1. **CAST OBJECTS TO A DATA TYPE**

SELECT customerNumber,

    COUNT(\*) AS number\_payments,

    MIN(CAST(amount AS INT)) AS min\_purchase,

    MAX(CAST(amount AS INT))  AS max\_purchase,

    AVG(CAST(amount AS INT)) AS avg\_purchase,

    SUM(CAST(amount AS INT)) AS total\_spent

FROM payments

pd.read\_sql('''

select cast(round(priceEach) as INTEGER) as rounded\_price\_int

        from orderDetails

            ''',conn)

**2.Strip year or month from date as a string object**

WHERE strftime('%Y',paymentDate)='2004'



pd.read\_sql('''

select orderDate,

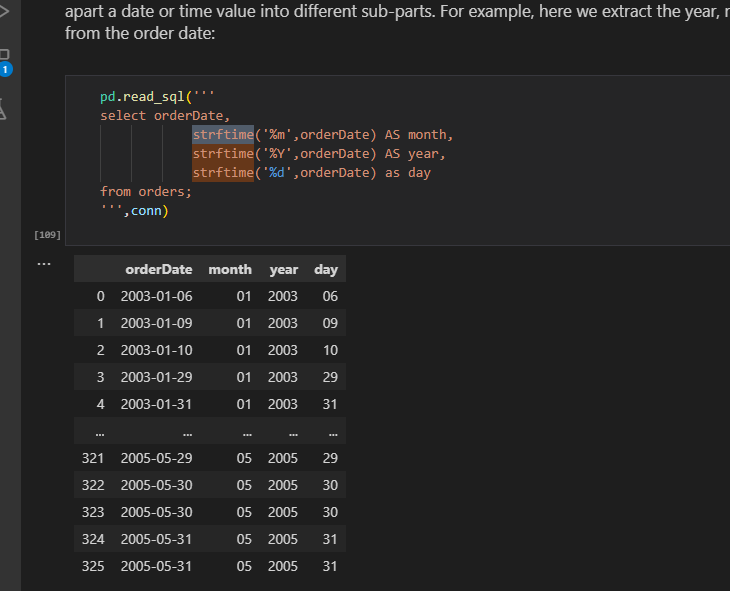
            strftime('%m',orderDate) AS month,

            strftime('%Y',orderDate) AS year,

            strftime('%d',orderDate) as day

from orders;

''',conn)



**Or use substr method**

pd.read\_sql('''

SELECT customerNumber,

    COUNT(\*) AS number\_payments,

    MIN(CAST(amount AS INT)) AS min\_purchase,

    MAX(CAST(amount AS INT))  AS max\_purchase,

    AVG(CAST(amount AS INT)) AS avg\_purchase,

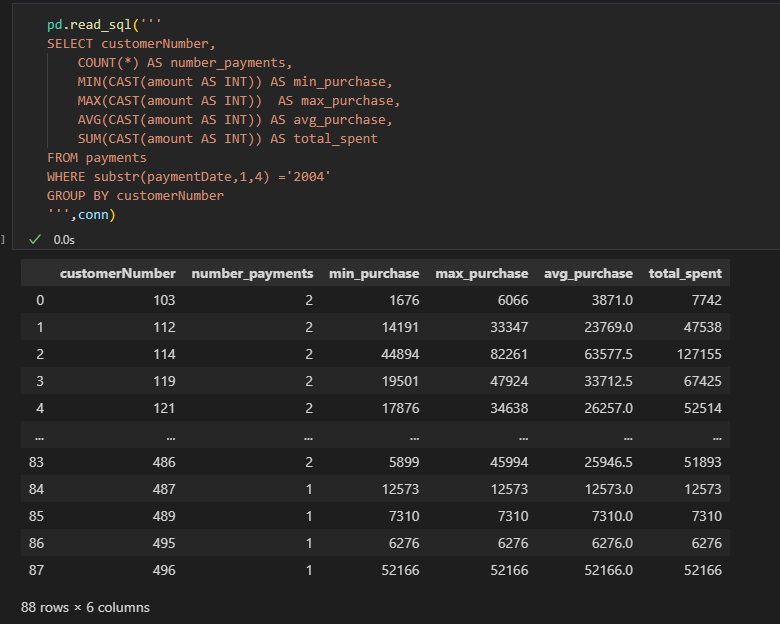
    SUM(CAST(amount AS INT)) AS total\_spent

FROM payments

WHERE substr(paymentDate,1,4) ='2004'

GROUP BY customerNumber

''',conn)



**3.Convert select statement to dataframe**

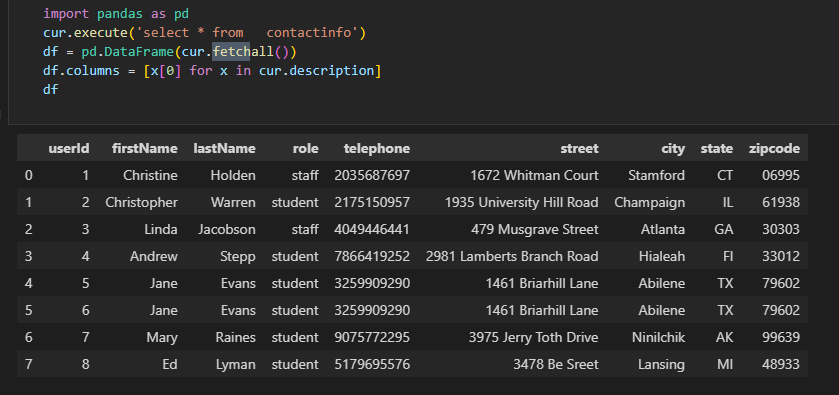
import pandas as pd

cur.execute('select \* from   contactinfo')

df = pd.DataFrame(cur.fetchall())

df.columns = [x[0] for x in cur.description]

df



* 4. Highest -**altitude**
* Southern/northern – **latitude eg** northern-most airport is the highest latitude

Southern-most – smallest latitude

**5.Pandasql Error**

**----> 6** passenger\_names **=** pysqldf**(**q**)**

**ImportError**: Unable to find a usable engine; tried using: 'sqlalchemy'.

A suitable version of sqlalchemy is required for sql I/O support.

Trying to import the above resulted in these errors:

**- Pandas requires version '1.4.0' or newer of 'sqlalchemy' (version '1.3.19' currently installed).**

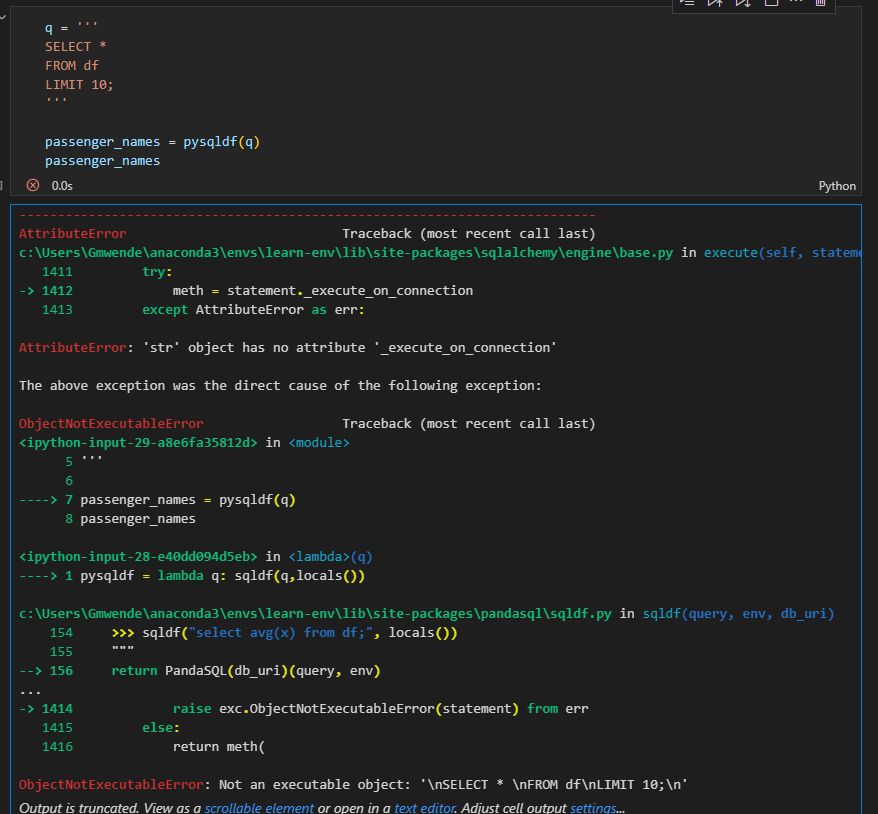
**TO update use**

conda update sqlalchemy

Check version if updated

pip show sqlalchemy

6.



Works well in colab

**7.Put dataframe in memory as to use conn**

#put df to memory

import sqlite3

conn = sqlite3.connect(':memory:')

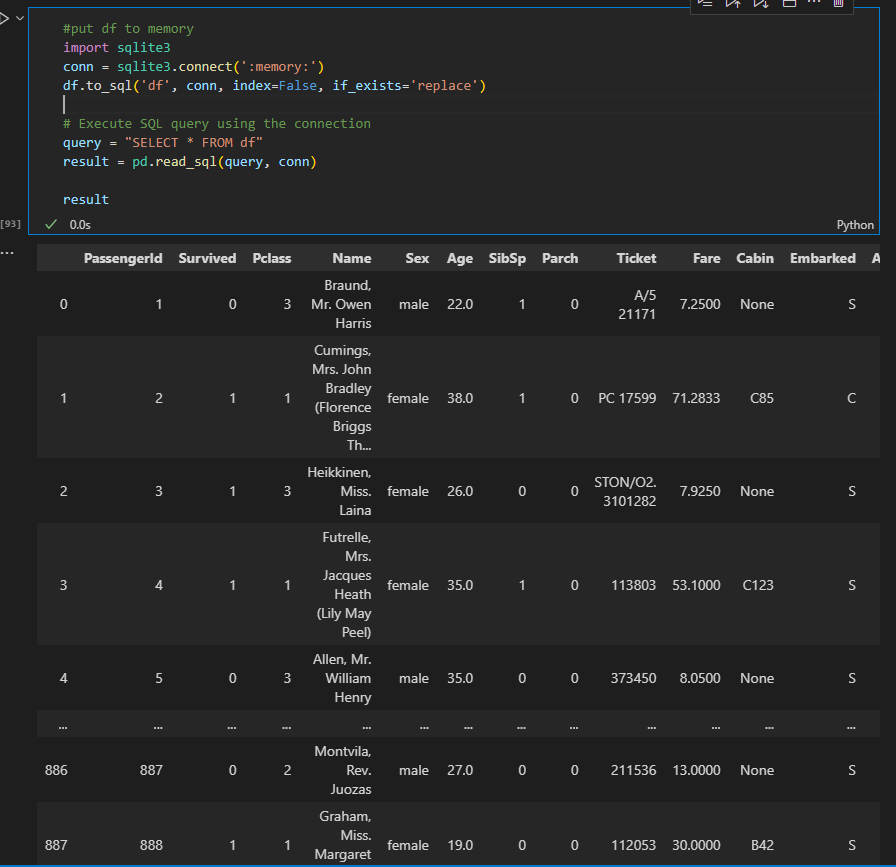
df.to\_sql('df', conn, index=False, if\_exists='replace')

# Execute SQL query using the connection

query = "SELECT \* FROM df"

result = pd.read\_sql(query, conn)

result



**8.Get female and children that is female and male less than or equal to 15**

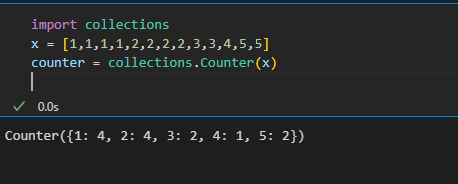
df[(df['Sex'] == 'female') | (df['Age'] <= 15)]

**9.Select everyonelse other than 1**

df[df['Pclass'] != '1']

**10.calculate totals using counter(get frequency for each value)**

counter = collections.Counter(x)



11. **Create two vertical subplots sharing 15% and 85% of plot space**

**Create density instead of count on seaborn histogram**

*#Create two vertical subplots sharing 15% and 85% of plot space*

*#sharex allows sharing of axes i.e building multiple plots on the same axes*

*fig, (ax,ax2) = plt.subplots(2,sharex=True,gridspec\_kw={'height\_ratios':(.15,.85)},figsize=(10,8))*

*sns.histplot(data['Height'],*

*lw=2,*

*edgecolor='r',*

*alpha=0.4,*

*color='w',*

*label='Histogram',*

*stat='density',*

*ax=ax2*

*)*

*sns.kdeplot(data.Height,*

*lw=3,*

*color='b',*

*label='Kernerl Density Estimation plot',*

*alpha=0.7,*

*ax=ax2*

*)*

*mean = data.Height.mean()*

*std = data.Height.std()*

*parametric\_dist = stats.norm(loc=mean, scale=std)*

*x=np.linspace(parametric\_dist.ppf(0.01),parametric\_dist.ppf(0.99),100)*

*ax2.plot(x,*

*parametric\_dist.pdf(x),*

*color='g',*

*alpha=0.7,*

*lw=3,*

*label = 'Parametric Fit'*

*)*

*ax2.set\_title('Density Estimations')*

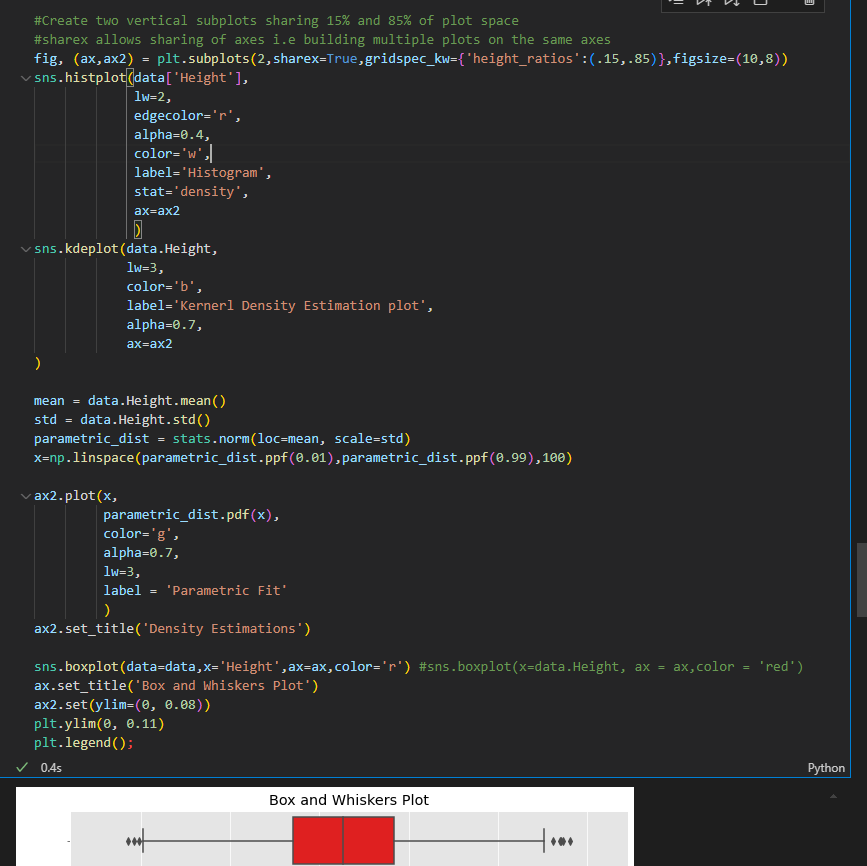
*sns.boxplot(data=data,x='Height',ax=ax,color='r') #sns.boxplot(x=data.Height, ax = ax,color = 'red')*

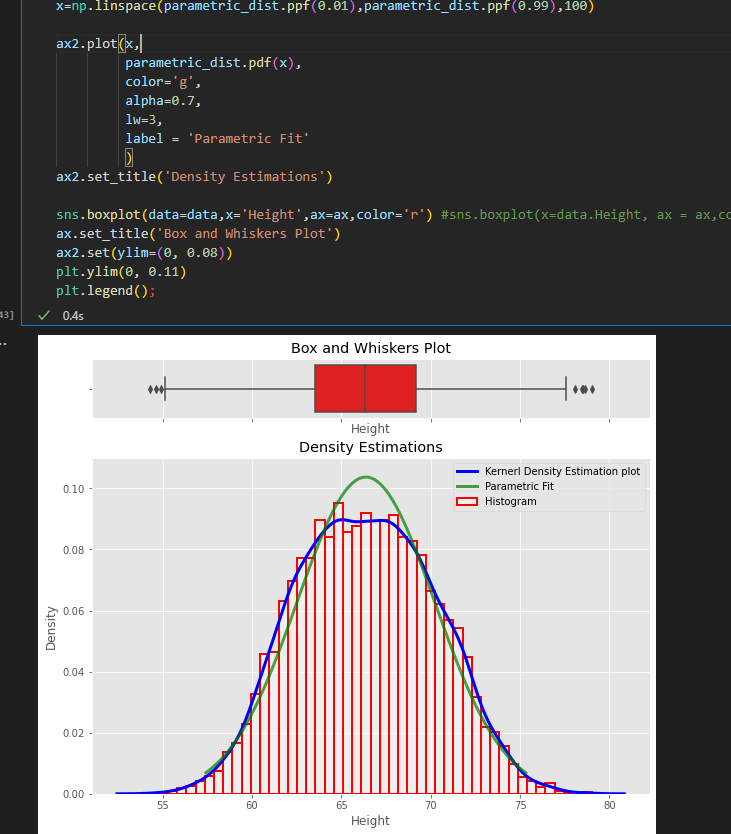
*ax.set\_title('Box and Whiskers Plot')*

*ax2.set(ylim=(0, 0.08))*

*plt.ylim(0, 0.11)*

*plt.legend();*





**12. Add density (probability) instead of counts in matplotlib histogram**

xtick\_locations = range(1,6)

bins = np.arange(6) +0.5 #[0.5, 1.5, 2.5, 3.5, 4.5, 5.5]

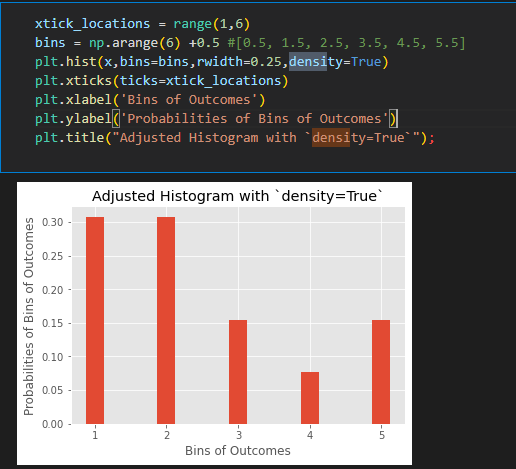
plt.hist(x,bins=bins,rwidth=0.25,density=True)

plt.xticks(ticks=xtick\_locations)

plt.xlabel('Bins of Outcomes')

plt.ylabel('Probabilities of Bins of Outcomes')

plt.title("Adjusted Histogram with `density=True`");



**13. ttest**

t= (x\_bar-mu)/(sample\_std/np.sqrt(25))

