MDM_AI-ML_Lect5_Pandas_Basics

January 9, 2025

Data Analytics with Python

Lecture 5 (MDM)

By Ajit Kumar (ICT Mumbai)

Jan. 09, 2025

1 Introduction to Pandas

Pandas is a powerful, open-source data analysis and manipulation library for Python. It is widely used in data science, machine learning, and scientific computing due to its ability to handle structured data effectively.

Some key features and functionalities of pandas: * Data Structures: Series, DataFrame, xarray * Data Handling:loading, cleaning, processing * Indexing and Selection: accessing through indices, labels or boolean conditions, advanced indexing using .loc, .iloc, .at etc * Data Analysis: statistical and mathematical manipulations, aggregation, summation, tramsformation * Time Series: Tools for working with time-indexed data, resampling, and frequency conversion * Visualization: Integrated with Matplotlib for basic plotting.

[ ]:
```
# pip install pandas
```

[ ]:
```
import pandas as pd
from pandas import Series, DataFrame
```

[ ]:
```
obj = Series([4, 7, -5, 3,-12.5, 9.6,13])
```

[ ]:
```
obj.
```

[ ]:
```
obj
```

[ ]:
```
obj.index
```

[ ]:
```
obj.values
```

[ ]:
```
obj2 = Series([14, 17, -50, 31,92,13,19],index=['a','b','c','d','e','f','g'])
```

[ ]:
```

```

[ ]:
```
obj2['d']
```

[ ]:
```
obj2['d']=21
obj2
```

[ ]:
```
obj2<20
```

[ ]:
```
obj2[obj2<20]
```

[ ]:
```
('g' in obj2,'h' in obj2 )
```

[ ]:
```
obj2.keys()
```

[ ]:
```
list(obj2.items())
```

[ ]:
```
obj2*2
```

[ ]:
```
import numpy as np
np.sin(obj2)
```

1.0.1 Question:

Generate 200 random numbers between 1 and 100 and find the sum of all the numbers between 40 and 60 both inclusive.

[ ]:
```
import numpy as np
dt = Series(np.random.randint(100,size=200))
dt
```

[ ]:
```
sum(dt[(dt>=40) & (dt<=60)])
```

[ ]:
```
len(dt[(dt>=40) & (dt<=60)])
```

[ ]:
```
import numpy as np
np.exp(obj)
```

[ ]:
```
##
def f(x):
    return x**2-np.sin(x)+1/(1+x**x)

f(Series(range(1,50,4)))
```

Dictionary in Pandas

[ ]:
```
dict = {1:'Jan',2:'Feb','name':'Ajit'}
dict.keys()
dict.items()
```

1.1 Series as dictionary

[ ]:
```
### Indian states population in million
sdata={'UP':210,'Bihar':116,'WB':95,'MP':76,'TN':79,\
       'MAH':119}
```

[ ]:
```
sdata.keys()
```

[ ]:
```
sdata.values()
```

[ ]:
```
obj3 = Series(sdata)
obj3
```

[ ]:
```
states=['UP','MAH','WB','PB','TN','Bihar','MP','Raj','CG']
Series(sdata,index=states)
```

[ ]:
```
states=['UP','MAH','CG','WB','TN','Bihar','Raj','MP','GUJ']
obj4=Series(sdata,index=states)
obj4
```

[ ]:
```
pd.isnull(obj4)
```

[ ]:
```
sum(pd.isnull(obj4))
```

[ ]:
```

```

[ ]:
```
pd.notnull(obj4)
```

[ ]:
```
obj4.isnull()
```

[ ]:
```
obj4
```

[ ]:
```
obj4['TN':'MP']
```

[ ]:
```
obj4['MP']
```

1.2 The Pandas DataFrame Object

[ ]:
```
sdata={'states':['UP','MAH','WB','TN','Bihar','MP','GUJ'],\
'population':[210.,  119.,   95.,   79.,  116.,   79,61],\
'Density':[828,365,1029,555,1102,236,308]
}
st_data = pd.DataFrame(sdata)
st_data
```

[ ]:
```
st_data.shape
```

```
[ ]: st_data.head(2)
     st_data.tail(2)
```

```
[ ]: st_data.info()
```

```
[ ]: st_data.describe()
```

```
[ ]: st_data['Area'] = st_data['population']/st_data['Density']
```

```
[ ]: st_data
```

```
[ ]: st_data.T
```

```
[ ]: st_data.to_csv('states_pop.csv')
```

```
[ ]: ST = pd.read_csv('states_pop.csv')
```

```
[ ]:
```

```
[ ]: st_data.to
```

```
[ ]:
```

```
[ ]:
```

```
[ ]:
```

```
[ ]: st_data.index
```

```
[ ]: population ={'UP':207281477.,'MAH':112372972, 'WB':91347736,'TN':72138958,\
                 'Bihar':103804637,'MP':72597565,'GUJ':60383628, 'KER':np.nan}
     density={'UP':828,'MAH':365,'WB':1029,'TN':555,\
              'Bihar':1102,'MP':np.nan,'GUJ':308,'KER':576}
     population =pd.Series(population)
     density = pd.Series(density)
     states = pd.DataFrame({'population': population,
                            'density': density})
     states
```

```
[ ]: states['population']['TN']
```

```
[ ]: states.density # same as states['density']
```

```
[ ]: states.columns
```

