.69	21.25	130.0

3	84348301	M	11.42	20.38	77.58
4	84358402	M	20.29	14.34	135.1
5	843786	M	12.45	15.70	82.57
6	844359	M	18.25	19.98	119.6
7	84458202	M	13.71	20.83	90.20
8	844981	M	13.00	21.82	87.50
9	84501001	M	12.46	24.04	83.97
10	845636	M	16.02	23.24	102.7
11	84610002	M	15.78	17.89	103.6
12	846226	M	19.17	24.80	132.4
13	846381	M	15.85	23.95	103.7
14	84667401	M	13.73	22.61	93.60
15	84799002	M	14.54	27.54	96.73
16	848406	M	14.68	20.13	94.74
17	84862001	M	16.13	20.68	108.1
18	849014	M	19.81	22.15	130.0
22	8511133	M	15.34	14.26	102.5
23	851509	M	21.16	23.04	137.2
24	852552	M	16.65	21.38	110.0
25	852631	M	17.14	16.40	116.0
26	852763	M	14.58	21.53	97.41
27	852781	M	18.61	20.25	122.1
28	852973	M	15.30	25.27	102.4
29	853201	M	17.57	15.05	115.0
30	853401	M	18.63	25.11	124.8
31	853612	M	11.84	18.70	77.93
32	85382601	M	17.02	23.98	112.8
...	...	...	...	...	...
444	9110127	M	18.03	16.85	117.5
446	9110732	M	17.75	28.03	117.3
449	911157302	M	21.10	20.52	138.1
451	9111805	M	19.59	25.00	127.7
460	911296201	M	17.08	27.15	111.2
461	911296202	M	27.42	26.27	186.9
468	9113538	M	17.60	23.33	119.0
479	911916	M	16.25	19.51	109.8
487	913505	M	19.44	18.82	128.1
489	913535	M	16.69	20.20	107.1
492	914062	M	18.01	20.56	118.4

498	914769	M	18.49	17.52	121.3
499	91485	M	20.59	21.24	137.8
501	91504	M	13.82	24.49	92.33
503	915143	M	23.09	19.83	152.1

Did you find this Kernel useful?
Show your appreciation with an upvote

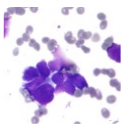
4

Data

Data Sources

Breast Cancer ...

32 columns



Breast Cancer Wisconsin (Diagnostic) Data Set

Predict whether the cancer is benign or malignant

Last Updated: 3 years ago (Version 2)

About this Dataset

Features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image. n the 3-dimensional space is that described in: [K. P. Bennett and O. L. Mangasarian: "Robust Linear Programming Discrimination of Two Linearly Inseparable Sets", Optimization Methods an Software 1, 1992, 23-34].

This database is also available through the UW CS ft server: ftp ftp.cs.wisc.edu cd math-prog/cpo-dataset/machine-learn/WDBC/

Also can be found on UCI Machine Learning Repository: https://archive.ics.uci.edu/ml/datasets/Breast+Canc r+Wisconsin+%28Diagnostic%29

Attribute Information:

1) ID number 2) Diagnosis (M = malignant, B = benign) 3-32)

Ten real-valued features are computed for each cell nucleus:

a) radius (mean of distances from center to points on the perimeter) b) texture (standard deviation of gray-

scale values) c) perimeter d) area e) smoothness (loc
variation in radius lengths) f) compactness
(perimeter^2 / area - 1.0) g) concavity (severity of

Run Info

Succeeded	True	Run Time	12.3 seconds
Exit Code	0	Queue Time	0 seconds
Docker Image Name	kaggle/python (Dockerfile)	Output Size	0
Timeout Exceeded	False	Used All Space	False
Failure Message			

Log

Download Lo

```
Time  Line #  Log Message
5.4s       1  [NbConvertApp] Converting notebook script.ipynb to
              html
5.4s       2  [NbConvertApp] Executing notebook with kernel:
              python3
11.4s      3  [NbConvertApp] Support files will be in
              __results___files/
              [NbConvertApp] Making directory __results___files
              [NbConvertApp] Making directory __results___files
11.5s      4  [NbConvertApp] Making directory __results___files
              [NbConvertApp] Making directory __results___files
              [NbConvertApp] Making directory __results___files
              [NbConvertApp] Writing 369799 bytes to
              __results__.html
11.5s      5
11.5s      7  Complete. Exited with code 0.
```

Comments (0)

Click here to comment...