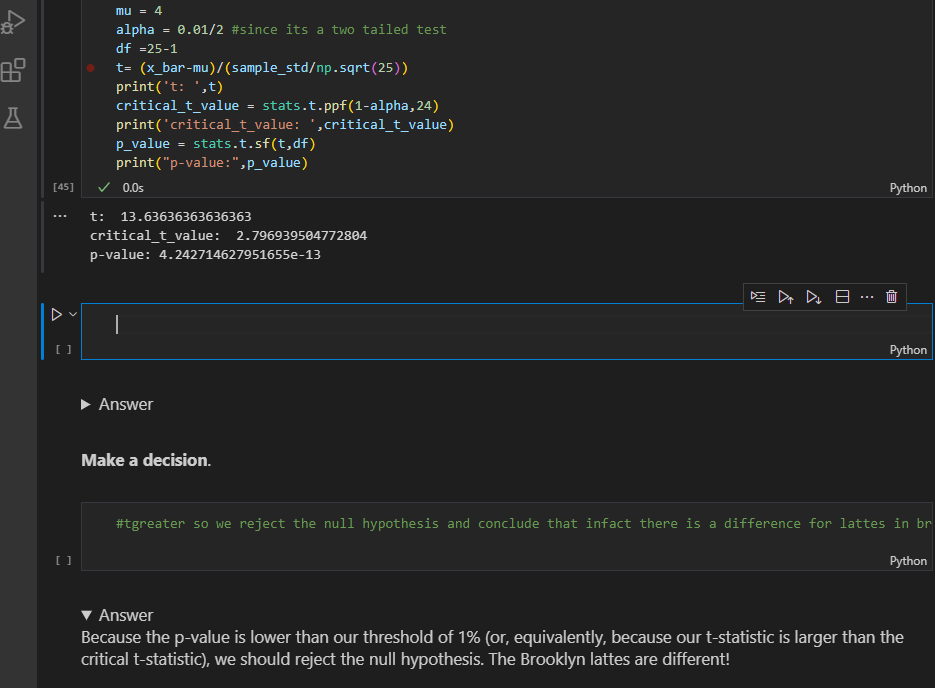
print('t: ',t)

critical\_t\_value = stats.t.ppf(1-alpha,24)

print('critical\_t\_value: ',critical\_t\_value)

p\_value = stats.t.sf(t,df)

print("p-value:",p\_value)



b)Example 2#one tailed left tail

sample =[20, 30, 30, 50, 75, 25, 30, 30, 40, 80]

x\_bar =  np.mean(sample)

sample\_std = np.std(sample,ddof=1)

n=len(sample)

df=n-1

mu =58

t\_stat1 = stats.ttest\_1samp(a=sample,popmean=58)

print('t\_stat1:', t\_stat1[0])

print('alpha:', t\_stat1[1]/2)

t\_stat2 = (x\_bar-mu)/(sample\_std/np.sqrt(n))

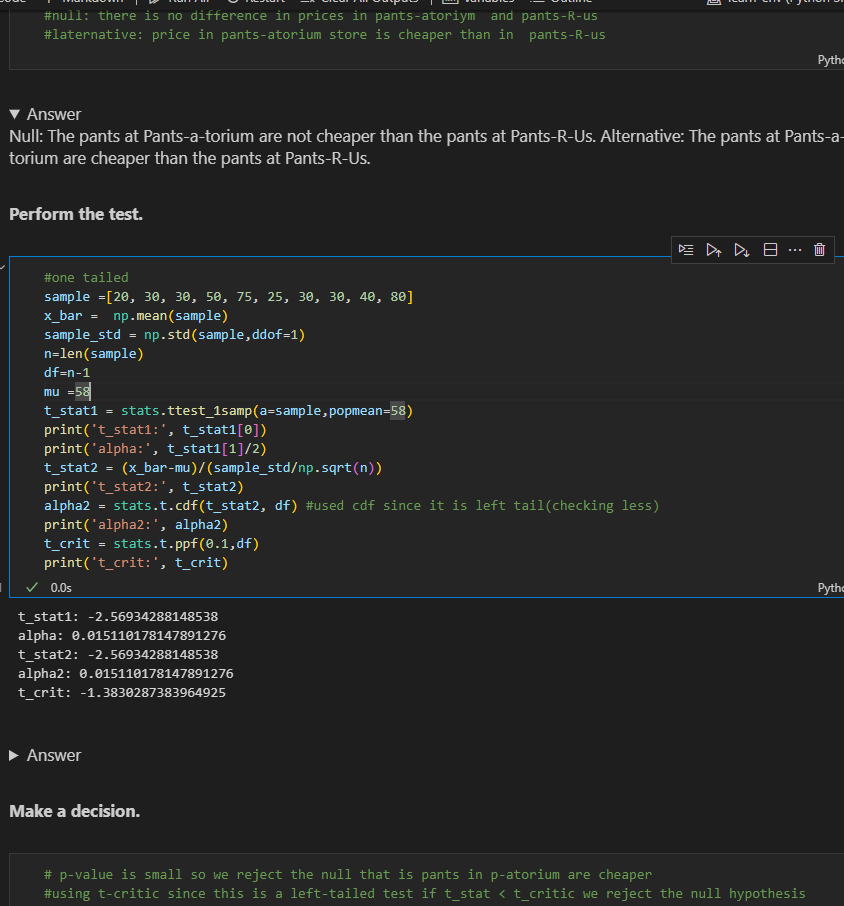
print('t\_stat2:', t\_stat2)

alpha2 = stats.t.cdf(t\_stat2, df) #used cdf since it is left tail(checking less)

print('alpha2:', alpha2)

t\_crit = stats.t.ppf(0.1,df)

print('t\_crit:', t\_crit)



**c)two-sample t-test**

**delivery\_times\_A = [28.4, 23.3, 30.4, 28.1, 29.4, 30.6, 27.8, 30.9, 27.0, 32.8]**

**mean\_A = np.mean(delivery\_times\_A)**

**std\_A = np.std(delivery\_times\_A)**

**nobs\_A = len(delivery\_times\_A)**

**mean\_B = 26.8**

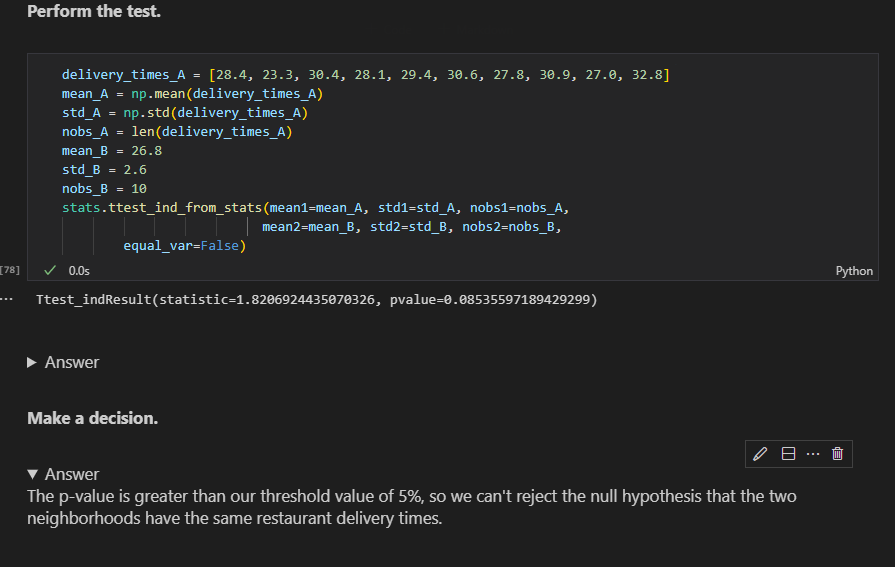
**std\_B = 2.6**

**nobs\_B = 10**

**stats.ttest\_ind\_from\_stats(mean1=mean\_A, std1=std\_A, nobs1=nobs\_A,**

**mean2=mean\_B, std2=std\_B, nobs2=nobs\_B,**

**equal\_var=False)**



**c)2 sample again two tailed**

**high\_protein = [134, 146, 104, 119, 124, 161, 107, 83, 113, 129, 97, 123]**

**low\_protein = [70, 118, 101, 85, 107, 132, 94]**

**stats.ttest\_ind(a=high\_protein, b=low\_protein)**



**d)2 sample one tailed**

h\_bar = np.mean(high\_protein)

l\_bar = np.mean(low\_protein)

h\_df = len(high\_protein) - 1

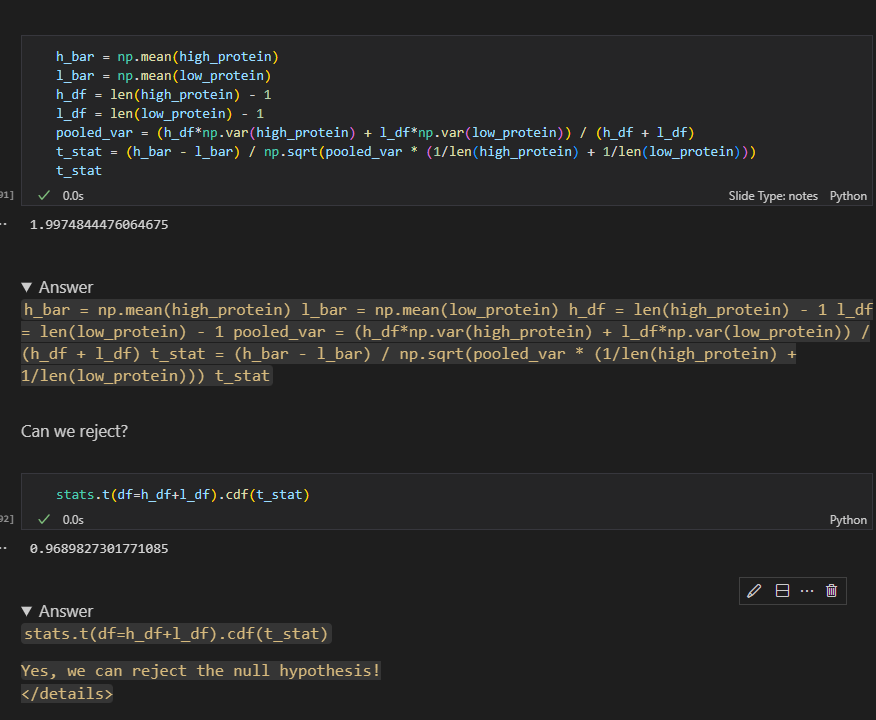
l\_df = len(low\_protein) - 1

pooled\_var = (h\_df\*np.var(high\_protein) + l\_df\*np.var(low\_protein)) / (h\_df + l\_df)

t\_stat = (h\_bar - l\_bar) / np.sqrt(pooled\_var \* (1/len(high\_protein) + 1/len(low\_protein)))

t\_stat

stats.t(df=h\_df+l\_df).cdf(t\_stat)



**SQL SUMMARY**

* We will primarily use SQLite in these lessons because it is lightweight and portable (and therefore useful for educational purposes)
* SQLite is a C library that provides lightweight disk-bases databse that doesn’t require a separate server process and allows accessing the database using a nonstandard variant of the SQL query language.
* Some applications can use SQLite for internal data storage.Its also possible to prototype an application using sqlite and then port the code to a larger databse such as postresgreSQL or Oracle



1. **ERD** (Entity Relationship Diagram)- shows the relationship between tables. It does not give us any information about the specific data stored in the database, but rather the metadata
2. **Connect to sqlite 3**

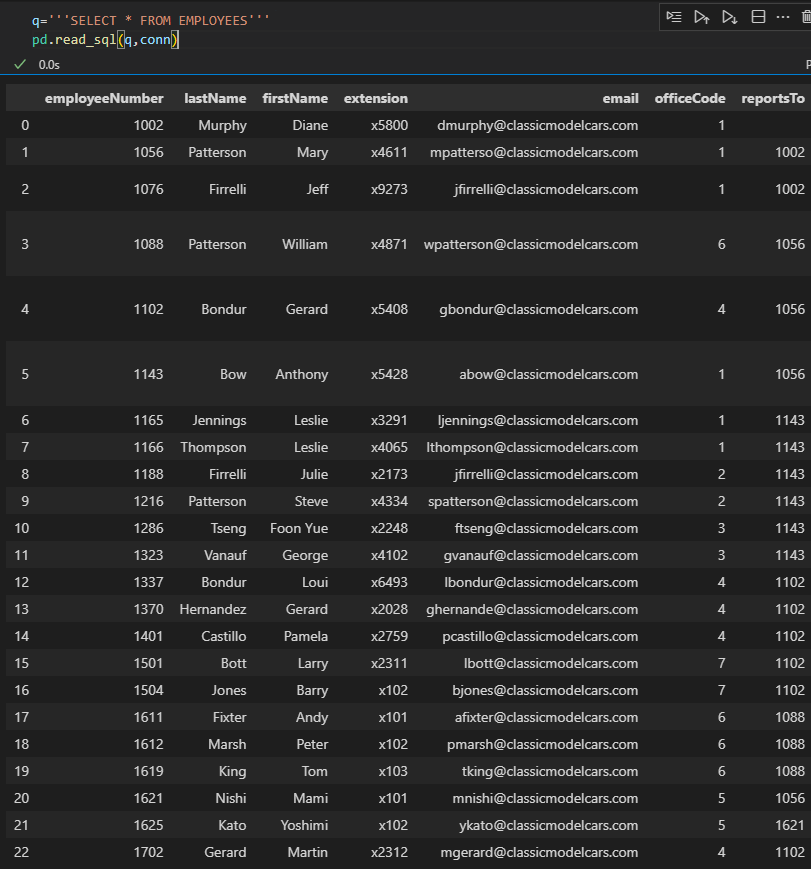
import sqlite3

conn = sqlite3.connect('data.sqlite')

1. **Select Records**

q='''SELECT \* FROM EMPLOYEES'''

pd.read\_sql(q,conn)



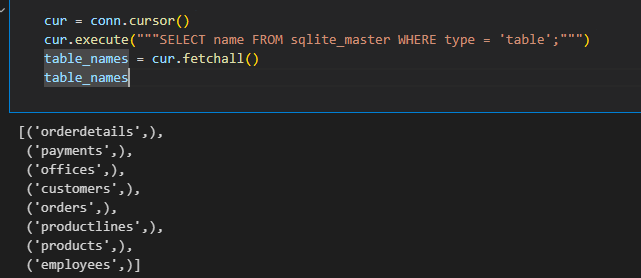
1. Check table in a database(Note that the .execute() method didn't actually return our data. The data is now just available in our cursor object. We'll use the .fetchall() method to get all the rows from our query.)

cur = conn.cursor()

cur.execute("""SELECT name FROM sqlite\_master WHERE type = 'table';""")

table\_names = cur.fetchall()

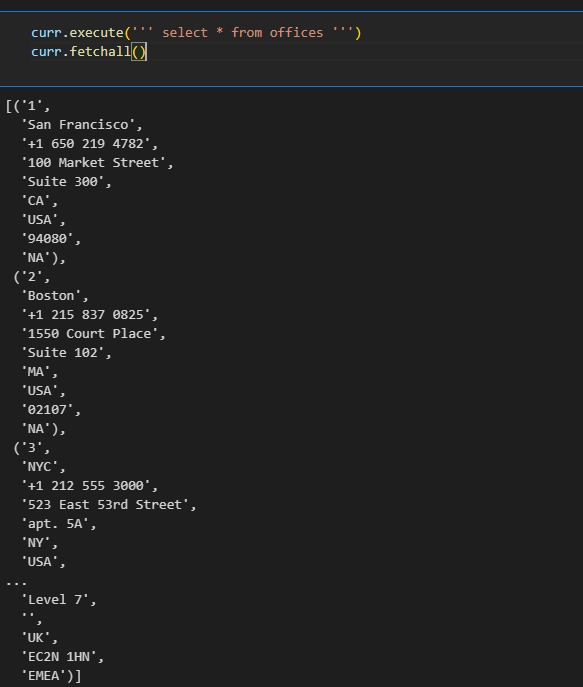
table\_names



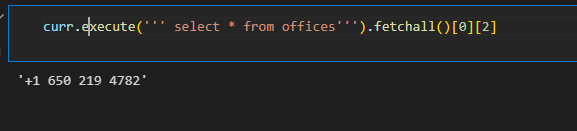
1. Select \* using cursor or can use ‘curr.execute(''' select \* from offices''').fetchall(),

curr.execute(''' select \* from offices ''')

curr.fetchall()



1. curr.execute(''' select \* from offices''').fetchall()[0][2]



1. Viewing column names(  
   Looks like we got some data, but it's not clear what each element represents. We can view the column names in the cursor's description attribute.)

curr.description



1. Create a dataframe with column names from the records/from a table

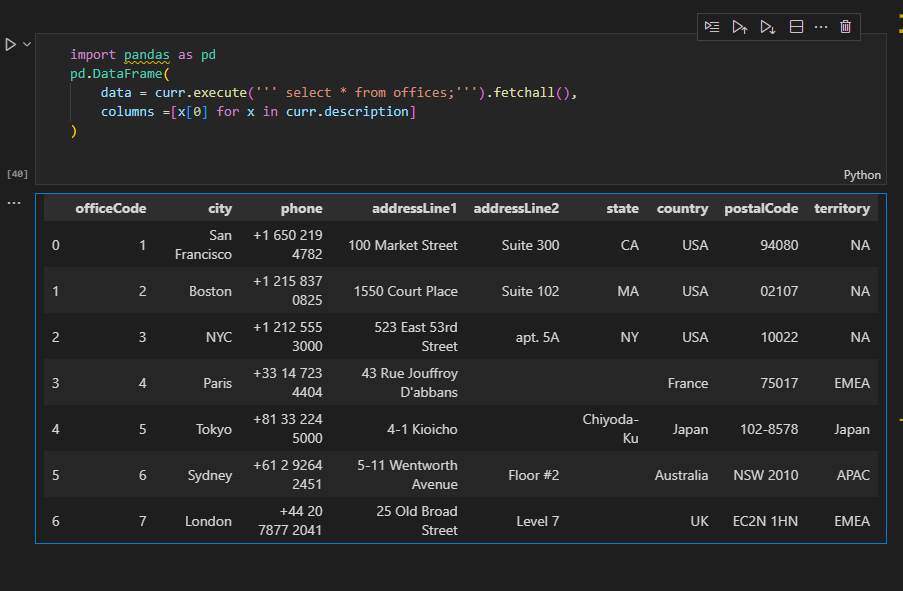
import pandas as pd

pd.DataFrame(

    data = curr.execute(''' select \* from offices;''').fetchall(),

    columns =[x[0] for x in curr.description]

)



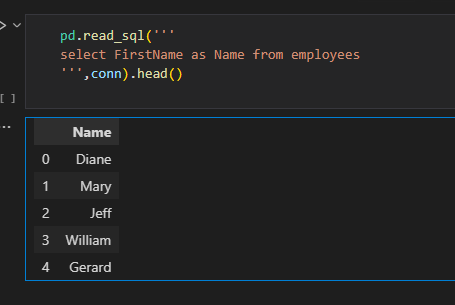
1. Retrieving a subset of columns

pd.read\_sql('''

      select lastName,FirstName from employees

            ''',conn).head()

1. Use Aliases (AS keyword) to change column names



1. Using SQL case statements

pd.read\_sql('''

select Firstname,LastName,jobTitle,

            CASE

            WHEN jobTitle = 'Sales Rep' then 'Sales Rep'

            ELSE 'Not Sales Rep'

            END AS role

From employees

''',conn)

1. Cases to make Human Readable

pd.read\_sql('''

select FirstName,lastName,officeCode,

            CASE officeCode

            WHEN '1' then 'San Francisco, CA'

            WHEN '2' then 'Boston, MA'

            WHEN '3' then 'New York, NY'

            WHEN '4' then 'Paris, France '

            WHEN  '5' then  'Tokyo, Japan'

            END as office

from employees

            ''',conn).head(10)

1. a**)Check length**

pd.read\_sql('''

select length(firstName)  as name\_length

    from employees

''',conn).head(5)

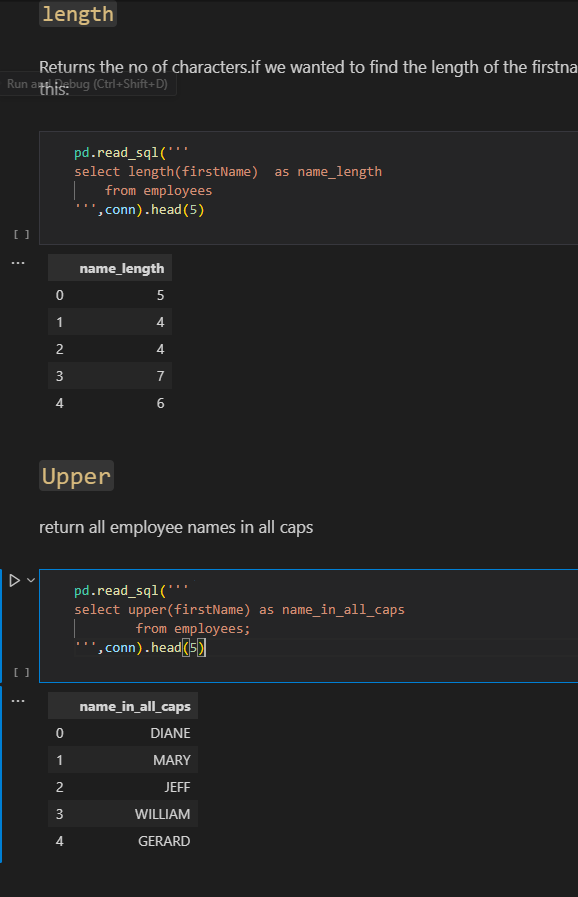
b)**Convert to upper**

pd.read\_sql('''

select upper(firstName) as name\_in\_all\_caps

        from employees;