$$Density = \frac{Population}{Area}$$

[ ]:
```
states['area'] = states['population'] / states['density']
states
```

[ ]:
```
states.notna()
```

[ ]:
```
states.isnull()
```

[ ]:
```
states.T
```

[ ]:
```
states.to_csv('states_pop.csv')
```

[ ]:
```
ST = pd.read_csv('states_pop.csv')
```

[ ]:
```
ST.info()
```

[ ]:
```
ST.describe()
```

1.3 Indexers: .loc, .iloc, .at and .ix

- The .loc method is label-based and is used to access rows and columns using labels or Boolean
- The .iloc method is position-based and is used to access rows and columns using integer indices.
- The .at method is optimized for fast access of a single scalar value by label.
- .ix is the most general indexer and will support any of the inputs in .loc and .iloc.
- .ix also supports floating point label schemes.
- .ix is exceptionally useful when dealing with mixed positional and label based hierarchical indexes.

[ ]:
```
data1 = pd.Series(['a', 'b', 'c','d','e'])
data1[2:4]
```

[ ]:
```
ST[2:5]
```

[ ]:
```
data = pd.Series(['a', 'b', 'c','d','e'], index=[1,2, 3, 5,8])
data
```

[ ]:
```
# implicit index when slicing
data[4:6]
```

[ ]:
```
data.loc[4:6]
```

[ ]:
```
data.iloc[2]
```

Because of this potential confusion in the case of integer indexes, Pandas provides some special indexer attributes that explicitly expose certain indexing schemes. These are not functional methods, but attributes that expose a particular slicing interface to the data in the Series.

First, the loc attribute allows indexing and slicing that always references the explicit index:

[ ]:
```
data.loc[4:6]
```

[ ]:
```
## Axis indexes with duplicate values
obj = Series(range(10),index=['a', 'b', 'c', 'a', 'b','d','b','f','b','f'])
obj
```

[ ]:
```
obj.index.is_unique
```

[ ]:
```
ojj=obj.drop_duplicates()
```

[ ]:
```
ojj
```

[ ]:
```

```

[ ]:
```
st_data
```

[ ]:
```
st_data.iloc[3][2]
```

[ ]:
```
st_data.columns
```

[ ]:
```
states.loc[states.density > 500, ['population', 'density','area']]
```

[ ]:
```
states.iloc[0, 1] = 103804657
states
```

1.4 Concactenating

[ ]:
```

```

[ ]:
```
### Concatenating Along an Axis
arr = np.arange(12).reshape((3, 4))
arr
```

[ ]:
```
np.concatenate([arr, arr])
```

[ ]:
```
np.concatenate([arr, arr],axis=1) # Columnwise
```

1.5 Dealing with Duplicates

[ ]:
```
data = DataFrame({'k1': ['one'] * 3 + ['two'] * 4,\
                  'k2': [1, 1, 2, 3, 3, 4, 4]})
data
```

[ ]:
```
data.drop_duplicates()
```

[ ]:
```
data['k3'] = range(7)
data
```

[ ]:
```
data.drop_duplicates()
```

[ ]:
```
data.drop_duplicates(['k2'])
```

1.6 Missing Data in Pandas

Pandas treats None and NaN as essentially interchangeable for indicating missing or null values. To facilitate this convention, there are several useful methods for detecting, removing, and replacing null values in Pandas data structures. They are:

isnull(): Generate a boolean mask indicating missing values

notnull(): Opposite of isnull()

dropna(): Return a filtered version of the data

fillna(): Return a copy of the data with missing values filled or imputed

[ ]:
```
data = pd.Series([1, np.nan, 'hello', 2.5, 'Ajit',None])
data.isnull()
```

[ ]:
```
data[data.isnull()]
```

[ ]:
```
data[data.notnull()]
```

[ ]:
```
from numpy import nan as NA

data = DataFrame([[1., 6.5, 3.,8.0],
                  [1., NA,3, NA],[NA, NA, 7,2], [NA, 6.5, 3.,NA]])
data
```

[ ]:
```
cleaned=data.dropna();cleaned
```

[ ]:
```
data.dropna(how='any')
```

[ ]:
```
data.dropna(axis='columns')
```

[ ]:
```
data.fillna(0)# Replace NA by 0
```

[ ]:
```
data.fillna({1: 0.5}) ## Replacing in 2nd column
```

[ ]:
```
data.fillna({3: 10}) ## Replacing in 4th column
```

1.7 IPL Data

[ ]:
```
IPL= { 'year': [2008 , 2009 ,2010 , 2011 , 2012 ,2013 , \
2014 , 2015,2016,2017,2018],
'team': ['MI', 'CSK','DD', 'KKR','RR', 'MI','RCB', 'KIXP','MI','KKR','RCB'],
'wins':[8 , 7 , 3 , 6 , 9 , 5 , 6 , 8,10,7,3 ] ,
'draws':[1 , 0 , 2 , 1 , 1 , 2 , 2 , 1 ,0,2,0] ,
```

```
'losses': [5 , 7 , 9 , 7 , 4 , 7 , 6 , 5 ,4,4,11]
}
```