''',conn).head(5)



**c)Slicing**

select substr(firstname,1,1) as first\_initial

from employees;

select substr(firstName,1,1) || '.' as first\_initial     --use || as **concatentate** operator

    from employees



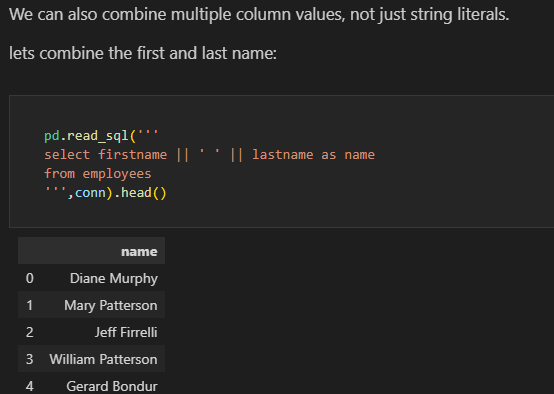
d) We can also **combine multiple column values**, not just string literals

pd.read\_sql('''

select firstname || ' ' || lastname as name

from employees

''',conn).head()



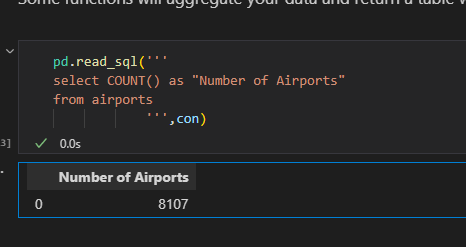
e)**Aggregations eg count**

pd.read\_sql('''

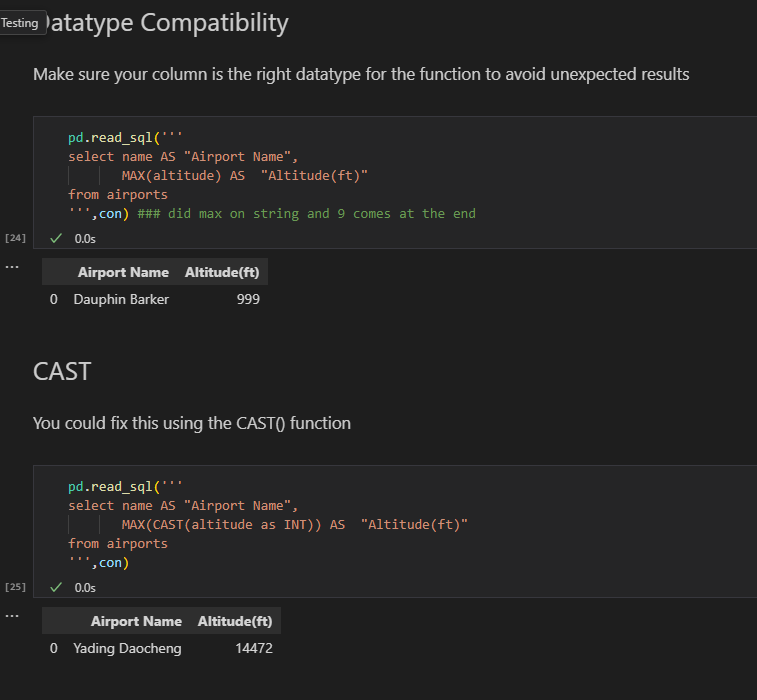
select COUNT() as "Number of Airports"

from airports

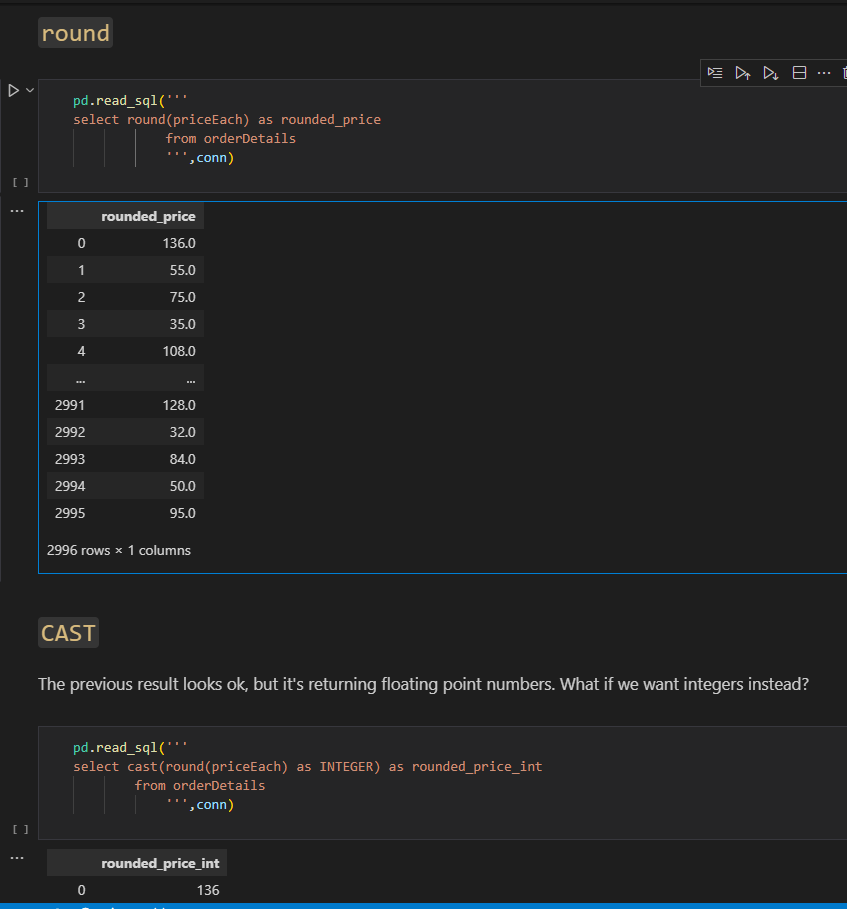
            ''',con)



NB: in AGGREGATIONS ensure that’s it’s the correct datatype for the function to avoid unexpected results eg

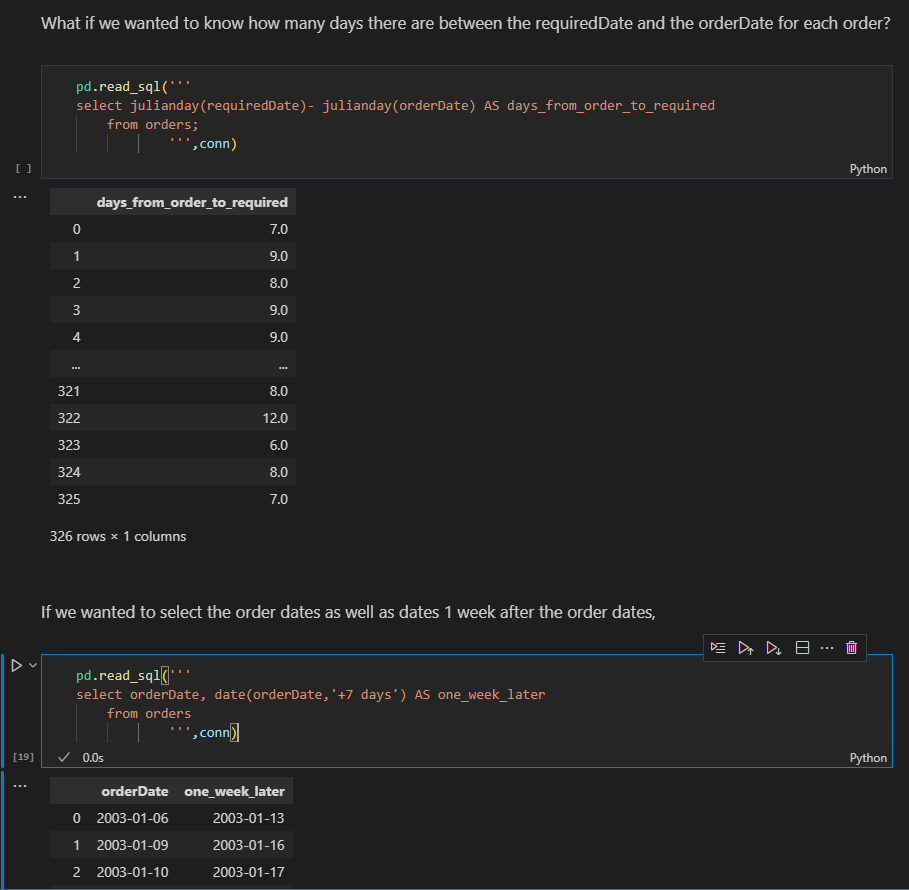


1. **Built in SQL for Math Functions**



1. **Built-in SQL Functions for Date and Time Operations-**

(JulianDay and Date)



1. strftime function

pd.read\_sql('''

select orderDate,

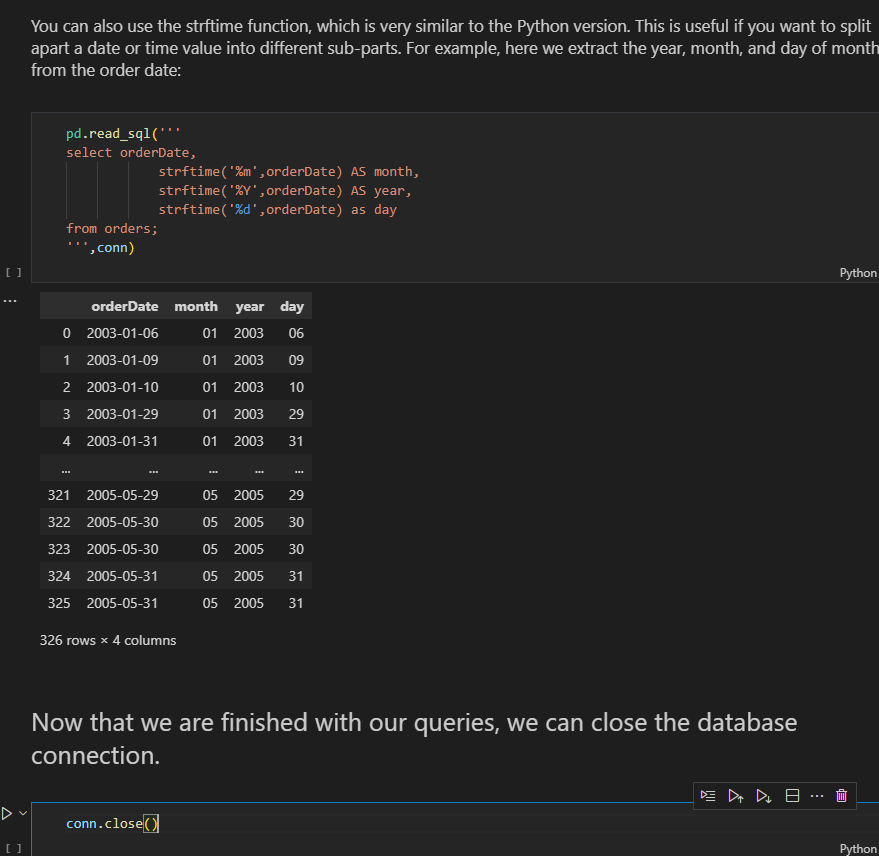
            strftime('%m',orderDate) AS month,

            strftime('%Y',orderDate) AS year,

            strftime('%d',orderDate) as day

from orders;

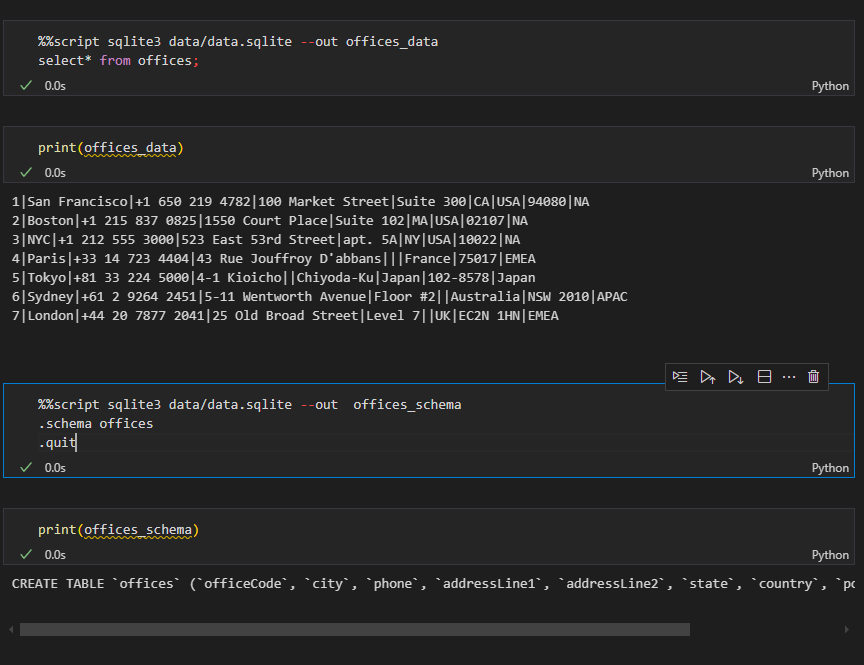
''',conn)



**13. If we really wanted to, we could just use those same SQLite terminal commands directly in a Jupyter Notebook using magic commands.**

%%script sqlite3 data/data.sqlite --out offices\_data

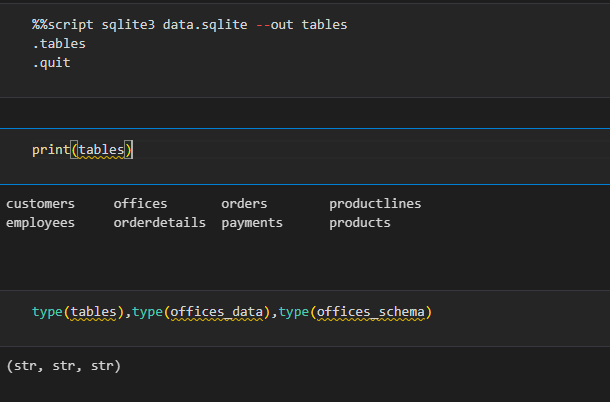
select\* from offices;



%%script sqlite3 data.sqlite --out tables

.tables

.quit



1. Connecting to the database using the terminal



1. **Relational databases** typically have multiple tables containing data, and the tables have defined relationships

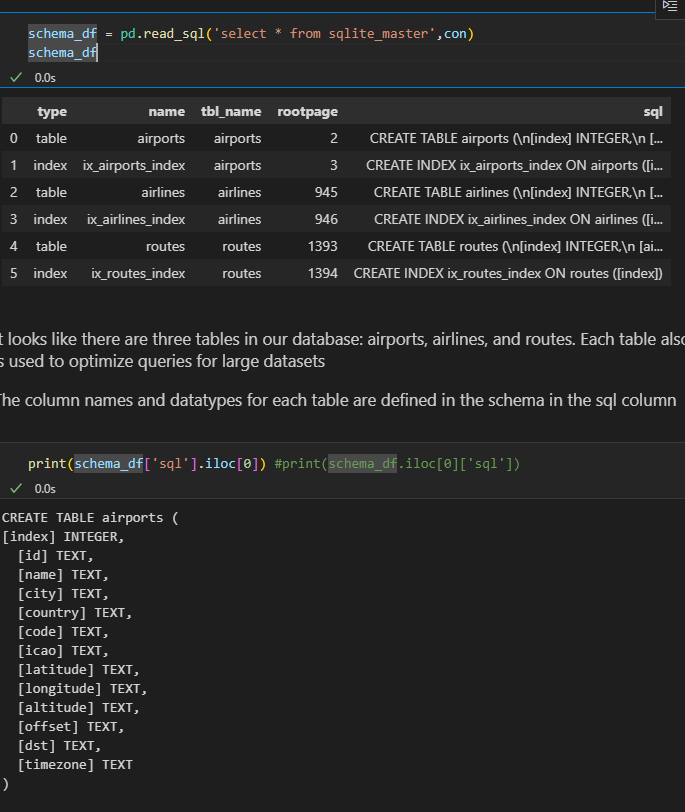
Database **Schema** – defines the structure of the database, including the tables and relationships between tables

**Primary key** – uniquely identifies each row in a table

**Foreign key** - used in one table to refer to the primary key of another table

1. Acess schema of lets say the first table

print(schema\_df['sql'].iloc[0]) #print(schema\_df.iloc[0]['sql'])



1. **Between**

pd.read\_sql('''

SELECT  name AS "Airport Name",

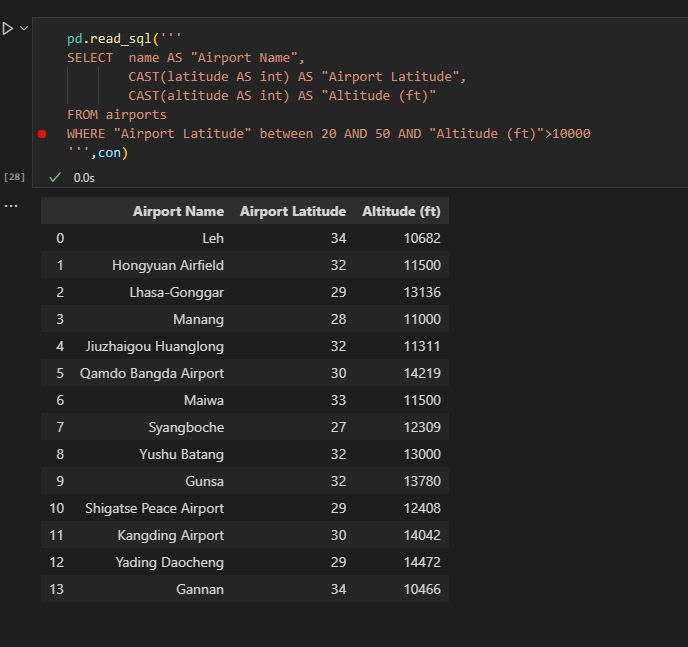
        CAST(latitude AS int) AS "Airport Latitude",

        CAST(altitude AS int) AS "Altitude (ft)"

FROM airports

WHERE "Airport Latitude" between 20 AND 50 AND "Altitude (ft)">10000

''',con)



1. **IS** – Useful when working with NULL values

pd.read\_sql('''

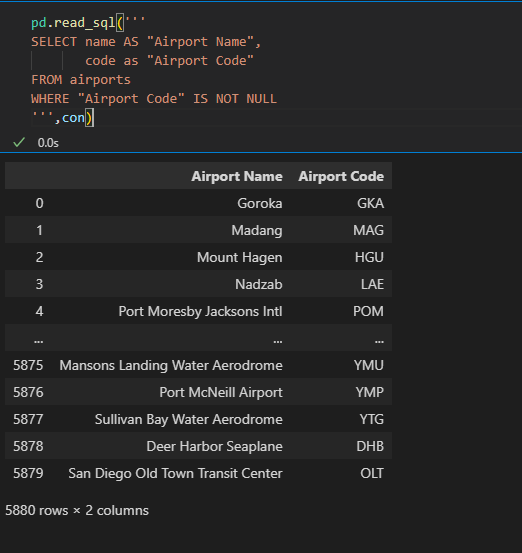
SELECT name AS "Airport Name",

       code as "Airport Code"

FROM airports

WHERE "Airport Code" IS NOT NULL

''',con)



1. **ORDER BY**

pd.read\_sql('''

SELECT name AS "Aiport Name",

        CAST(latitude AS int) AS "Airport Latitude",

        CAST(altitude AS int) AS "Altitude(ft)"

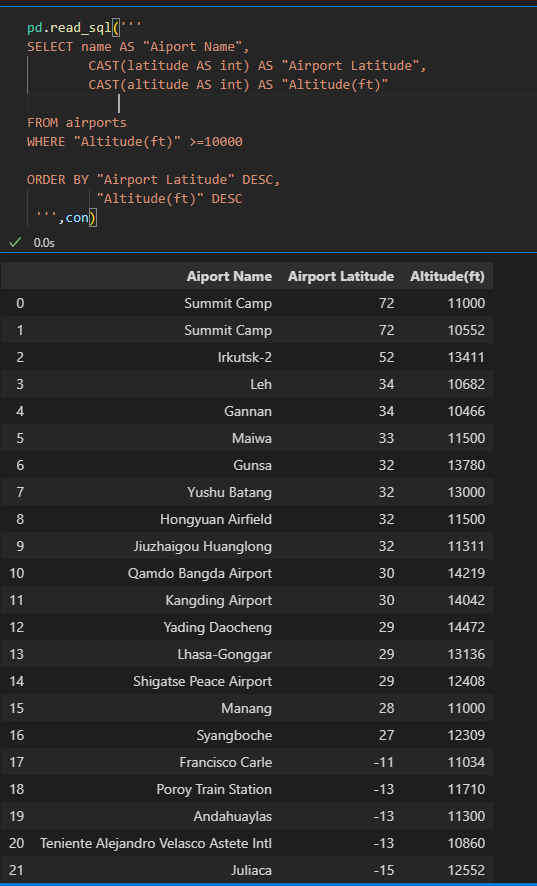
FROM airports

WHERE "Altitude(ft)" >=10000

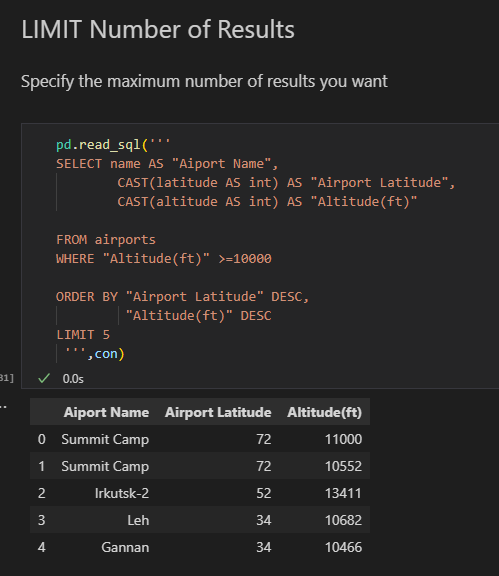
ORDER BY "Airport Latitude" DESC,

         "Altitude(ft)" DESC

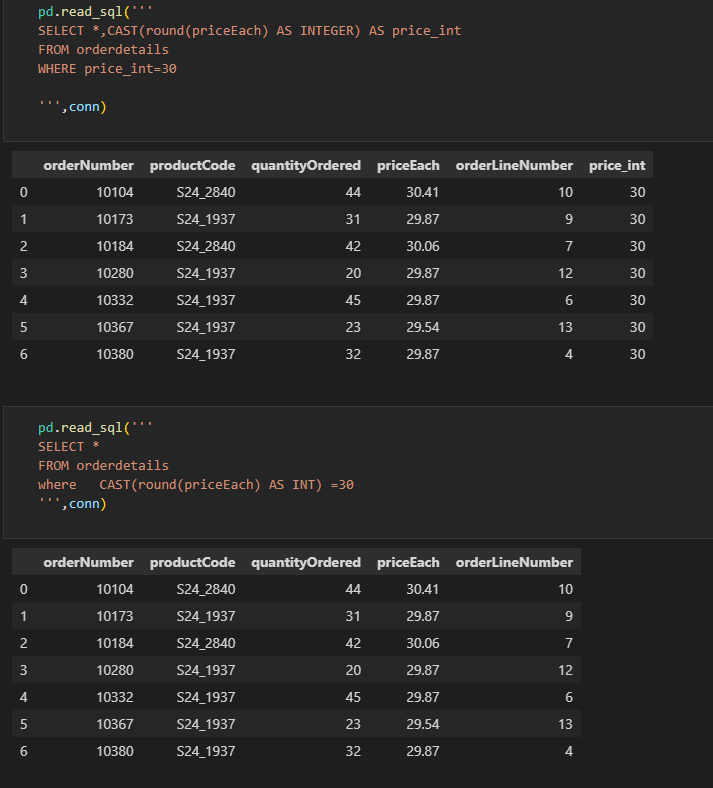
 ''',con)



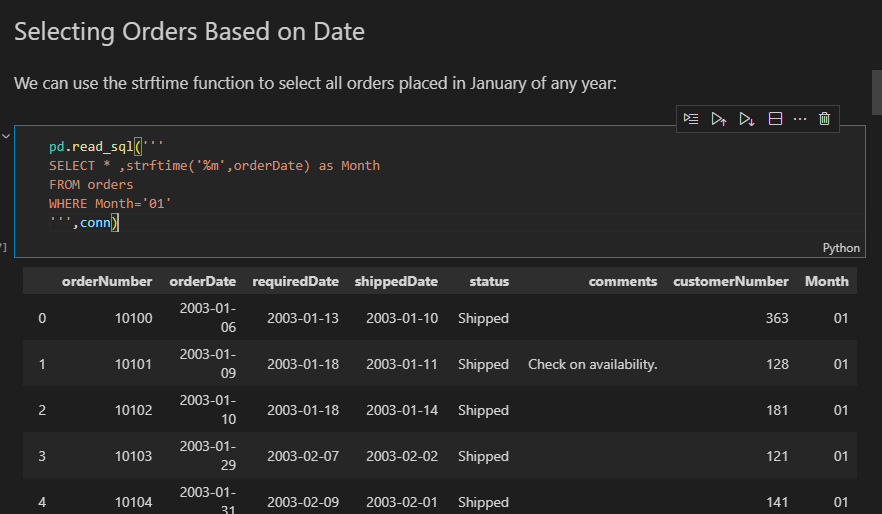
1. **LIMIT**



1. **a)Filtering by price**



b)**Filter by Date eg select all January orders**



**C)Filter late orders**

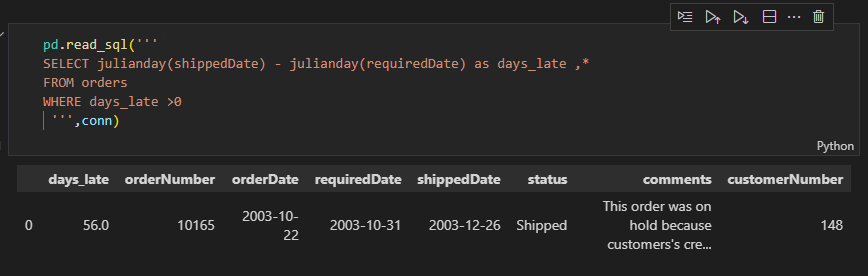
pd.read\_sql('''

SELECT julianday(shippedDate) - julianday(requiredDate) as days\_late ,\*

FROM orders

WHERE days\_late >0

 ''',conn)



**d)Like –** select cats starting with ‘M’ or ‘m’



**e) select all cats with four-letter names where the second letter was "a", we could use \_a\_\_**

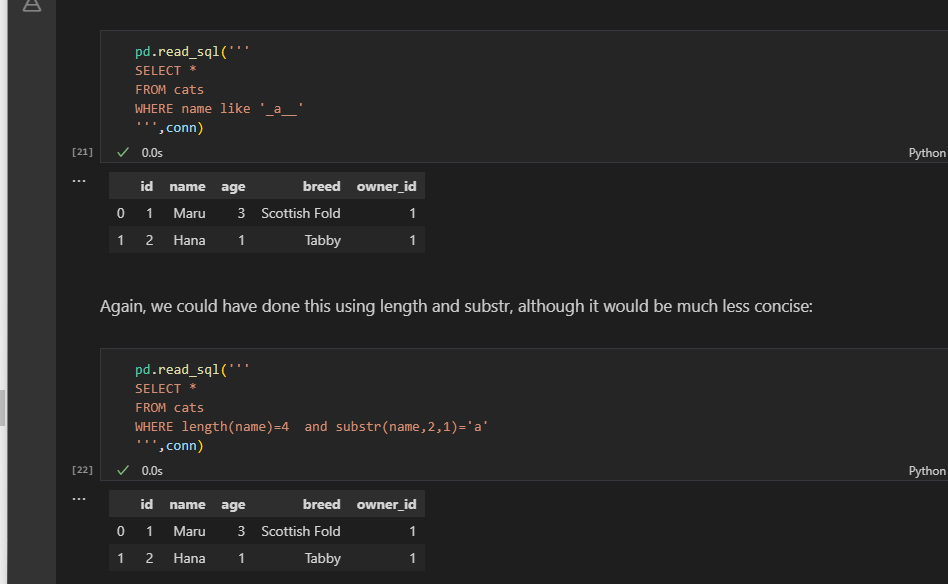
pd.read\_sql('''

SELECT \*

FROM cats

WHERE name like '\_a\_\_'

''',conn)



1. Select from 2 tables

pd.read\_sql('''

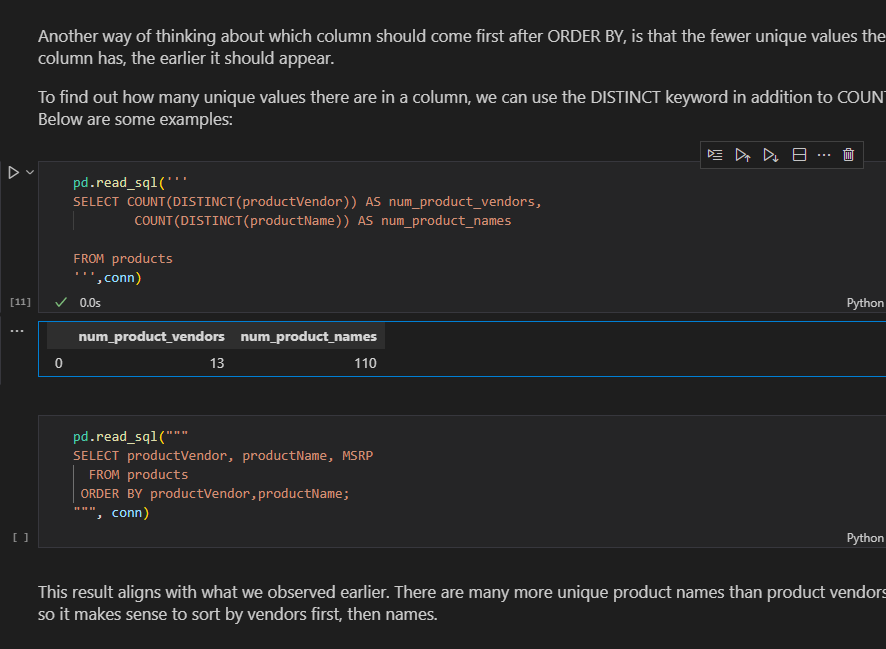
select cats.name,dogs.name

from cats,dogs;

''',conn)



1. way of thinking about which column should come first after ORDER BY, is that the fewer unique values the column has, the earlier it should appear



1. Create database and tables

**import sqlite3**

**conn = sqlite3.connect('pets\_database.db')**

**cur = conn.cursor()**

a)Create Cats table and insert records

#Creating the cats table

cur.execute('''

CREATE TABLE cats(

            id INTEGER PRIMARY KEY,

            name TEXT,

            age INTEGER,

            breed TEXT

            )

''')

============================================================= cur.execute('''

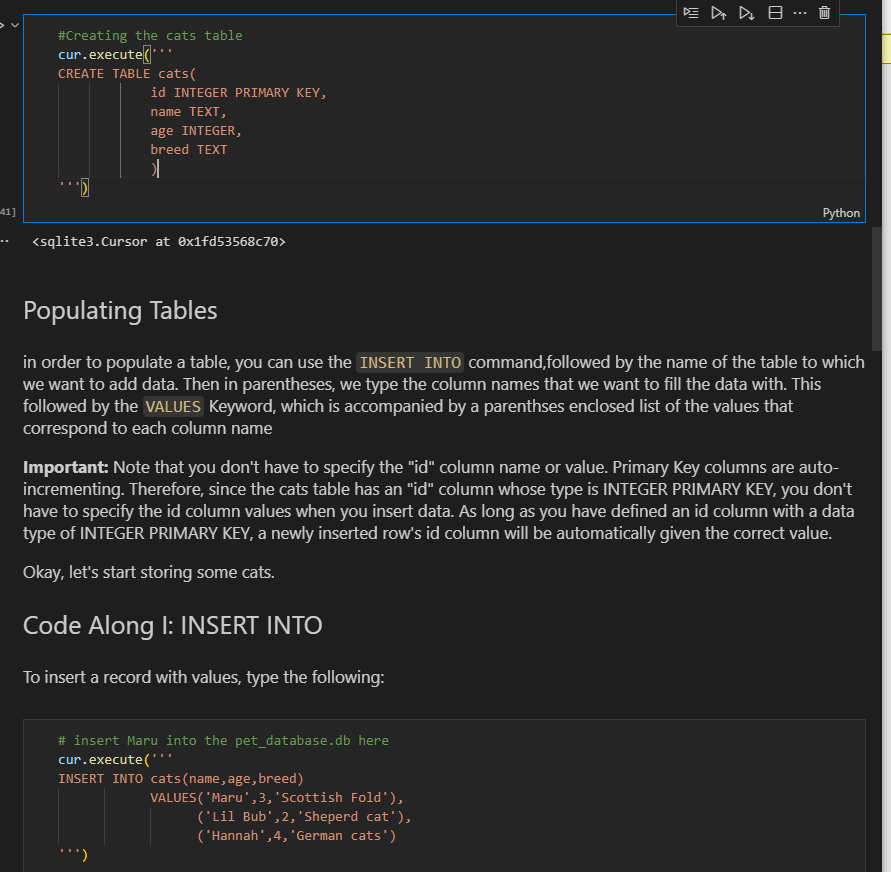
INSERT INTO cats(name,age,breed)

            VALUES('Maru',3,'Scottish Fold'),

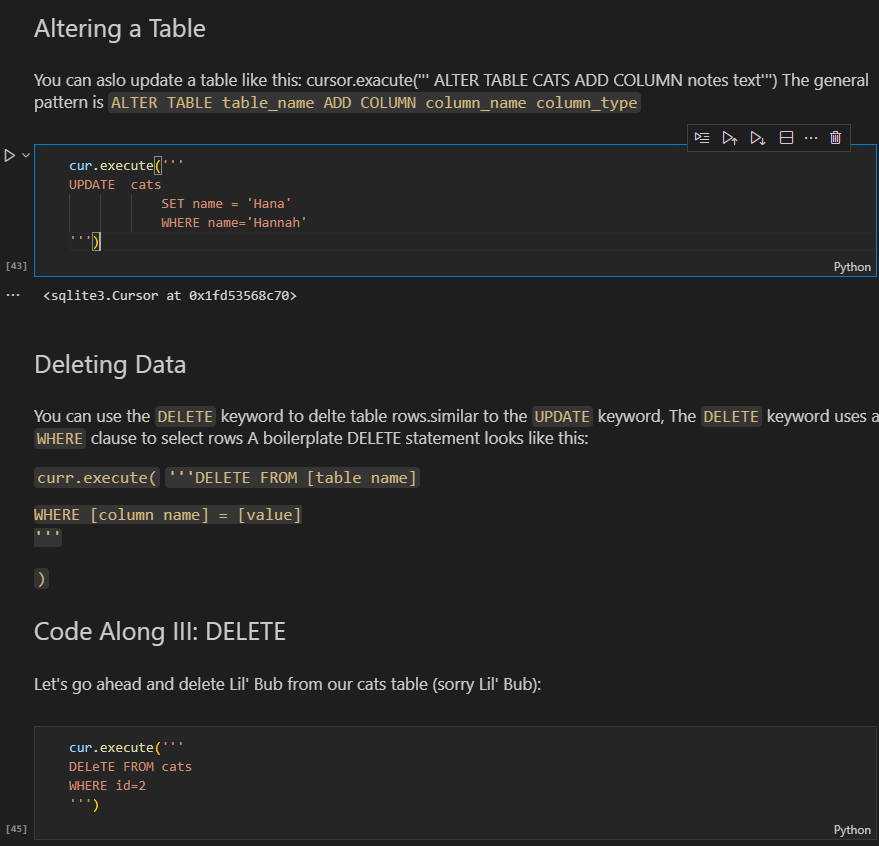
                  ('Lil Bub',2,'Sheperd cat'),

                  ('Hannah',4,'German cats')

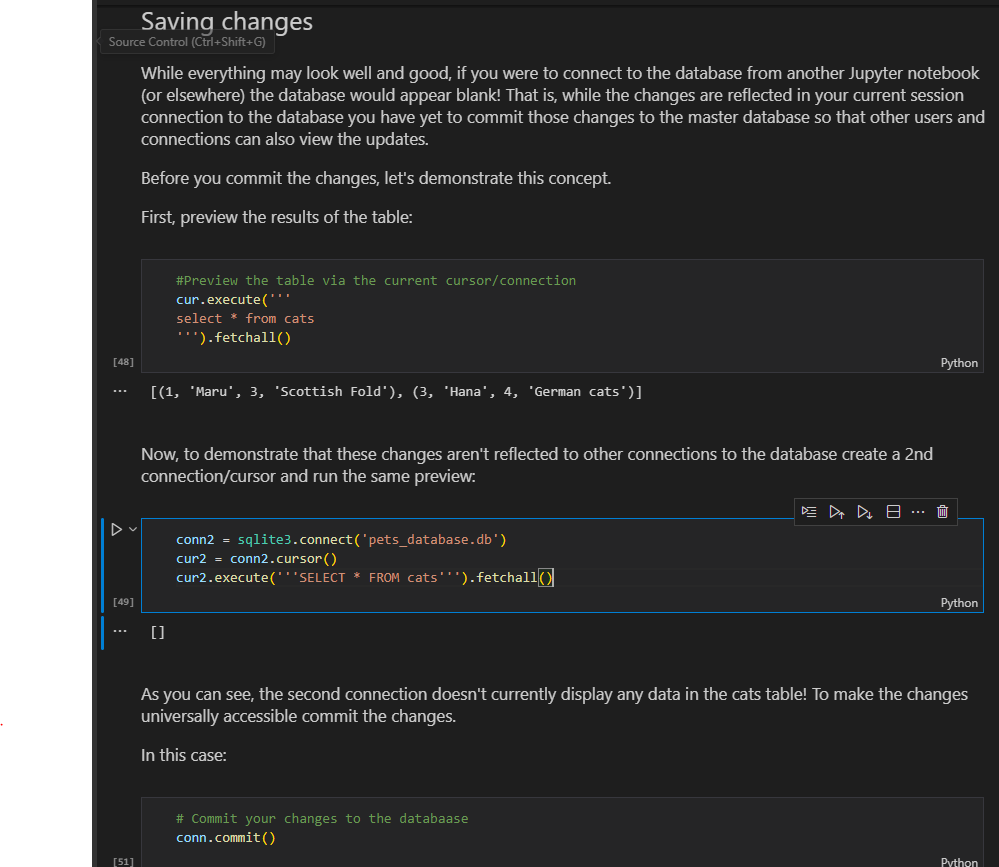
''')



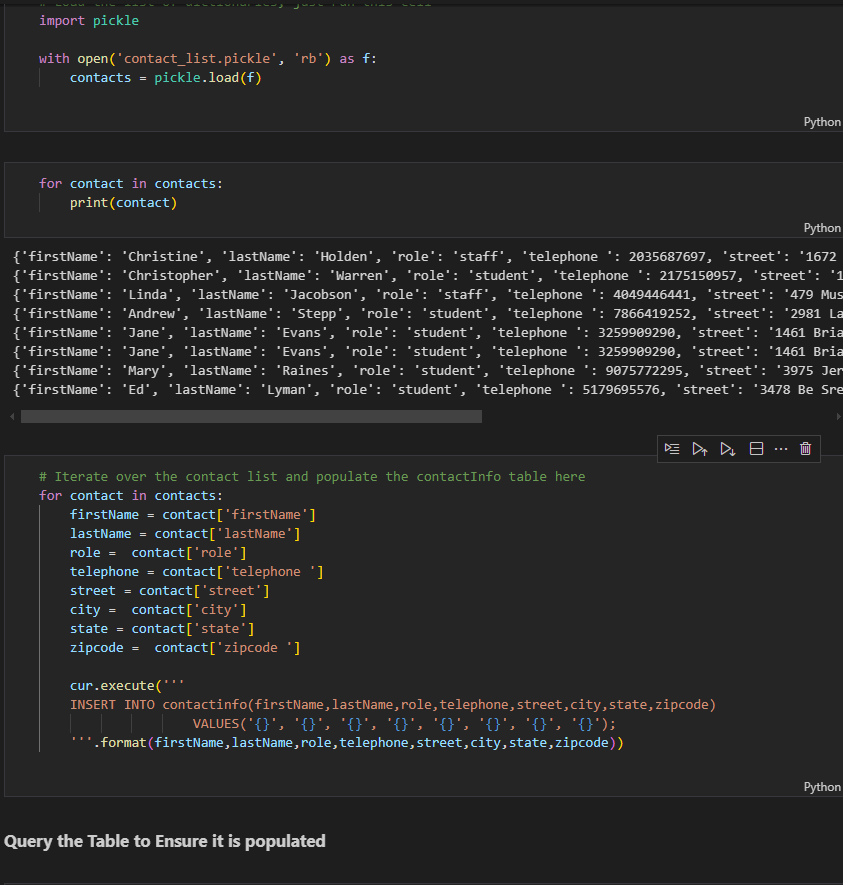
1. Altering(update) and deleting data



1. We need to save the changes otherwise when we create and new connection and acess record we find that its empty



1. Insert into table from dictionary



1. Create table with dual key

cur.execute('''