[ ]:
```
IPL = pd.DataFrame (IPL , columns = ['year' , 'team' , 'wins' , \
'draws' , 'losses'])
IPL
```

[ ]:
```
IPL.shape
```

[ ]:
```
IPL['wins']
```

[ ]:
```
IPL.describe()
```

[ ]:
```
IPL[IPL['wins']>=8]
```

[ ]:
```
wins_sorted = IPL.sort_values(by='losses',ascending=False)
wins_sorted
```

[ ]:
```
IPL[IPL['team']=='MI']
```

[ ]:
```
newdata=IPL[['wins','draws','losses']]
newdata.cov()
```

[ ]:
```
newdata.corr()
```

[ ]:
```
states.drop(labels='Bihar')
```

[ ]:
```
states
```

1.8 Grouping

[ ]:
```
df= pd.DataFrame({'A' : ['foo', 'bar', 'foo', 'bar','bar','foo','foo', 'bar',␣
↪'foo', 'foo'],
  'B' : ['one', 'one', 'two', 'three','one','two', 'two', 'one',␣
↪'three','two'],
  'C' : np.random.randn(10),
   'D' : np.random.randn(10)})

df
```

[ ]:
```
df.groupby('A').sum()
```

[ ]:
```
df.groupby('B').mean()
```

[ ]:
```
df.groupby(['A','B']).median()
```

1.9 Pivot Tables

A pivot table in Pandas is a powerful tool to summarize, reshape, and analyze data. It is similar to pivot tables in spreadsheet software like Microsoft Excel. It allows you to aggregate data based on one or more keys and apply functions like sum, mean, count, etc.

[ ]:
```
import pandas as pd
data = {
    'Department': ['HR', 'HR', 'IT', 'IT', 'Finance', 'Finance'],
    'Employee': ['John', 'Sarah', 'Om', 'David', 'Rupa', 'Yash'],
    'Salary': [50000, 65000, 75000, 40000, 100000, 75000],
    'Experience': [5, 7, 10, 12, 15, 9]
}
```

[ ]:
```
df = pd.DataFrame(data)
```

[ ]:
```
df.to_csv('salary1.csv')
```

[121]:
```
df1 = pd.read_csv('salary1.csv')
```

[ ]:
```
# Create a pivot table
pivot = pd.pivot_table(df1, values='Salary', index='Department', aggfunc='mean')
print(pivot)
```

[ ]:
```
pivot = pd.pivot_table(df1, values=['Salary', 'Experience'],
                       index='Department', aggfunc=['mean', 'max'])
print(pivot)
```

[ ]:
```
df = pd.DataFrame({'A' : ['one', 'one', 'two', 'three'] * 3,
   .....:                    'B' : ['A', 'B', 'C'] * 4,
   .....:                    'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] *␣
 ↪2,
   .....:                    'D' : np.random.randn(12),
   .....:                    'E' : np.random.randn(12)})
   .....:
df
```

[ ]:
```
pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
```

[ ]:
```
pip install seaborn
```

1.10 Working with Iris data set

[123]:
```
import seaborn as sns
iris = sns.load_dataset('iris')
```

[130]:
```
iris.shape
```

[130]: (150, 5)

[ ]:
```
iris = pd.read_csv('iris.csv')
```

[ ]:
```
print(iris.shape)
```

[131]:
```
print(iris.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   sepal_length  150 non-null    float64
 1   sepal_width   150 non-null    float64
 2   petal_length  150 non-null    float64
 3   petal_width   150 non-null    float64
 4   species       150 non-null    object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
None
```

[132]:
```
print(iris.describe())
```

```
       sepal_length  sepal_width  petal_length  petal_width
count    150.000000   150.000000    150.000000   150.000000
mean       5.843333     3.057333      3.758000     1.199333
std        0.828066     0.435866      1.765298     0.762238
min        4.300000     2.000000      1.000000     0.100000
25%        5.100000     2.800000      1.600000     0.300000
50%        5.800000     3.000000      4.350000     1.300000
75%        6.400000     3.300000      5.100000     1.800000
max        7.900000     4.400000      6.900000     2.500000
```

[133]:
```
print(iris.isnull().sum())
```

```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

[134]:
```
print(iris['species'].unique())
```

```
['setosa' 'versicolor' 'virginica']
```

[ ]:
```
print(iris['species'].value_counts())
```

[135]:
```
print(iris.groupby('species').mean())
```

```
            sepal_length  sepal_width  petal_length  petal_width
species
setosa             5.006        3.428         1.462        0.246
versicolor         5.936        2.770         4.260        1.326
virginica          6.588        2.974         5.552        2.026
```

[ ]:
```
print(iris.groupby('species')['petal_length'].median())
```

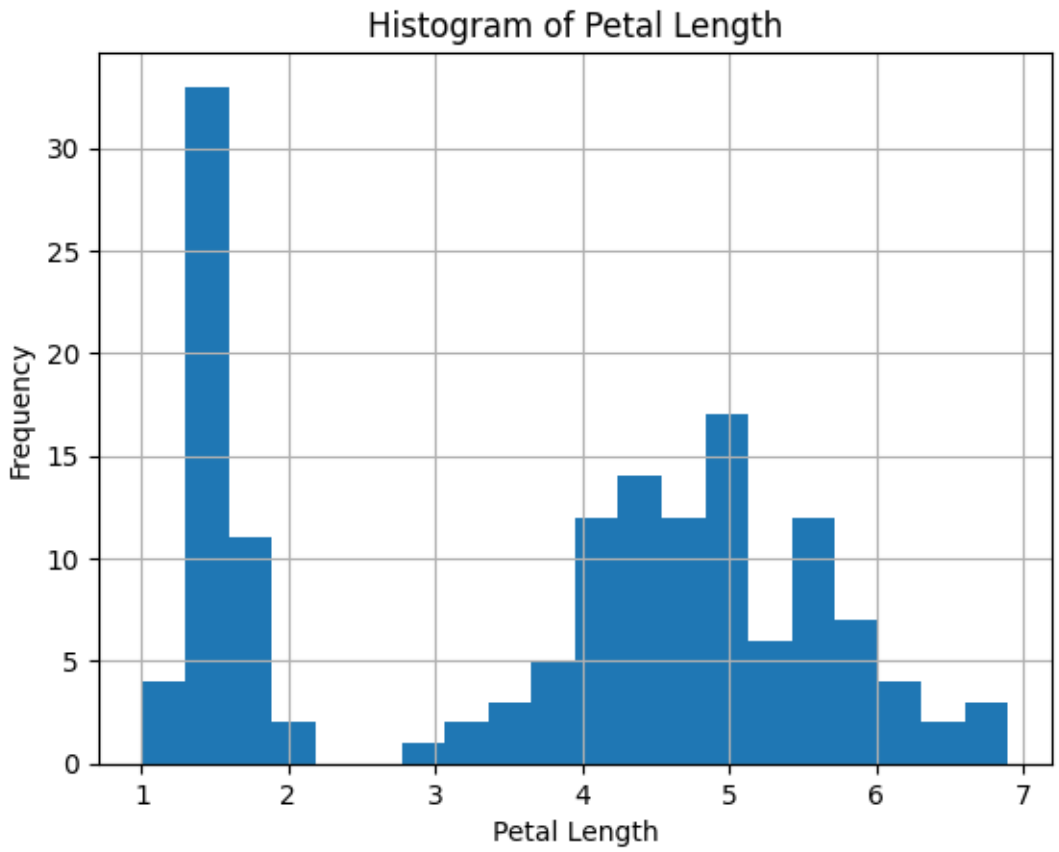
[136]:
```
# Filter flowers with petal length > 5
filtered = iris[iris['petal_length'] > 5]
print(filtered.head())
```

```
     sepal_length  sepal_width  petal_length  petal_width     species
83            6.0          2.7           5.1          1.6  versicolor
100           6.3          3.3           6.0          2.5   virginica
101           5.8          2.7           5.1          1.9   virginica
102           7.1          3.0           5.9          2.1   virginica
103           6.3          2.9           5.6          1.8   virginica
```

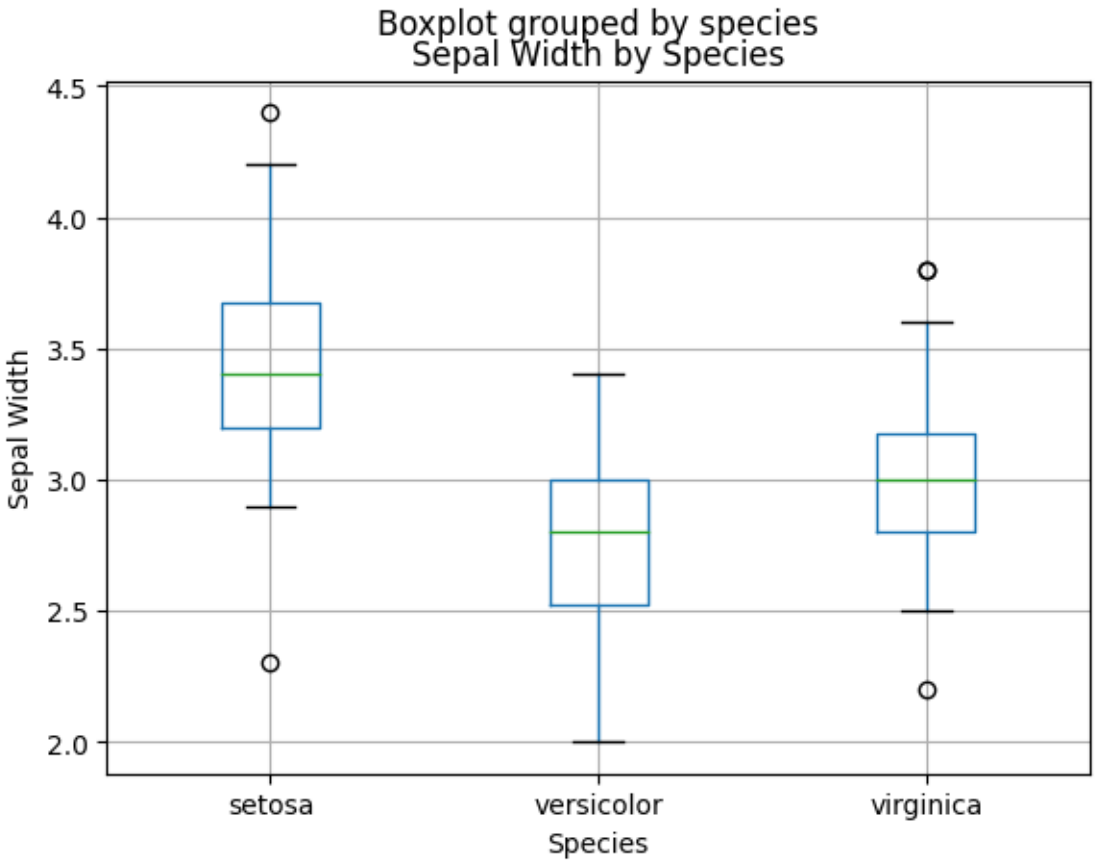
[137]:
```
# Filter by species
setosa = iris[iris['species'] == 'setosa']
print(setosa.head())
```