CREATE TABLE Grades(

            userid INTEGER NOT NULL,

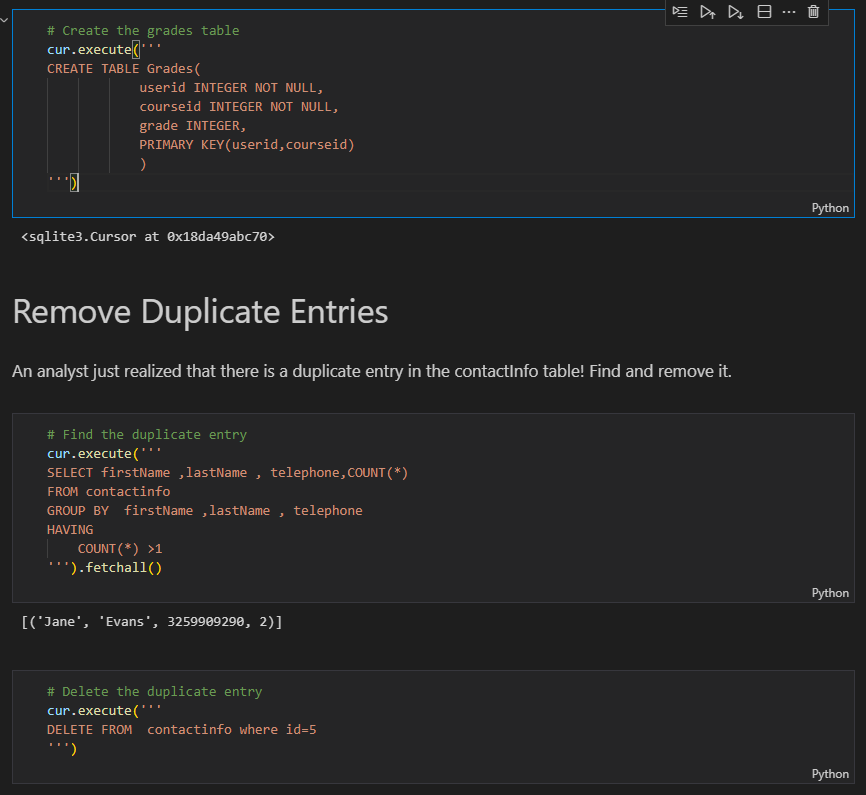
            courseid INTEGER NOT NULL,

            grade INTEGER,

            PRIMARY KEY(userid,courseid)

            )

''')



1. **Typing** – the practice of explicitly declaring a type.execises some level of control over our data.Without typig our data can become complicated and messy and it would be difficult to ask the database questions about large sets of data

* TEXT -str
* INTEGER- int
* REAL -float
* BLOB

1. **JOINS**

q= '''

select \*

FROM orderdetails

JOIN products

ON orderdetails.productCode = products.productCode

LIMIT 10

'''

pd.read\_sql(q,conn)

**b)Columns have the same name**

#IF COLUMNS HAVE THE SAME NAME

q= '''

select \*

FROM orderdetails

JOIN products

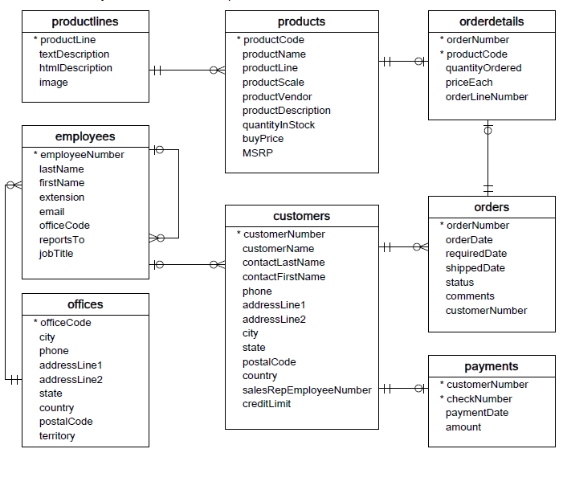
USING (productCode)

LIMIT 10

'''

pd.read\_sql(q,conn)

1. **JOIN IF table is related by joining another table** eg checking how many customers are there per office yet customer does relate to offices but related through employee

q = """

SELECT

    o.officeCode,

    o.city,

    COUNT(c.customerNumber) AS n\_customers

FROM offices AS o

JOIN employees AS e

    USING(officeCode)

JOIN customers AS c

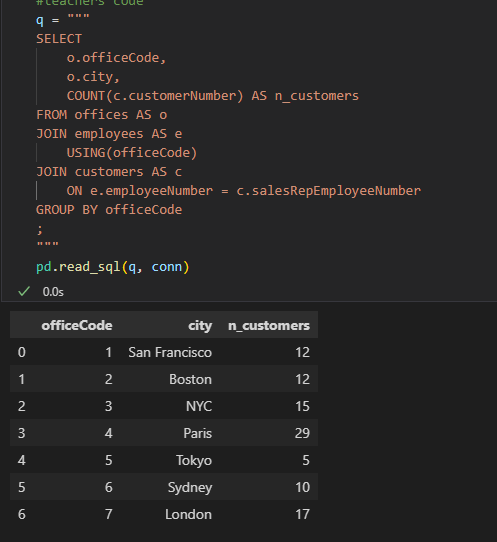
    ON e.employeeNumber = c.salesRepEmployeeNumber

GROUP BY officeCode

;

"""

pd.read\_sql(q, conn)



1. **Cast** for numeric purposes

q='''

SELECT

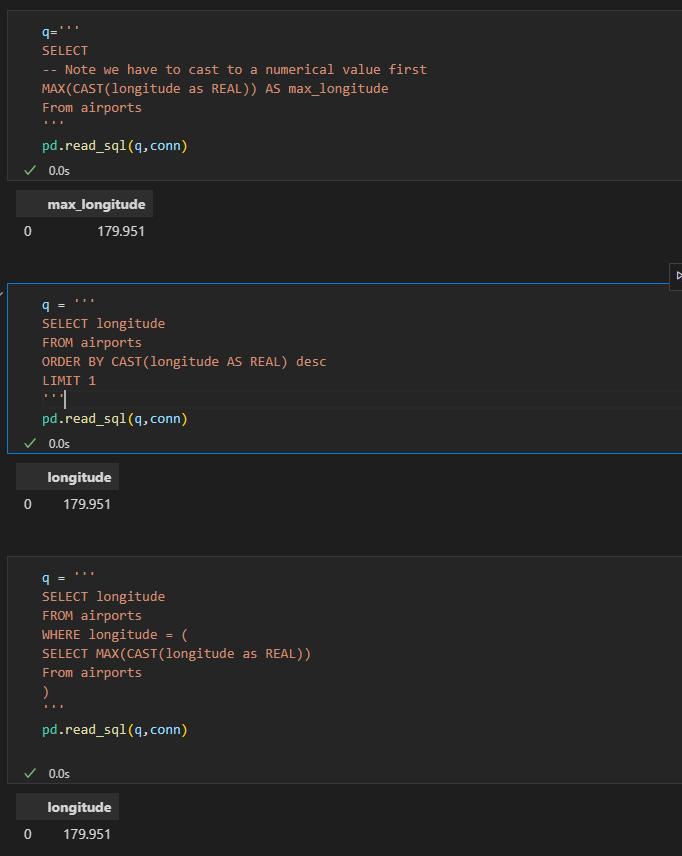
-- Note we have to cast to a numerical value first

MAX(CAST(longitude as REAL)) AS max\_longitude

From airports

'''

pd.read\_sql(q,conn)



1. Using eval to create new columns

df = df.eval('Age\_x\_Fare= Age \* Fare')

df.head()

1. Querying dataframes with sql(using pandassql)- easier to write than from dataframes

pip install pandasql

from pandasql import sqldf

pysqldf = lambda q: sqldf(q,globals())

q = '''

SELECT name

FROM df

LIMIT 10;

'''

passenger\_names = pysqldf(q)

passenger\_names



1. **Statistical Distribution** – Representation of the frequencies of potential events or the percentage of time each event occurs.

**a.Discrete Distribution**- Known number of possible outcomes(describes random variables that can take on a countable number of distinct values).use PMF

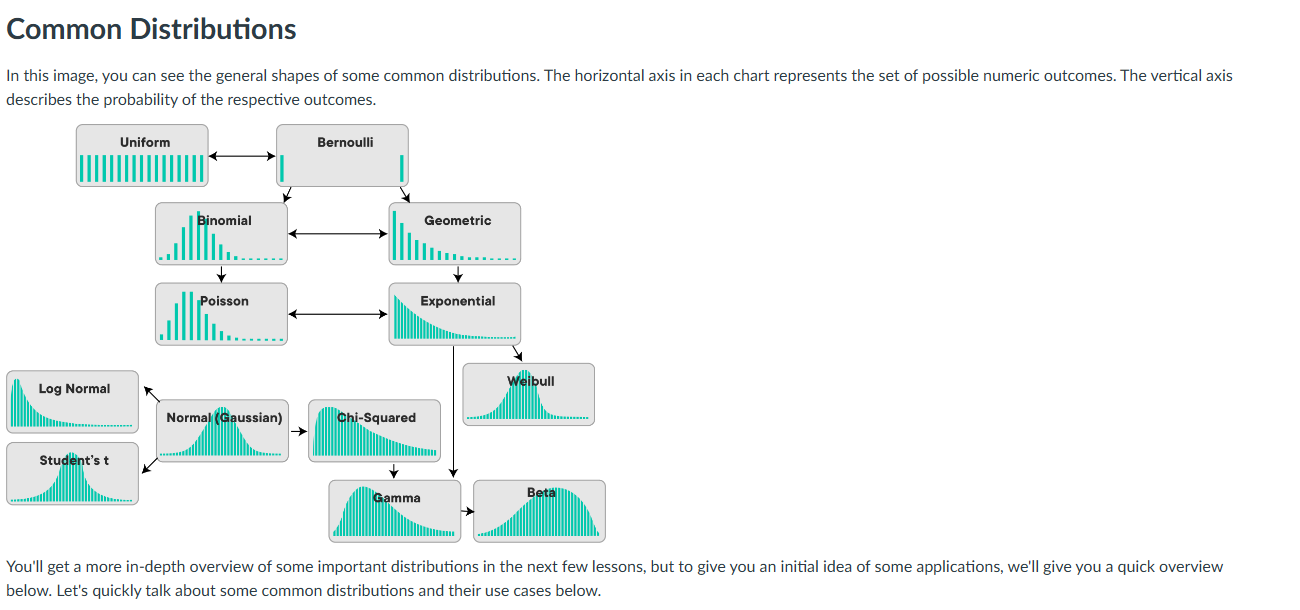
**Eg**

* Rolling a dice(possible outcomes 1,2,3,4,5,6)
* Number of defective items in a batch
* Count of customers arriving at the store in an hour

**b. Continous Distributions** – infinite no of possible values(describes random variables that can take on an infinite number of values within a given range)

**Eg**

* Heights of individuals
* Time taken to complete a task
* Temperature measurements



**c. Common distributions and their use cases**

**Examples of Discrete Distributions**

* **The Bernoulli Distribution –** deals with a series of Boolean events eg coin toss
* **Poisson Distribution –** rep the likelihood of a given no of successes over a given period of time. A typical example is pieces of mail. If your overall mail received is constant, the number of items received on a single day (or month) follows a Poisson distribution. Other examples might include visitors to a website, or customers arriving at a store, or clients waiting to be served in a queue.
* **Uniform distribution-** Occurs when all possible outcomes are equally likely eg dice
* **Binomial distribution –** Probability of observing a specific no of successes (Bernoulli trials) in a specific no of trials
* **Geometric**

**Examples of Continuous Distributions**

* **Normal/Gaussian Distribution –** follows a bell shape and is a foundational distribution for many models and theories un statistics and data science. A normal distribution turns up very often when dealing with real-world data including heights, weights of different people, errors in some measurement or grades on a test. Our temperature example above follows a normal distribution as well!
* **Exponential –**  rep the amt of time it takes before an event occurs
* **Continous uniform**

1. **Population** -Whole set of possible outcomes(universal set)
2. **Sample** – subset of the population

**Reasons we cant observe a whole group/population**

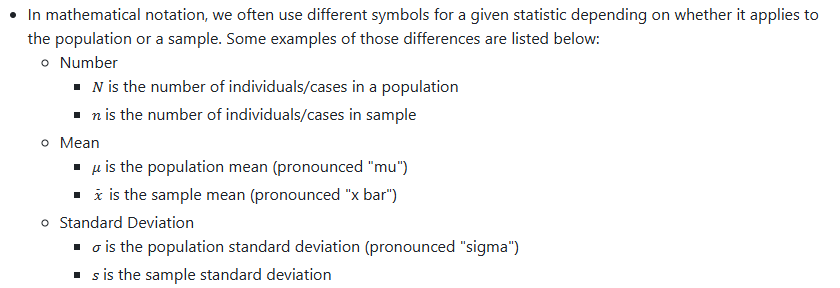
* Expensive(not enough fuding)
* Unrealistic (grp too large)
* We don’t need a whole population to gain insights
* People not willing to give info
* A large enough sample has mean close to the population mean,small samples can be misleading
* Sample means with multiple samples has mean close to the population mean

1. **Descriptive statistics** – summary metrics that describe and summarize the main features of a dataset. They help in understanding the basic aspects of the data by providing quantitative descriptions eg meausures of central tendency(mean,mode, median),measures of dispersion/spread eg(variance. Std,IQR) and measures of shape(skweness and kurtosis)

-We use visualizations such as histograms and boxpplots to understand the shape of the distributions

**Population statisctics**- when descriptive statistics are applied to populations

**Point estimates**- descriptive statisctics applied to samples



1. **Probability Mass Function(PMF) –** Used for discrete random variables.It gives the probability that a discrete random variable is exacly equal to a specific value.

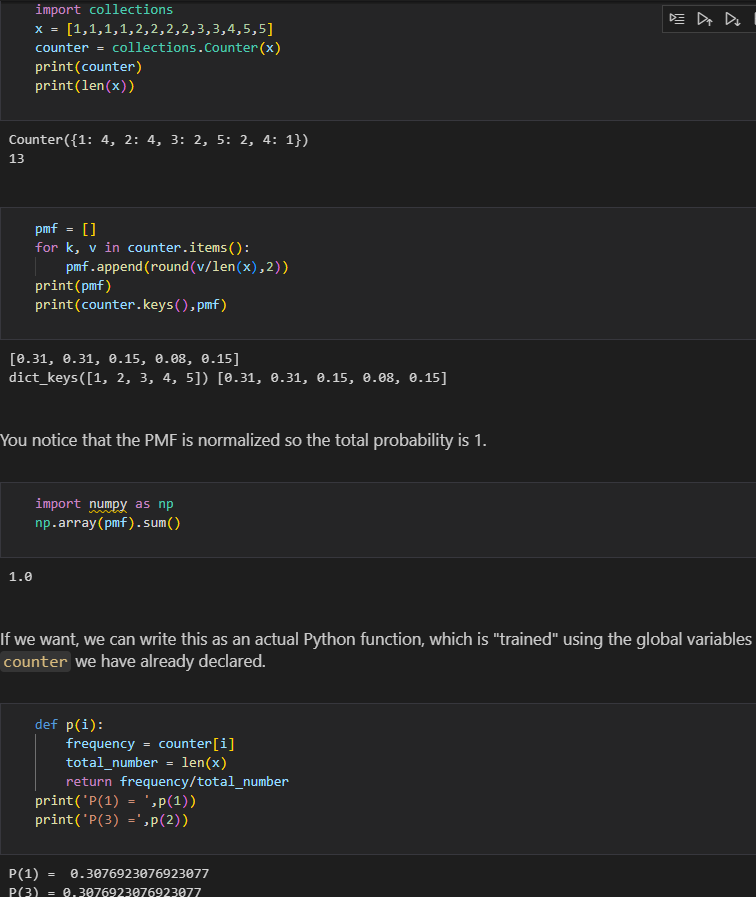
**PMF Intuition -** Probability is a number in the range [0,1] that is calculated as the frequency expressed as fraction of the sample size

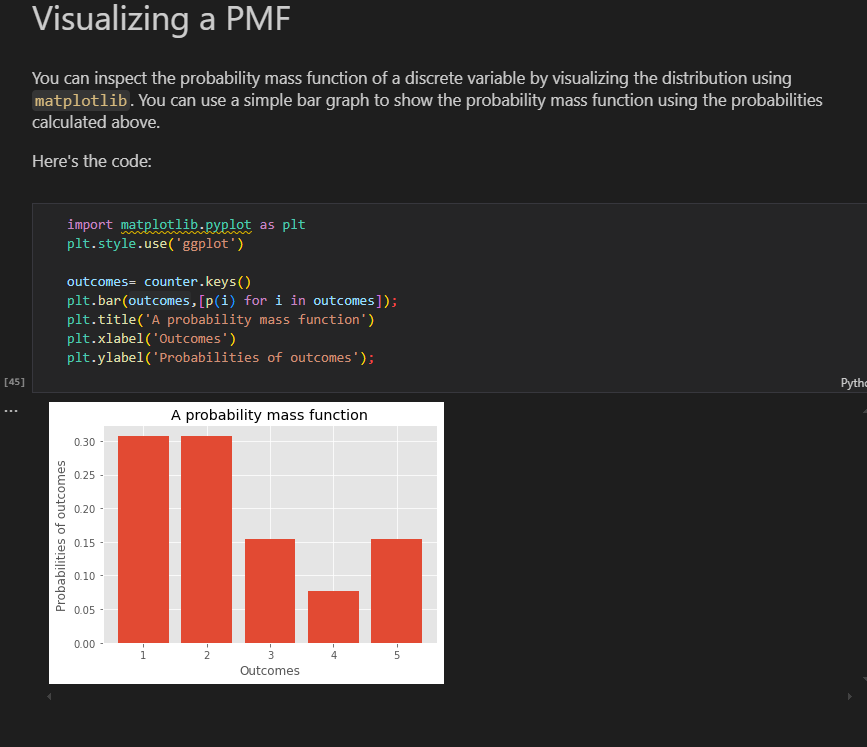
**-**In order to convert any random variables frequency into a probability we need to perform the following steps

* Get the frequency of each possible value in te dataset
* Divide the frequency of ach value by the total number of values(length of the dataset
* Get the probability of each value

**EXAMPLE1**

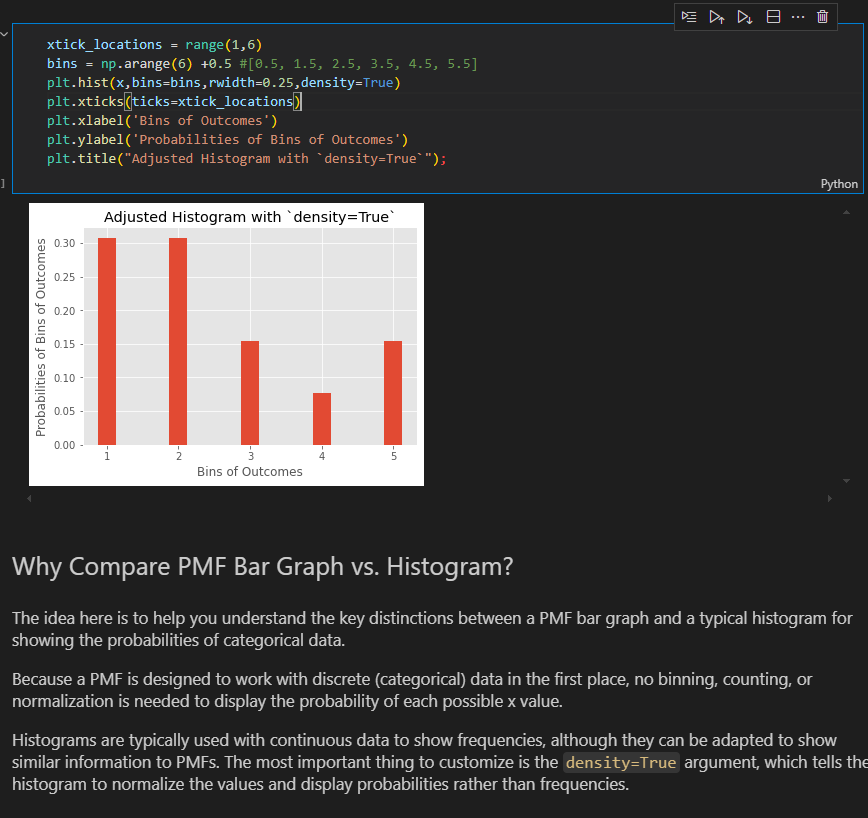
**Simple toy example**





**-**Can use histogram as well but include desnsity =true

plt.hist(x,bins=bins,rwidth=0.25,density=True)



**EXAMPLE 2(Titanic dataset)**



1. **Probability Density Function(PDF)** – used for continuous random variables. It describes the likelihood of the variable falling within a particular range of values , rather than taking on a specific value

**PDF Intution**-

**1. Concept of Density**

**\*\*Continuous Variables\*\***: Unlike discrete variables, continuous variables can take on an infinite number of values within a given range (e.g., heights, weights). Therefore, we can't assign a probability to a specific value (e.g., the probability of someone being exactly 170 cm tall is zero).