```
   sepal_length  sepal_width  petal_length  petal_width species
0           5.1          3.5           1.4          0.2  setosa
1           4.9          3.0           1.4          0.2  setosa
2           4.7          3.2           1.3          0.2  setosa
3           4.6          3.1           1.5          0.2  setosa
4           5.0          3.6           1.4          0.2  setosa
```

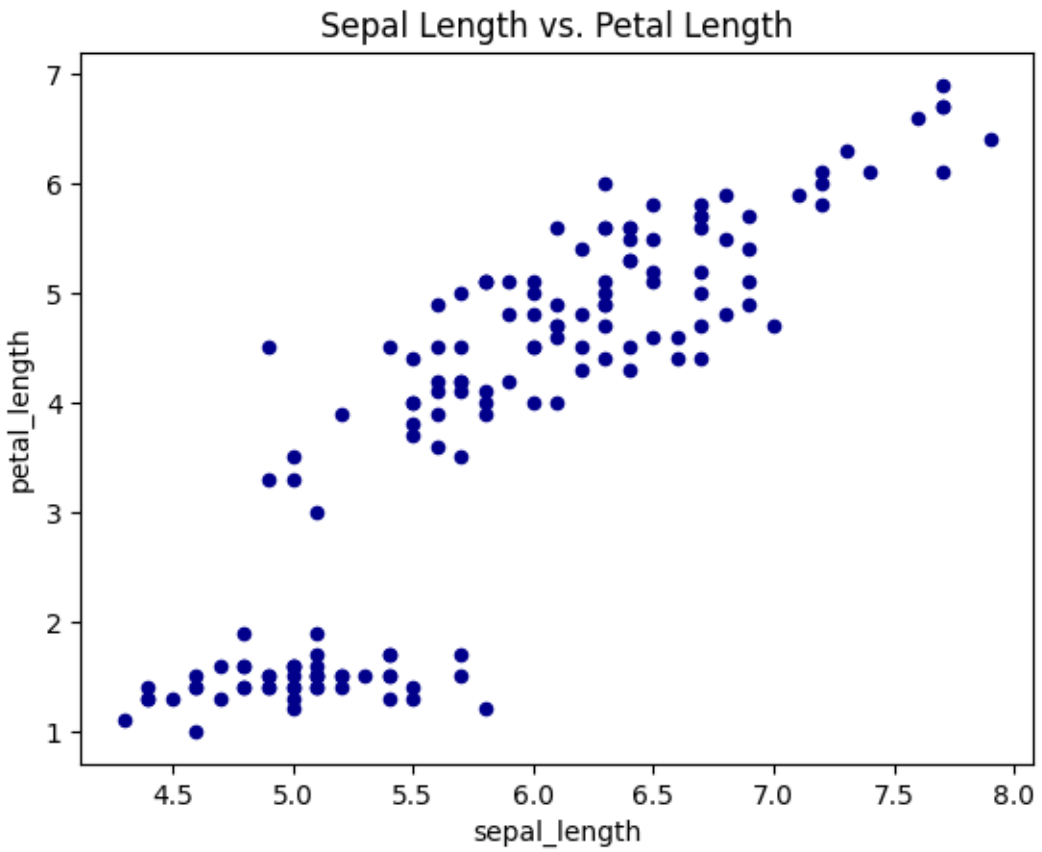
[138]:
```
import matplotlib.pyplot as plt
# Histogram of petal lengths
iris['petal_length'].hist(bins=20)
plt.title('Histogram of Petal Length')
plt.xlabel('Petal Length')
plt.ylabel('Frequency')
plt.show()
```



[139]:
```
# Boxplot of sepal width by species
iris.boxplot(column='sepal_width', by='species')
plt.title('Sepal Width by Species')
plt.xlabel('Species')
plt.ylabel('Sepal Width')
plt.show()
```



[140]:
```
# Scatter plot of sepal length vs. petal length
iris.plot.scatter(x='sepal_length', y='petal_length', c='DarkBlue')
plt.title('Sepal Length vs. Petal Length')
plt.show()
```



[ ]:
```
import pandas as pd
import numpy as np
ts = pd.Series(np.random.randn(1000), index=pd.date_range("1/1/2000",␣
 ↪periods=1000))
ts = ts.cumsum()
ts.plot();
```

[ ]:
```
import matplotlib.pyplot as plt

df = pd.DataFrame(
    np.random.randn(1000, 4), index=ts.index, columns=["A", "B", "C", "D"]
)

df = df.cumsum()

plt.figure();

df.plot();
```

```
plt.legend(loc='best');
```

[143]:
```
!pip install yfinance
```

```
DEPRECATION: mermaid 0.3.2 has a non-standard dependency specifier
torch>=1.7torchvision. pip 23.3 will enforce this behaviour change. A possible
replacement is to upgrade to a newer version of mermaid or contact the author to
suggest that they release a version with a conforming dependency specifiers.
Discussion can be found at https://github.com/pypa/pip/issues/12063
  WARNING: The script sample.exe is installed in
'C:\Users\Ajit\AppData\Roaming\Python\Python311\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
warning, use --no-warn-script-location.

Defaulting to user installation because normal site-packages is not writeable
Collecting yfinance
  Obtaining dependency information for yfinance from https://files.pythonhosted.
org/packages/b2/38/7533745b517c34b7b749a7a21f631711354a3d4d39a840d75d20c94d71a0/
yfinance-0.2.51-py2.py3-none-any.whl.metadata
  Downloading yfinance-0.2.51-py2.py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.3.0 in
c:\users\ajit\appdata\roaming\python\python311\site-packages (from yfinance)
(1.5.3)
Requirement already satisfied: numpy>=1.16.5 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (2.31.0)
Collecting multitasking>=0.0.7 (from yfinance)
  Obtaining dependency information for multitasking>=0.0.7 from https://files.py
thonhosted.org/packages/3e/8a/bb3160e76e844db9e69a413f055818969c8acade64e1a9ac5c
e9dfdcf6c1/multitasking-0.0.11-py3-none-any.whl.metadata
  Downloading multitasking-0.0.11-py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: lxml>=4.9.1 in c:\programdata\anaconda3\lib\site-
packages (from yfinance) (4.9.3)
Requirement already satisfied: platformdirs>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (2023.3.post1)
Collecting frozendict>=2.3.4 (from yfinance)
  Obtaining dependency information for frozendict>=2.3.4 from https://files.pyth
onhosted.org/packages/04/13/d9839089b900fa7b479cce495d62110cddc4bd5630a04d846991
6c0e79c5/frozendict-2.4.6-py311-none-any.whl.metadata
  Downloading frozendict-2.4.6-py311-none-any.whl.metadata (23 kB)
Collecting peewee>=3.16.2 (from yfinance)
  Using cached peewee-3.17.8.tar.gz (948 kB)
  Installing build dependencies: started
  Installing build dependencies: finished with status 'done'
  Getting requirements to build wheel: started
```

```
  Getting requirements to build wheel: finished with status 'done'
  Preparing metadata (pyproject.toml): started
  Preparing metadata (pyproject.toml): finished with status 'done'
Requirement already satisfied: beautifulsoup4>=4.11.1 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (4.12.2)
Collecting html5lib>=1.1 (from yfinance)
  Obtaining dependency information for html5lib>=1.1 from https://files.pythonho
sted.org/packages/6c/dd/a834df6482147d48e225a49515aabc28974ad5a4ca3215c18a882565
b028/html5lib-1.1-py2.py3-none-any.whl.metadata
  Downloading html5lib-1.1-py2.py3-none-any.whl.metadata (16 kB)
Requirement already satisfied: soupsieve>1.2 in
c:\programdata\anaconda3\lib\site-packages (from
beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in c:\programdata\anaconda3\lib\site-
packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in
c:\programdata\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance)
(0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance)
(2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
(1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
(2023.7.22)
Using cached yfinance-0.2.51-py2.py3-none-any.whl (104 kB)
Downloading frozendict-2.4.6-py311-none-any.whl (16 kB)
Using cached html5lib-1.1-py2.py3-none-any.whl (112 kB)
Using cached multitasking-0.0.11-py3-none-any.whl (8.5 kB)
Building wheels for collected packages: peewee
  Building wheel for peewee (pyproject.toml): started
  Building wheel for peewee (pyproject.toml): finished with status 'done'
  Created wheel for peewee: filename=peewee-3.17.8-py3-none-any.whl size=139064
sha256=6c28339546b130b45e7d0741be197f9b6621afddc4cb20e7a3ed3005c8c92fd7
  Stored in directory: c:\users\ajit\appdata\local\pip\cache\wheels\ff\6c\15\506
e25bc390de450a7fa53c155cd9b0fbd13ad3e84a9abc183
Successfully built peewee
Installing collected packages: peewee, multitasking, html5lib, frozendict,
yfinance
Successfully installed frozendict-2.4.6 html5lib-1.1 multitasking-0.0.11
peewee-3.17.8 yfinance-0.2.51
```