    Density: Instead, the PDF represents the density of probabilities. Higher values of the PDF at a point indicate a higher likelihood of the random variable being near that value.

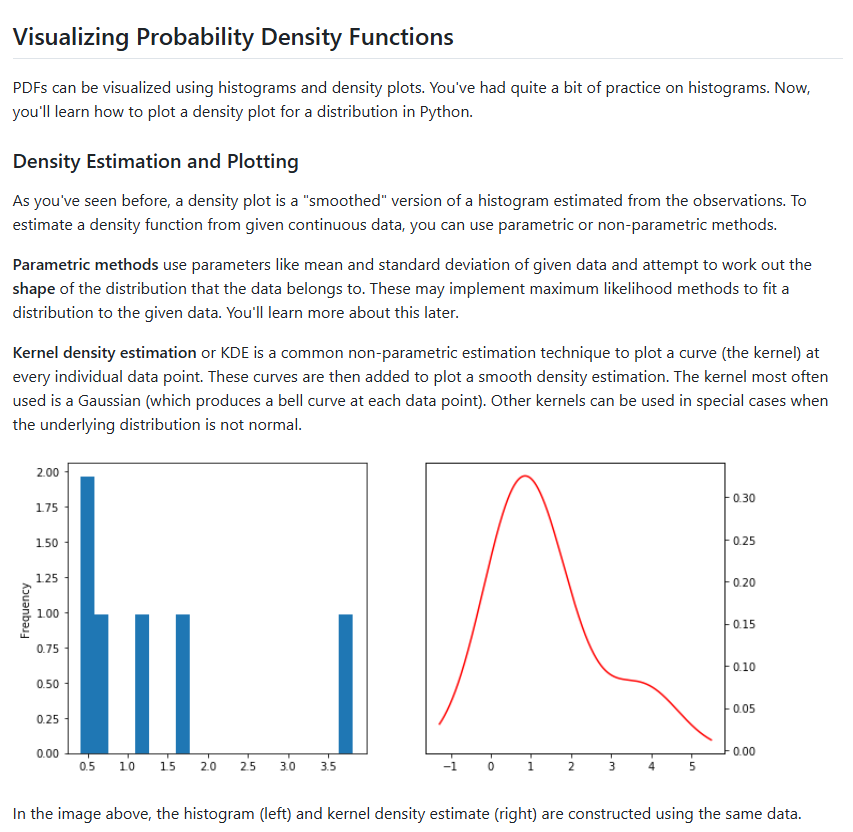
**2. Area Under the Curve**

**\*\*Probability as Area:\*\*** The PDF itself does not give probabilities directly; instead, probabilities are found by calculating the area under the curve of the PDF over a specific interval. For instance, the probability that a random variable X falls between a and b is given by:

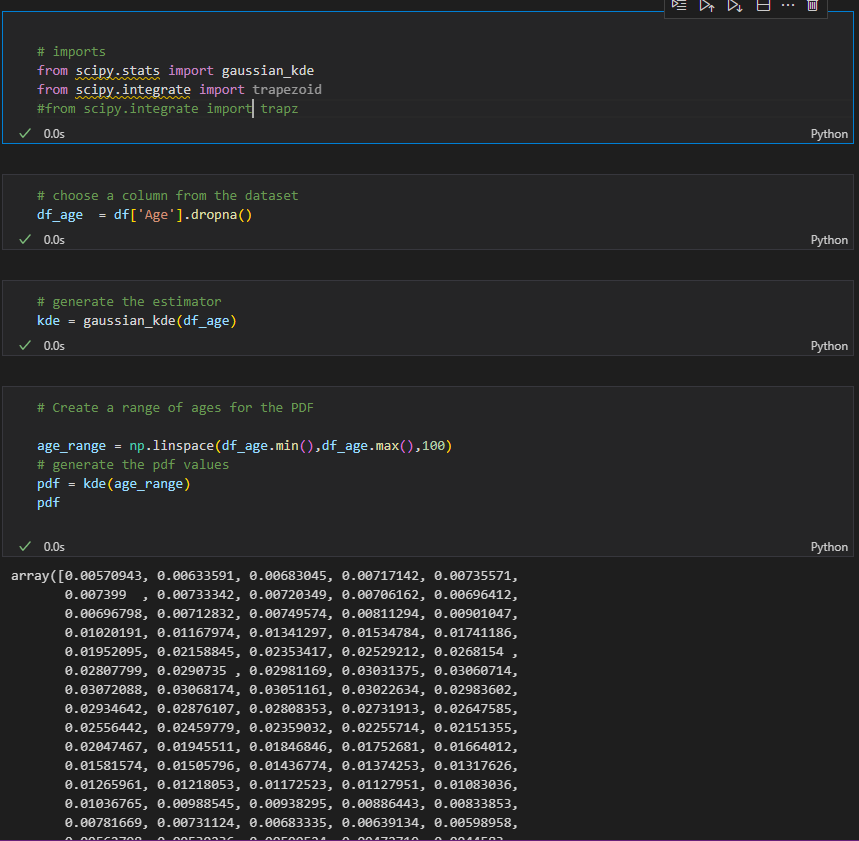
    P(a<X<b)=∫ab​f(x)dx

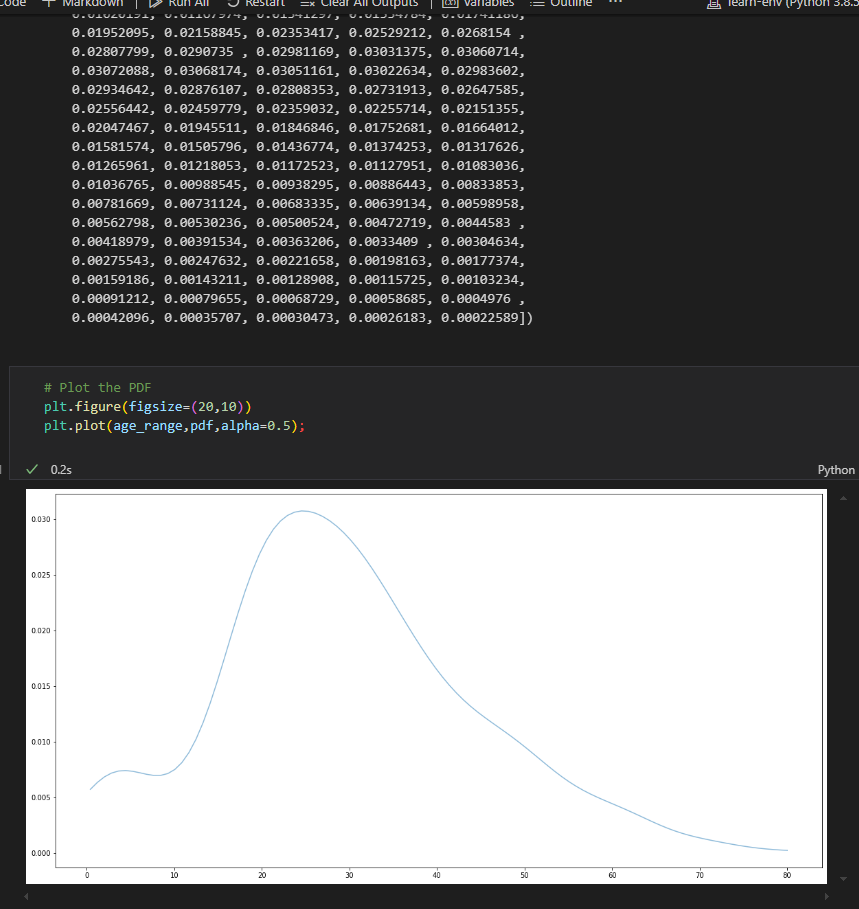
**3. Total Area Equals One**

**\*\*Normalization\*\***: The total area under the PDF curve is always equal to 1. This reflects the fact that the random variable must take on some value within the range of possible values.



**EXAMPLE:TITANIC DATA SET**





1. **CUMULATIVE DISTRIBUTION FUNCTION (CDF)-** provides a way to describe the probability distribution of a random variable

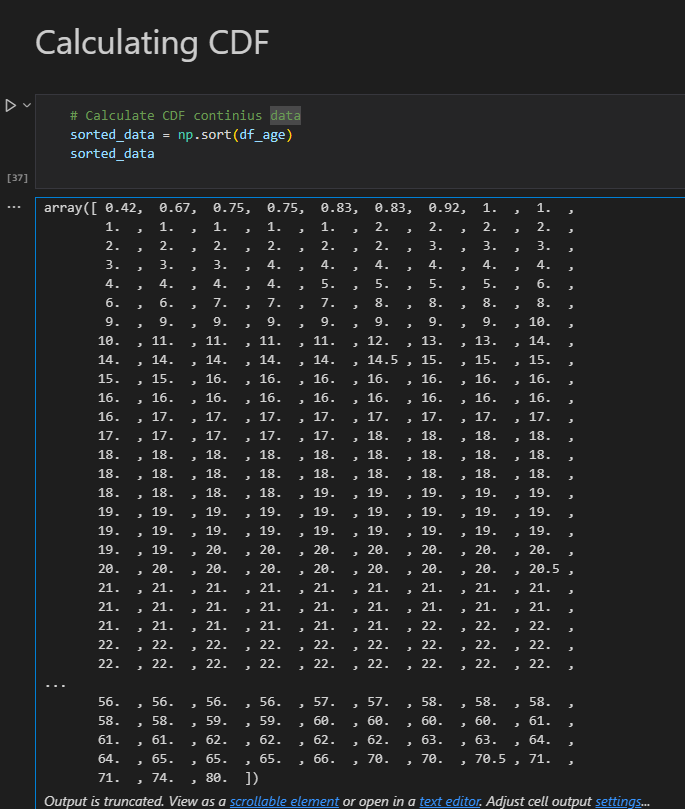
Gives the probability that variable X is less than or equal to a certain possible value of x

**Properties of CDF**

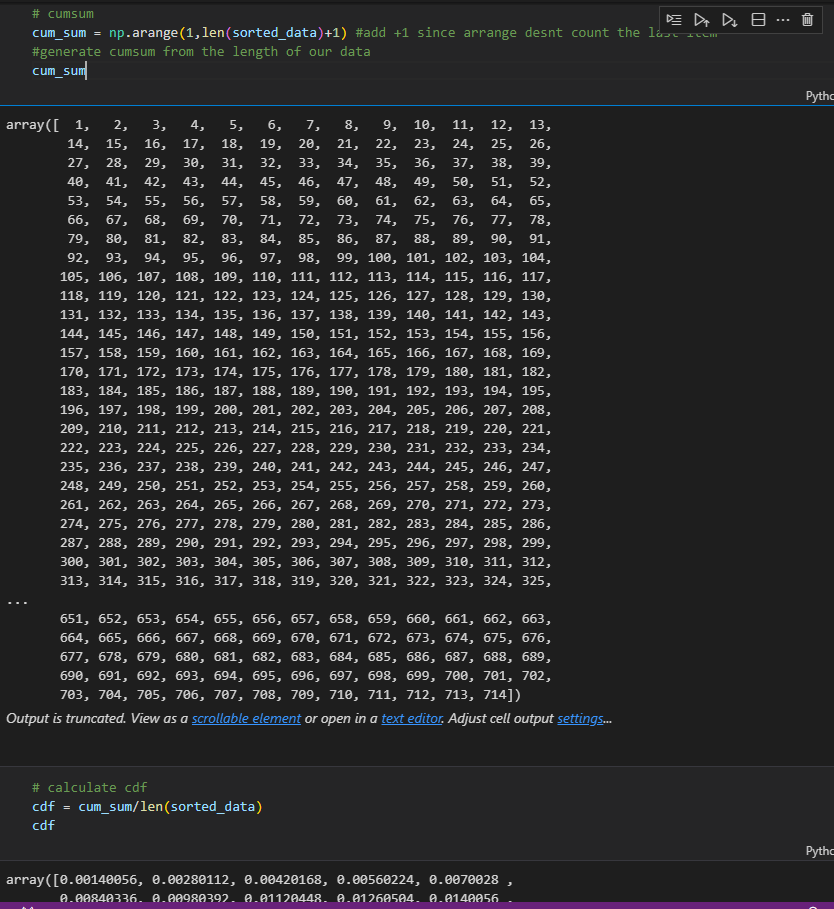
* Non-decreasing – CDF is always non-decreasing;as x increases,f(x) either increases or stays the same
* Range- CDF takes values in the range [0,1]
* Right-continous- CDF approaches its limit from the right

**EXAMPLE 1: CONT from titanic dataset**

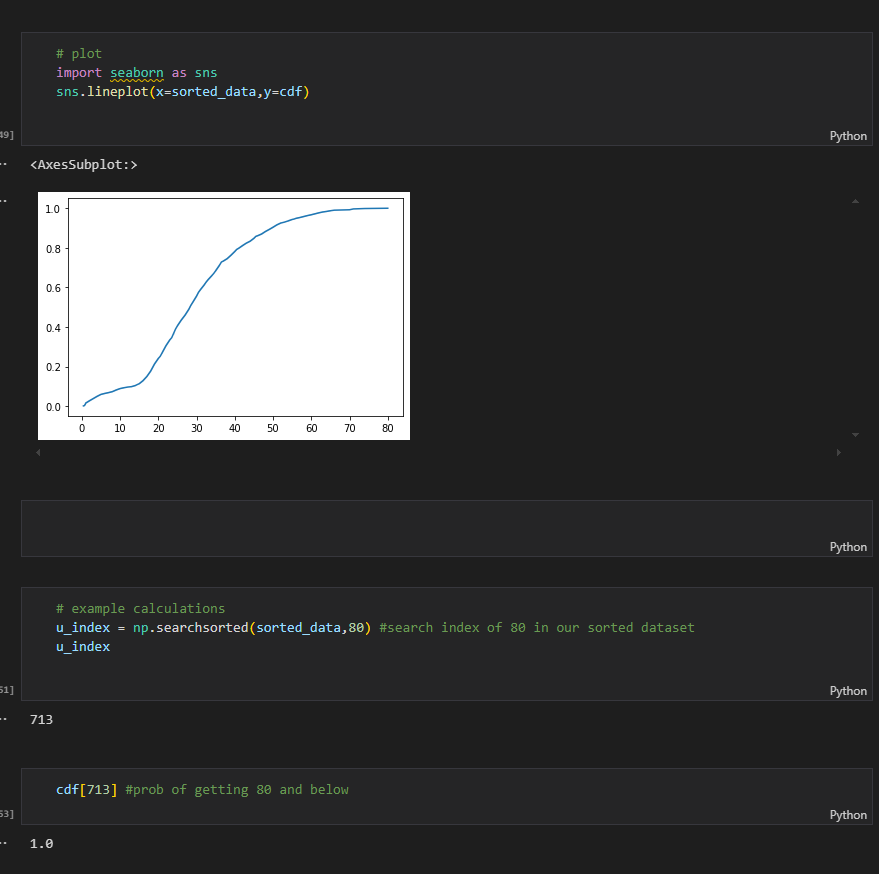
1. sort data

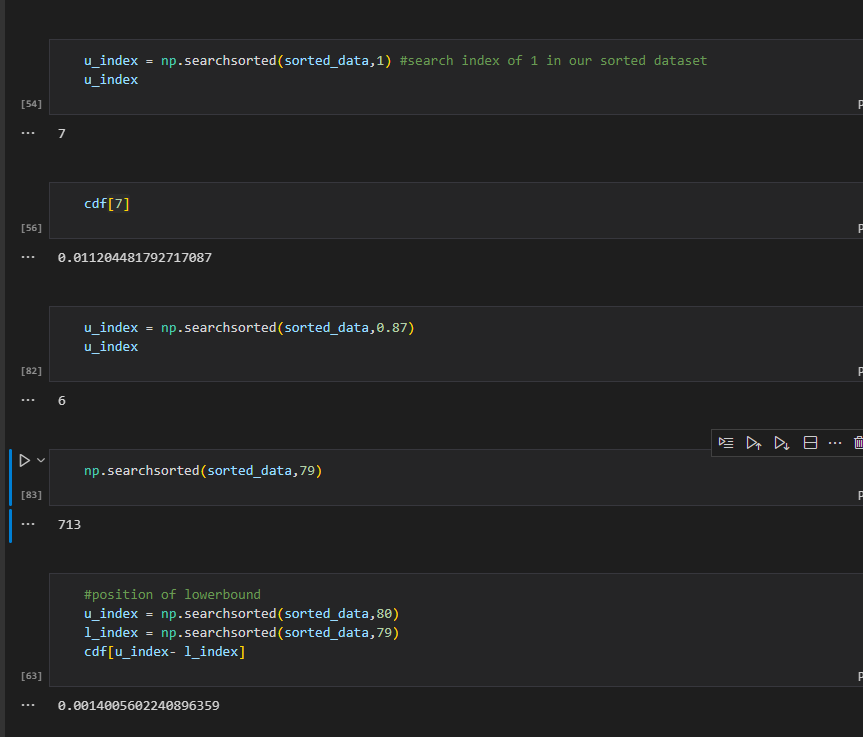


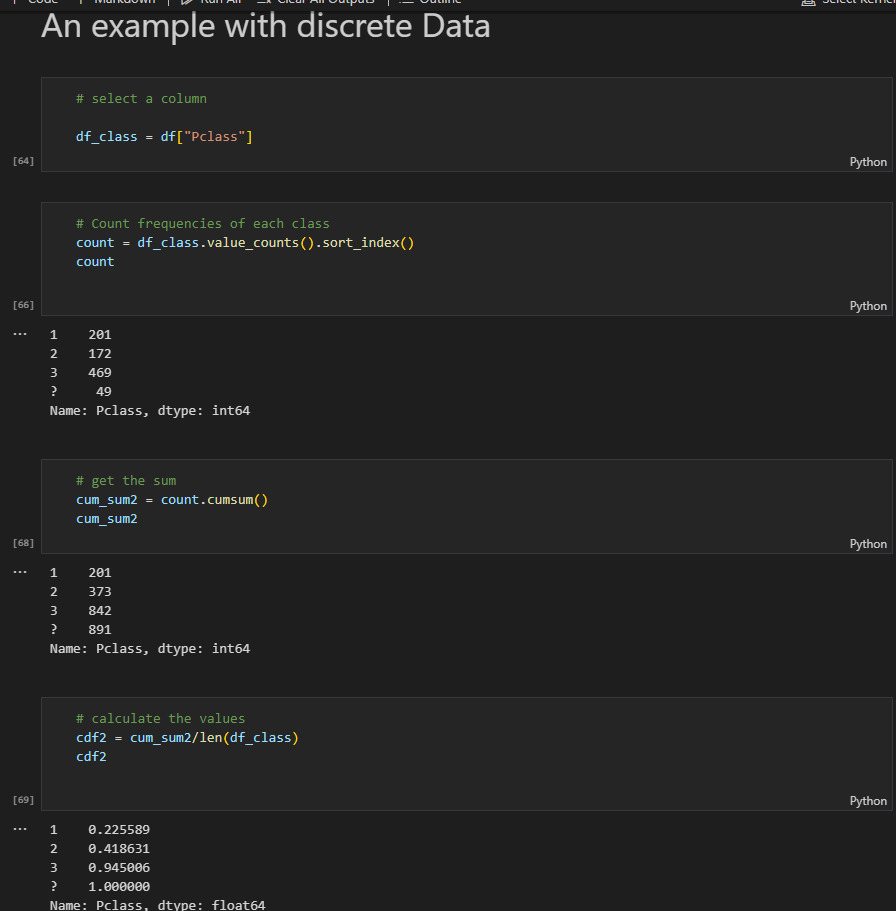
1. Do a cumulative sum and calculate cdf



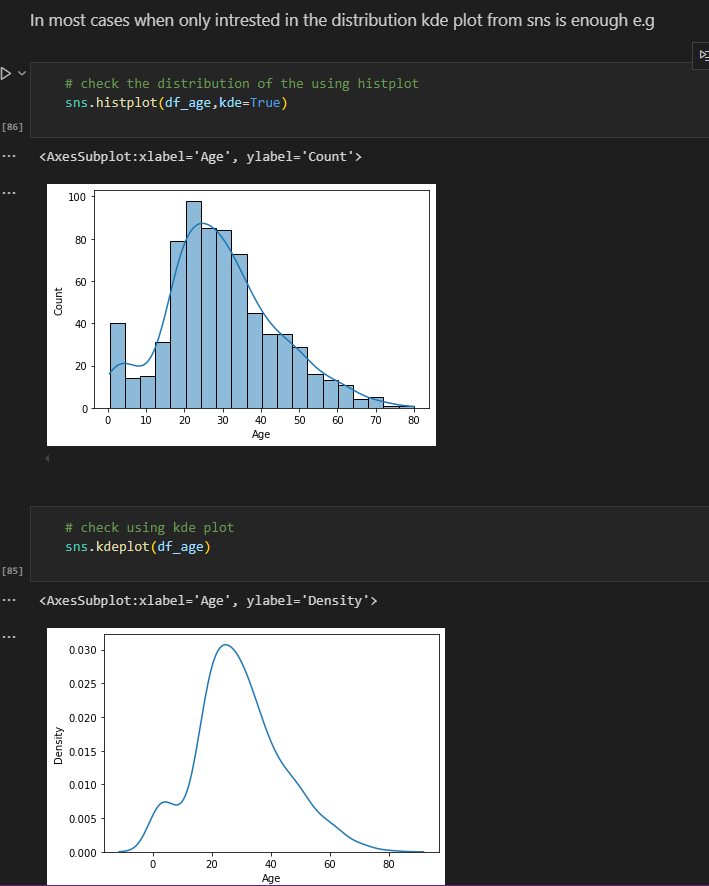
3.Plot

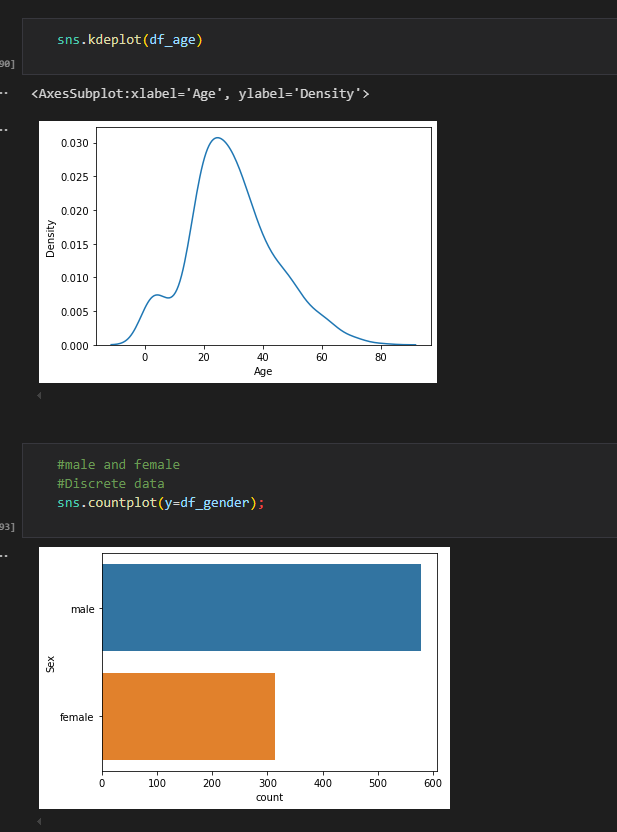


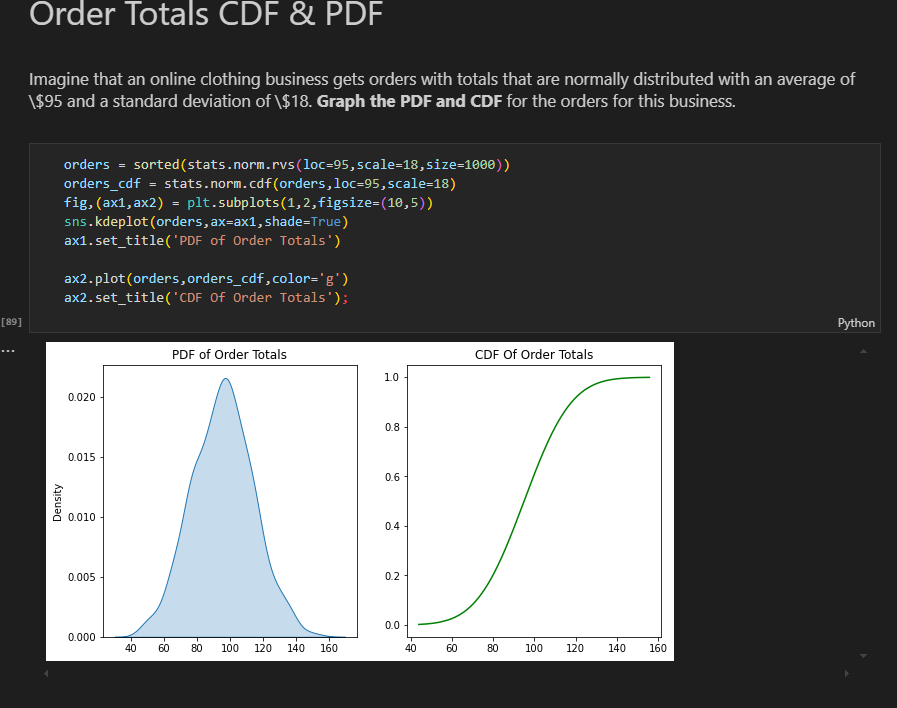






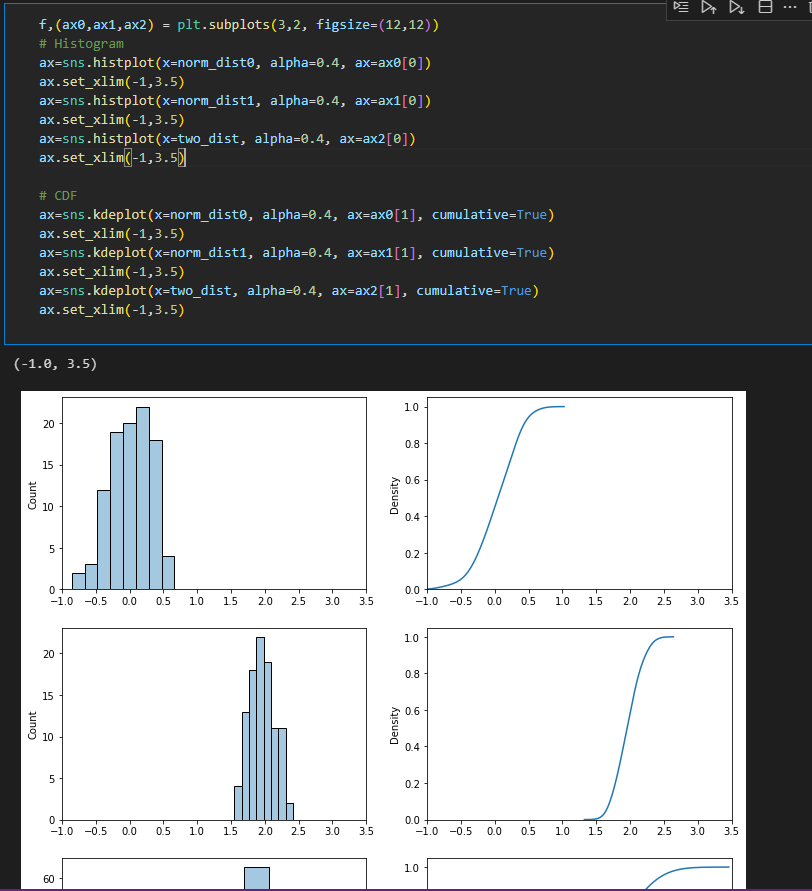








HISTOGRAM AND CDF

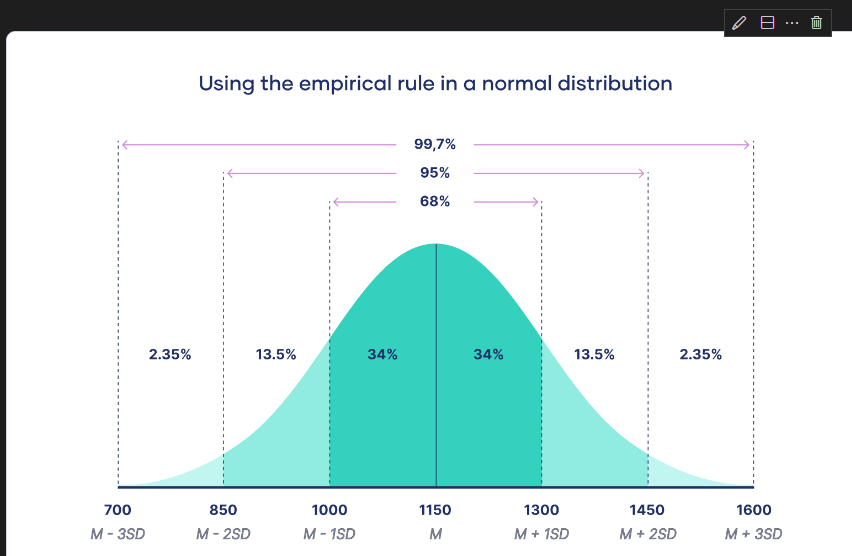


1. **Normal Distributions/Gaussian Distribution** - is defined by two parameters: the mean (μ) and the standard deviation (σ). The mean determines the center of the distribution, while the standard deviation measures the dispersion or spread of the data points around the mean. The probability density function

* The curve is symmetric about the mean, meaning that approximately 68% of the data lies within one standard deviation (σ) from the mean, about 95% within two standard deviations, and about 99.7% within three standard deviations. This is often referred to as the empirical rule or the 68-95-99.7 rule.

**Importance in Statistics**

* The normal distribution holds a central place in statistics for several reasons:
* Central Limit Theorem: One of the most significant reasons for the prominence of the normal distribution is the Central Limit Theorem (CLT). The CLT states that the distribution of the sample means will tend to be normally distributed, regardless of the original distribution of the data, as long as the sample size is sufficiently large. This makes the normal distribution a key tool for inferential statistics.
* Statistical Inference: Many statistical tests and methods, including t-tests, ANOVA, and regression analysis, assume that the underlying data is normally distributed. This assumption allows for the application of parametric tests, which can provide more powerful results than non-parametric alternatives.
* Natural Phenomena: Numerous real-world measurements—such as heights, weights, test scores, and measurement errors—tend to follow a normal distribution. This natural occurrence in various fields, including psychology, biology, and economics, makes the normal distribution a useful model for analyzing data



**X-Stics of Normal Distribution**

**1. Symmetry**- Bell-Shaped Curve: The normal distribution is symmetric about the mean (μ). This means that the left side of the curve is a mirror image of the right side. Mean = Median = Mode: In a normal distribution, the mean, median, and mode are all located at the center of the distribution.

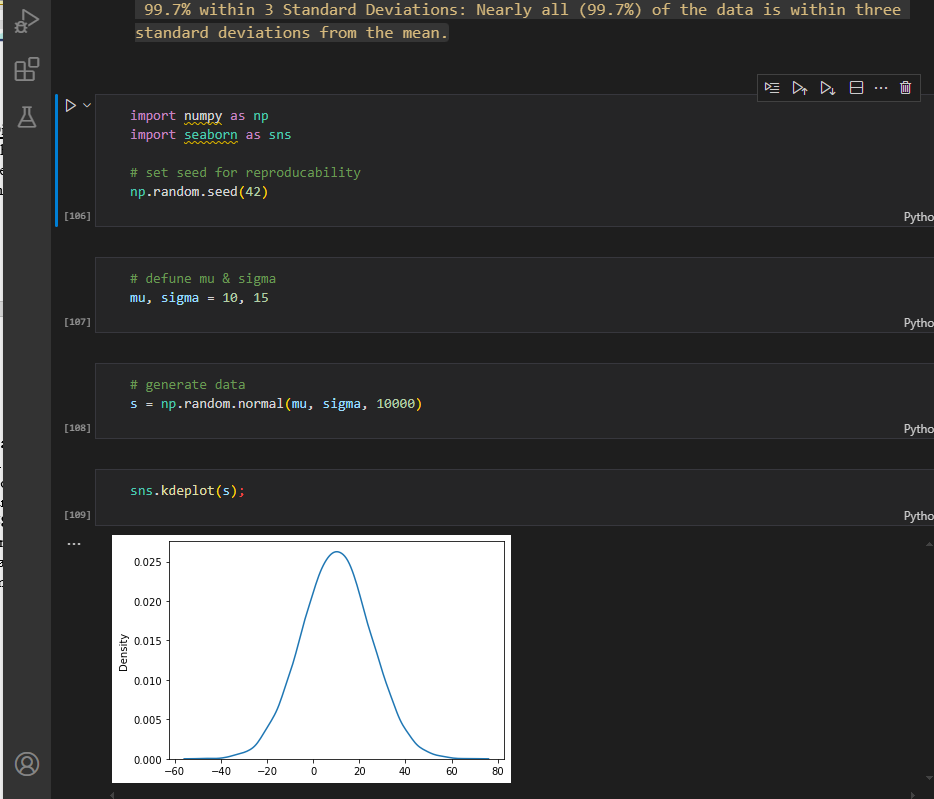
**2. Defined by Mean and Standard Deviation-** Mean (μ): The average value around which the data points are distributed. It determines the center of the distribution.

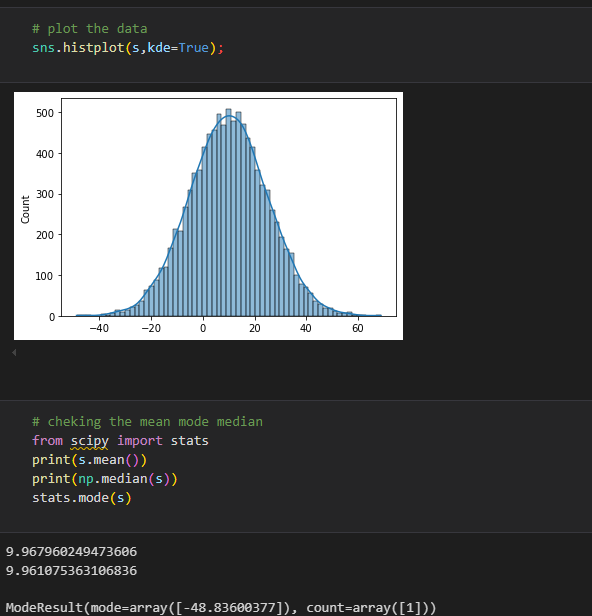
 Standard Deviation (σ): This measures the spread or dispersion of the data. A smaller standard deviation results in a steeper curve, while a larger standard deviation produces a flatter curve.

**3. Empirical Rule (68-95-99.7 Rule)-**  68% of Data within 1 Standard Deviation: Approximately 68% of the data falls within one standard deviation (σ) of the mean (μ).

  95% within 2 Standard Deviations: About 95% of the data lies within two standard deviations.

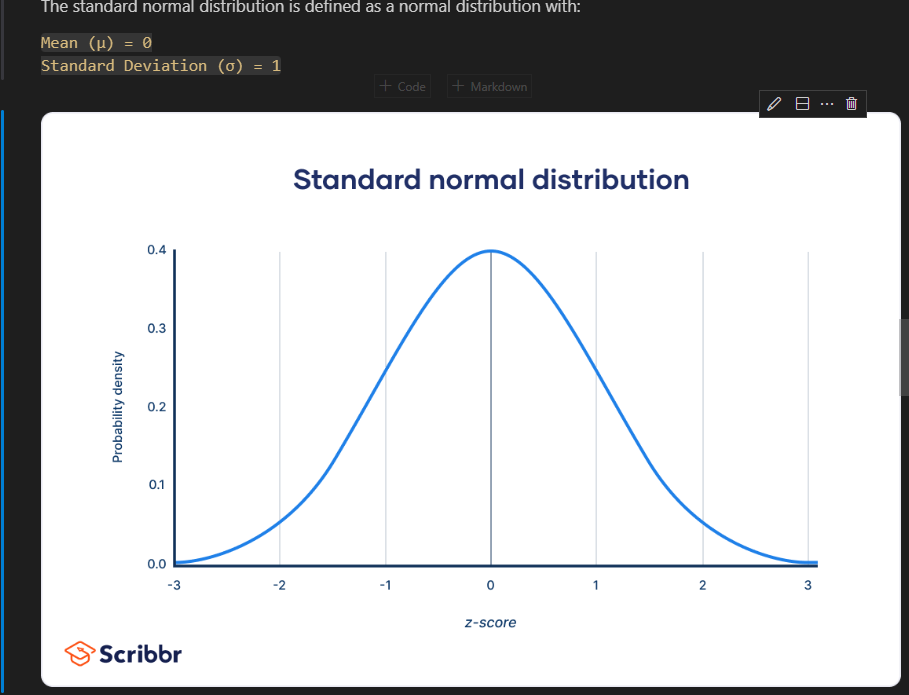
   99.7% within 3 Standard Deviations: Nearly all (99.7%) of the data is within three standard deviations from the mean.

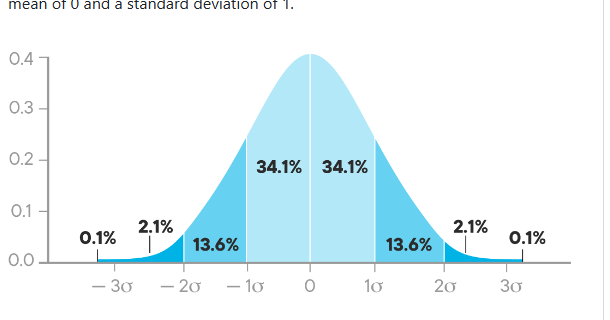


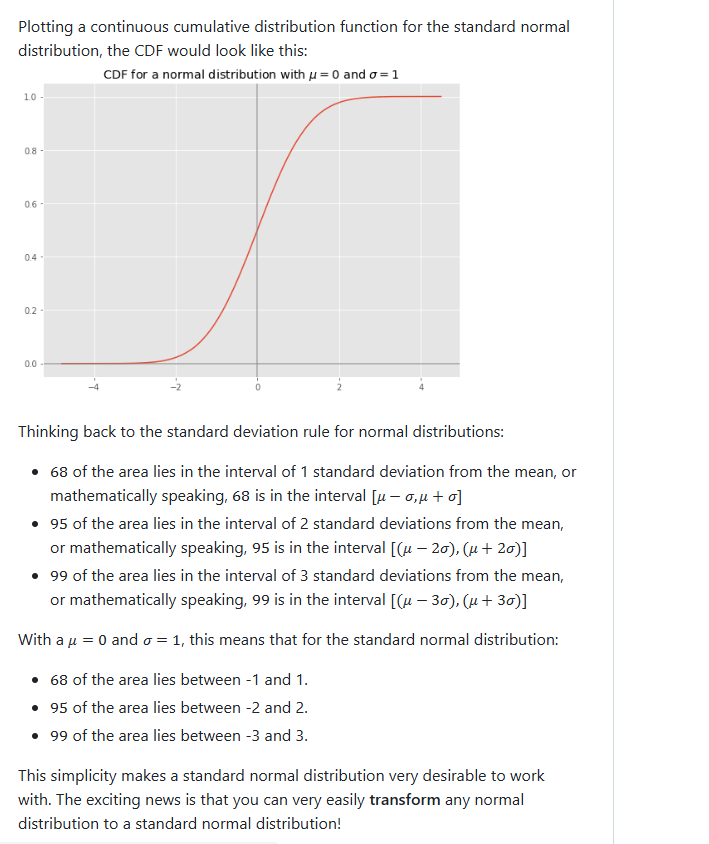


1. **Standard Normal Distribution**- Special case of the normal distribution that has been standardized to simplify analysis and interpretation.Its crucial in statisctics due to its properties and its role in various statisctical methods.