```
Defaulting to user installation because normal site-packages is not writeable
Collecting yfinance
  Obtaining dependency information for yfinance from https://files.pythonhosted.
org/packages/b2/38/7533745b517c34b7b749a7a21f631711354a3d4d39a840d75d20c94d71a0/
yfinance-0.2.51-py2.py3-none-any.whl.metadata
  Using cached yfinance-0.2.51-py2.py3-none-any.whl.metadata (5.5 kB)
Requirement already satisfied: pandas>=1.3.0 in
c:\users\ajit\appdata\roaming\python\python311\site-packages (from yfinance)
(1.5.3)
Requirement already satisfied: numpy>=1.16.5 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (1.24.3)
Requirement already satisfied: requests>=2.31 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (2.31.0)
Requirement already satisfied: multitasking>=0.0.7 in
c:\users\ajit\appdata\roaming\python\python311\site-packages (from yfinance)
(0.0.11)
Requirement already satisfied: lxml>=4.9.1 in c:\programdata\anaconda3\lib\site-
packages (from yfinance) (4.9.3)
Requirement already satisfied: platformdirs>=2.0.0 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (3.10.0)
Requirement already satisfied: pytz>=2022.5 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (2023.3.post1)
Collecting frozendict>=2.3.4 (from yfinance)
  Obtaining dependency information for frozendict>=2.3.4 from https://files.pyth
onhosted.org/packages/04/13/d9839089b900fa7b479cce495d62110cddc4bd5630a04d846991
6c0e79c5/frozendict-2.4.6-py311-none-any.whl.metadata
  Using cached frozendict-2.4.6-py311-none-any.whl.metadata (23 kB)
Requirement already satisfied: peewee>=3.16.2 in
c:\users\ajit\appdata\roaming\python\python311\site-packages (from yfinance)
(3.17.8)
Requirement already satisfied: beautifulsoup4>=4.11.1 in
c:\programdata\anaconda3\lib\site-packages (from yfinance) (4.12.2)
Requirement already satisfied: html5lib>=1.1 in
c:\users\ajit\appdata\roaming\python\python311\site-packages (from yfinance)
(1.1)
Requirement already satisfied: soupsieve>1.2 in
c:\programdata\anaconda3\lib\site-packages (from
beautifulsoup4>=4.11.1->yfinance) (2.4)
Requirement already satisfied: six>=1.9 in c:\programdata\anaconda3\lib\site-
packages (from html5lib>=1.1->yfinance) (1.16.0)
Requirement already satisfied: webencodings in
c:\programdata\anaconda3\lib\site-packages (from html5lib>=1.1->yfinance)
(0.5.1)
Requirement already satisfied: python-dateutil>=2.8.1 in
c:\programdata\anaconda3\lib\site-packages (from pandas>=1.3.0->yfinance)
(2.8.2)
Requirement already satisfied: charset-normalizer<4,>=2 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
```

```
(2.0.4)
Requirement already satisfied: idna<4,>=2.5 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
(1.26.16)
Requirement already satisfied: certifi>=2017.4.17 in
c:\programdata\anaconda3\lib\site-packages (from requests>=2.31->yfinance)
(2023.7.22)
Using cached yfinance-0.2.51-py2.py3-none-any.whl (104 kB)
Using cached frozendict-2.4.6-py311-none-any.whl (16 kB)
Installing collected packages: frozendict, yfinance
Successfully installed frozendict-2.4.6 yfinance-0.2.51

DEPRECATION: mermaid 0.3.2 has a non-standard dependency specifier
torch>=1.7torchvision. pip 23.3 will enforce this behaviour change. A possible
replacement is to upgrade to a newer version of mermaid or contact the author to
suggest that they release a version with a conforming dependency specifiers.
Discussion can be found at https://github.com/pypa/pip/issues/12063
  WARNING: The script sample.exe is installed in
'C:\Users\Ajit\AppData\Roaming\Python\Python311\Scripts' which is not on PATH.
  Consider adding this directory to PATH or, if you prefer to suppress this
warning, use --no-warn-script-location.
```

1.11 Working with stock data

[144]:
```
import yfinance as yf
import pandas as pd
import matplotlib.pyplot as plt
```

[148]:
```
# Fetch stock data
data = yf.download('AAPL', start='2020-01-01', end='2022-12-31')
```

```
[**********************100%**********************]  1 of 1 completed
```

[149]:
```
data.head()
```

[149]:

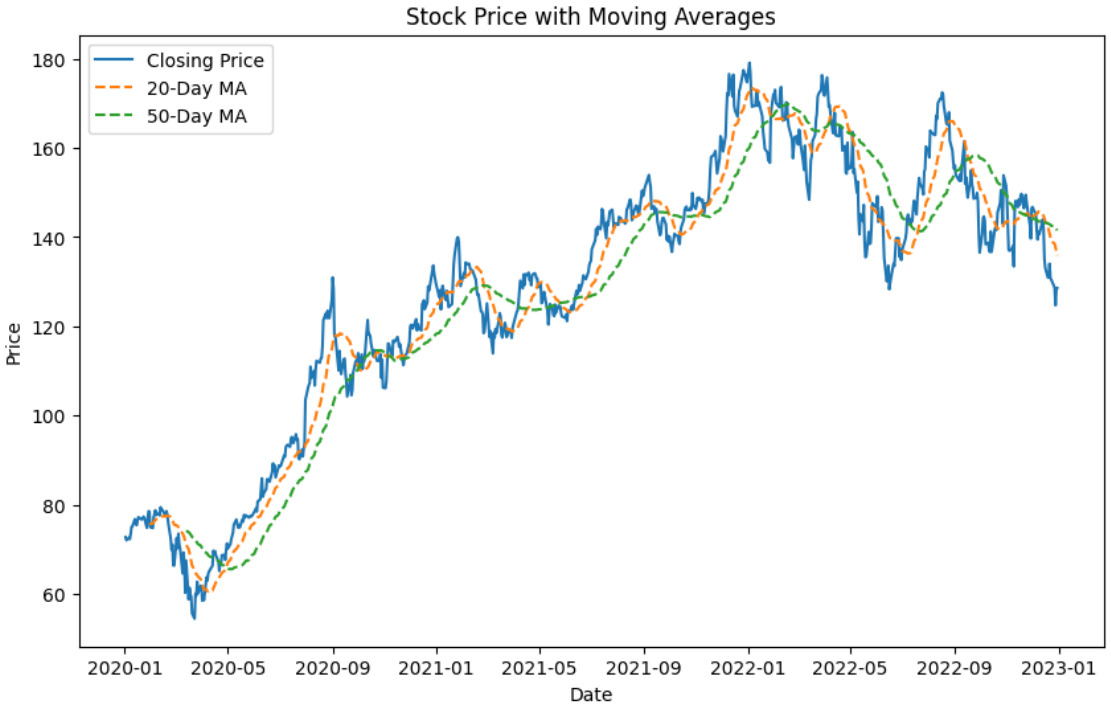
Price	Close	High	Low	Open	Volume
Ticker	AAPL	AAPL	AAPL	AAPL	AAPL
Date					
2020-01-02	72.796036	72.856628	71.545402	71.799888	135480400
2020-01-03	72.088287	72.851753	71.862884	72.020424	146322800
2020-01-06	72.662720	72.701500	70.954010	71.206077	118387200
2020-01-07	72.320976	72.929322	72.100418	72.672409	108872000
2020-01-08	73.484344	73.787308	72.022850	72.022850	132079200

[150]:
```
data.shape
```

[150]:
```
(756, 5)
```

[151]:
```
# Add features
data['Daily Return'] = data['Close', 'AAPL'].pct_change()
data['20-Day MA'] = data['Close', 'AAPL'].rolling(window=20).mean()
data['50-Day MA'] = data['Close', 'AAPL'].rolling(window=50).mean()

# Visualization
plt.figure(figsize=(10, 6))
plt.plot(data['Close', 'AAPL'], label='Closing Price')
plt.plot(data['20-Day MA'], label='20-Day MA', linestyle='--')
plt.plot(data['50-Day MA'], label='50-Day MA', linestyle='--')
plt.title('Stock Price with Moving Averages')
plt.xlabel('Date')
plt.ylabel('Price')
plt.legend()
plt.show()
```



1.12 Assignment Problmes (explorative)

1. Basic DataFrame Operations

- Create a DataFrame using a dictionary.
- Display the first 5 rows of the DataFrame.
- Retrieve the shape, column names, and data types of the DataFrame.
- Rename one of the columns in the DataFrame.
- Add an extra column

2. Data Selection

- All rows where a specific column has a value greater than a given number.
- All columns where a specific column has a value greater than a given number.
- A subset of columns.
- A specific row by its index.

3. Missing Values Treatments

- Create a DataFrame with some missing values.
- Identify the missing values.
- Replace the missing values with a fixed value, mean of column, median of a column, interpolation

4. Grouping and Aggregation

- Create a DataFrame with columns like Branch, Students, and CGPA
- Group the data by by branch and calculate.
- What more grouping can you do?

5. Merging and Joining (use help on pandas)

- Create two DataFrames
- One with Students IDs and Names.
- Another with IDs IDs and CGPA.
- Merge the two DataFrames on Students IDs.
- Perform different types of joins: inner, left, and right

6. Handling Time Series Data

- Download some stock data
- Analyse this data
- Create appropriate plots