Special case of normal distribution with a mean of 0 and std of 1







**a)Standard score/z-score –** Statisctical measurement that describes a value’s relation to the mean of a group of values.It indicates how many standard deviations a data point is from the mean of the dataset

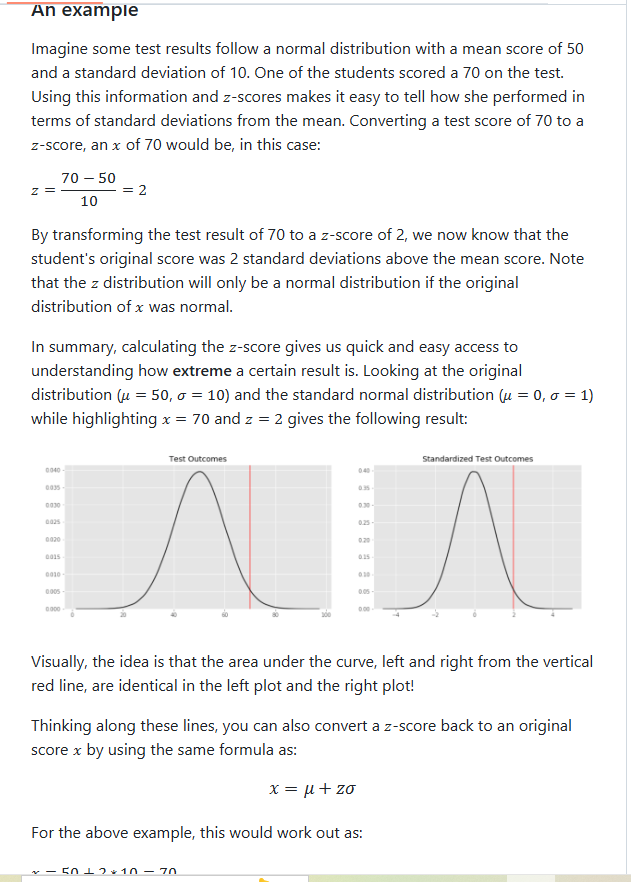
**z=(X−μ)/σ**

Where:

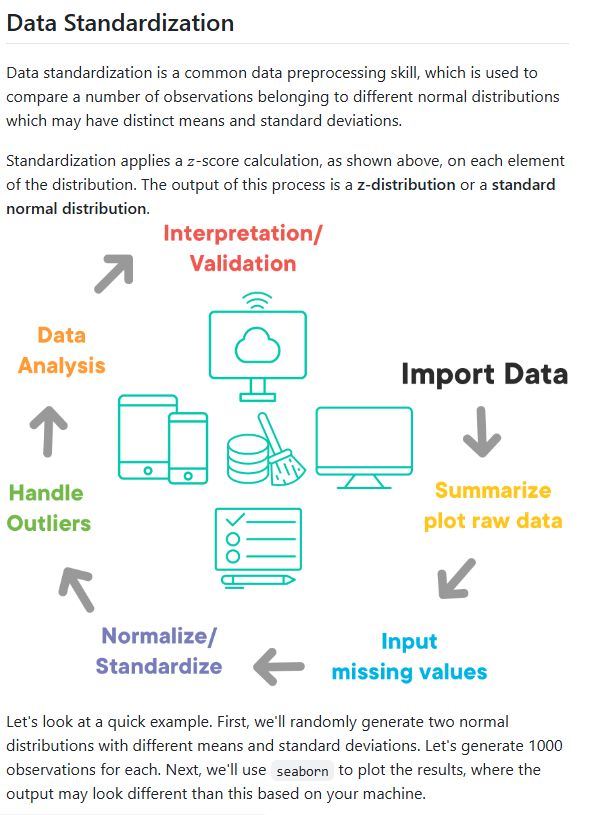
    X is the value in question.

    μ is the mean of the dataset.

    σ is the standard deviation of the dataset.



**b. Data Standardization –** common data preprocessing skill, which is used to compare a number of observations belonging to the different normal distributions which may have distinct means and standard deviations



import numpy as np

import seaborn as sns

mean1,sd1 = 5,3 #dist1

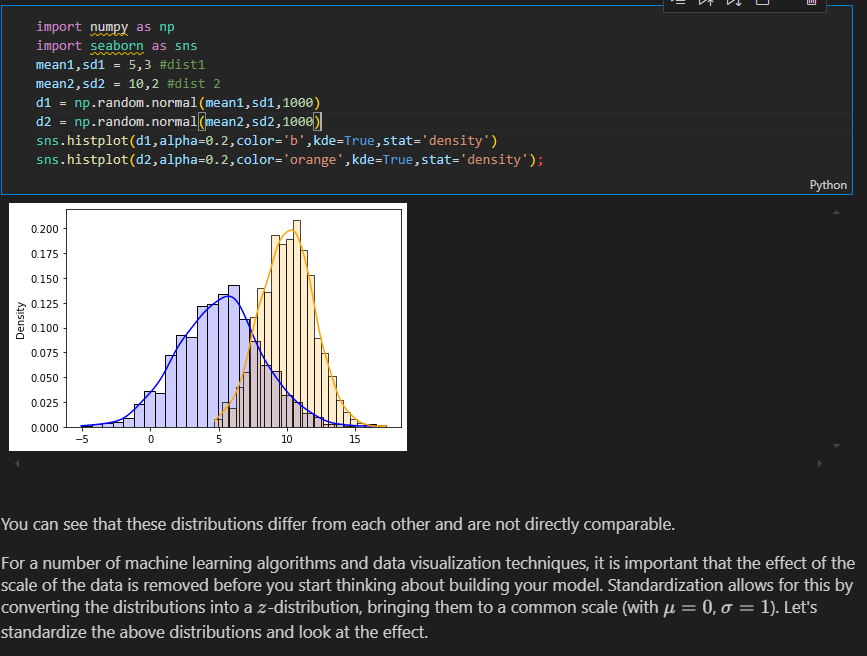
mean2,sd2 = 10,2 #dist 2

d1 = np.random.normal(mean1,sd1,1000)

d2 = np.random.normal(mean2,sd2,1000)

sns.histplot(d1,alpha=0.2,color='b',kde=True,stat='density')

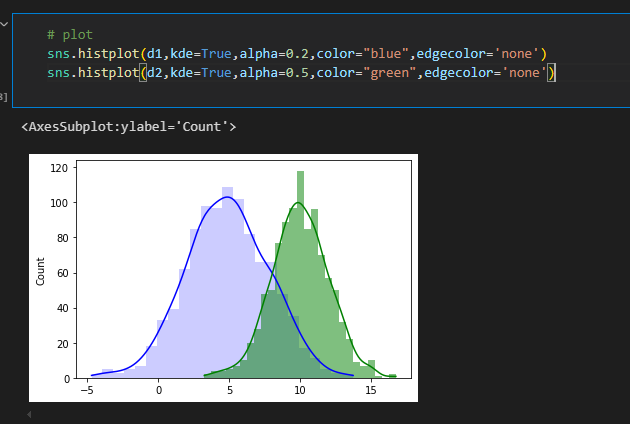
sns.histplot(d2,alpha=0.2,color='orange',kde=True,stat='density');



**TEACHERS CODE using kde instead of stat**

sns.histplot(d1,kde=True,alpha=0.2,color="blue",edgecolor='none')

sns.histplot(d2,kde=True,alpha=0.5,color="green",edgecolor='none')

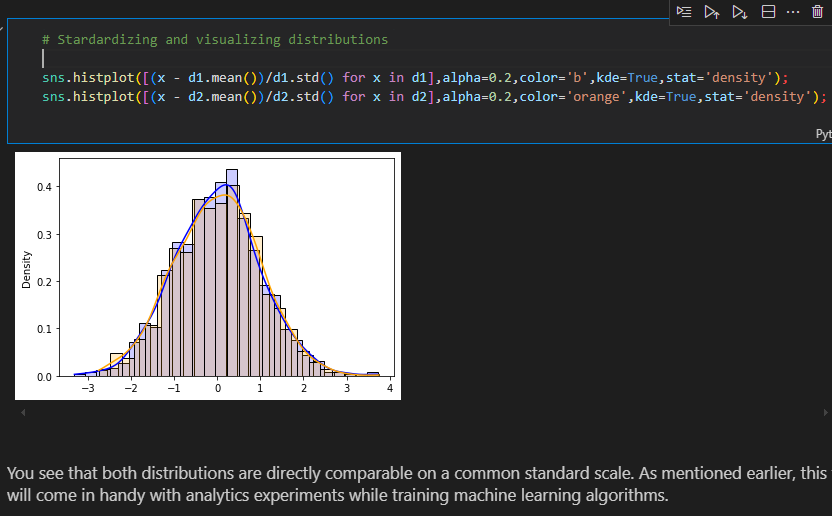


**AFTER STANDARDIZATION**

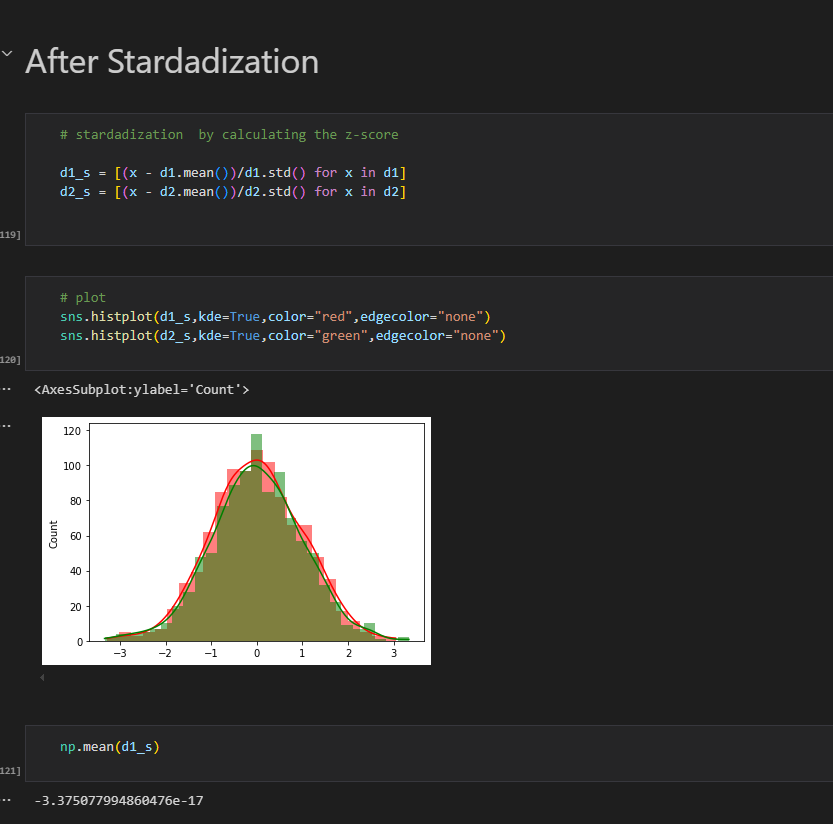
# Stardardizing and visualizing distributions

sns.histplot([(x - d1.mean())/d1.std() for x in d1],alpha=0.2,color='b',kde=True,stat='density');

sns.histplot([(x - d2.mean())/d2.std() for x in d2],alpha=0.2,color='orange',kde=True,stat='density');

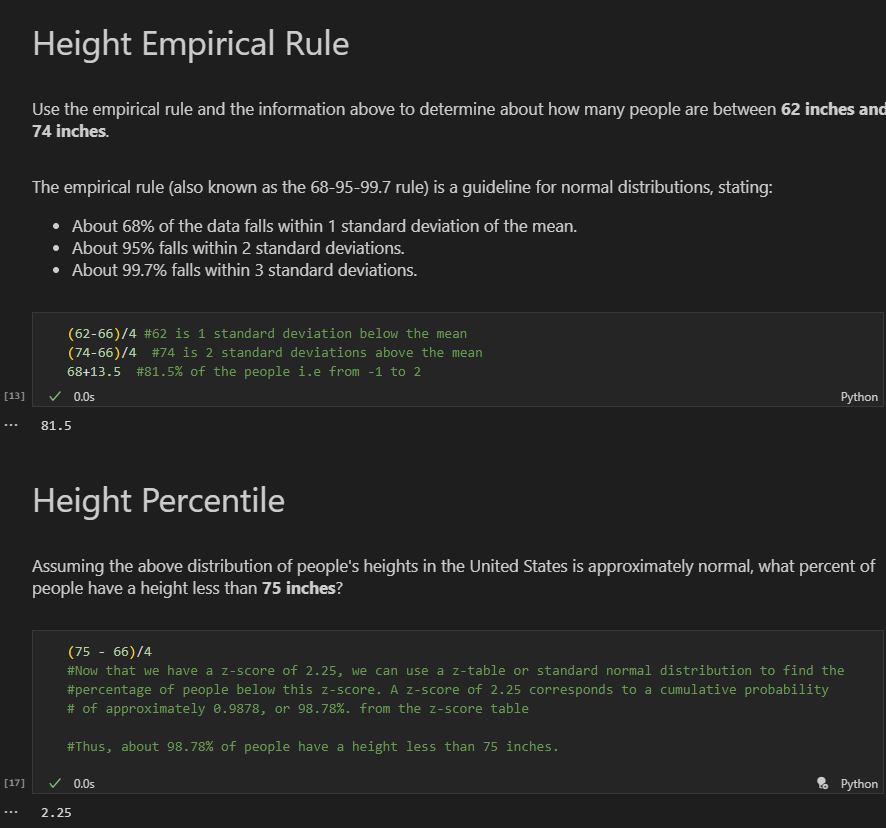


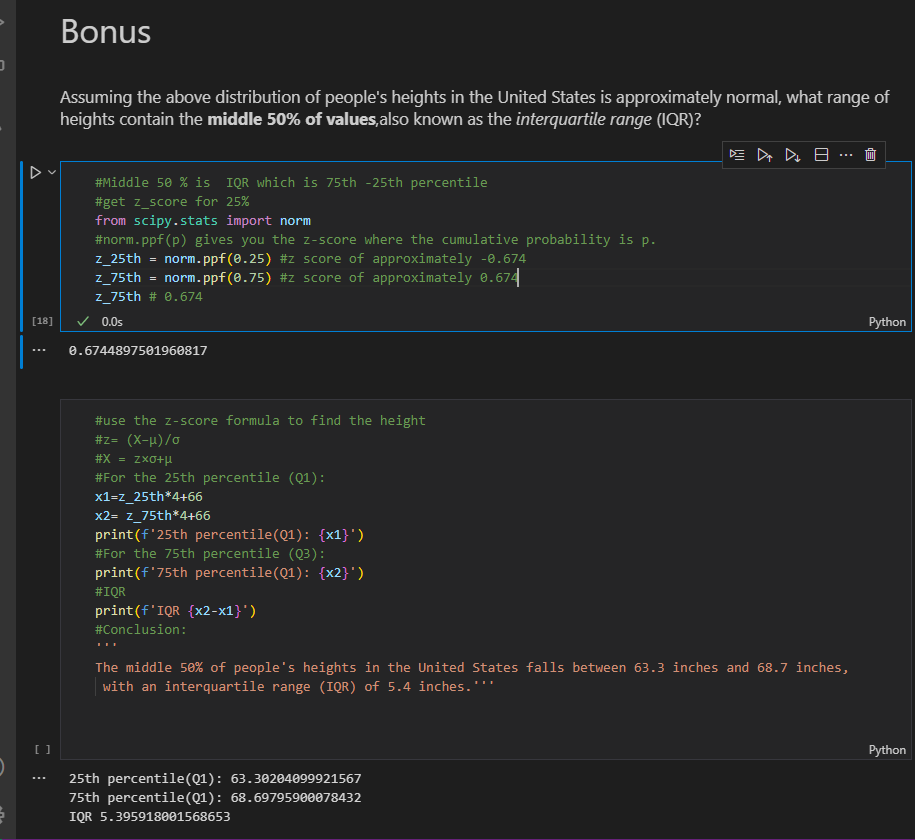
**Teachers code without stat**



1. The **z-score** can be used to understand how extreme a certain result is
2. **Skewness** and **kurtosis** can be used to measure how different a given distribution is from a normal distribution

**Examples of z scores**



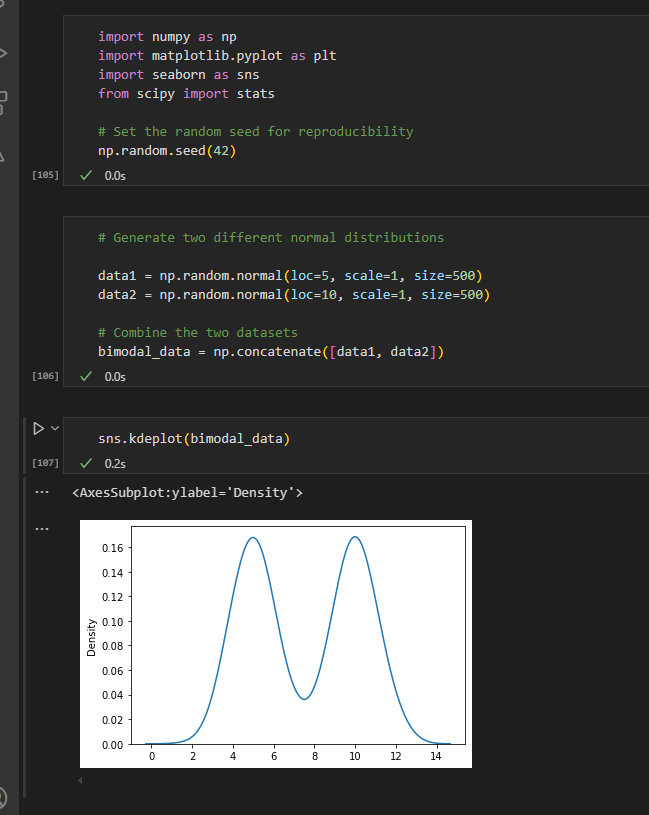


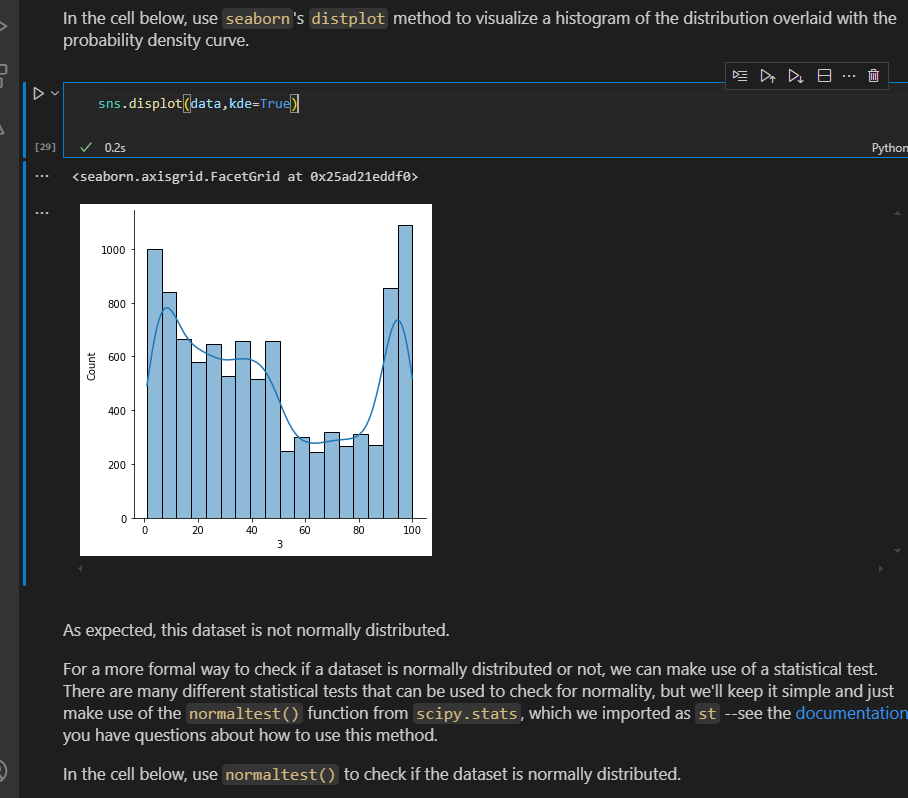
1. **Inferential statistics** – drawing conclusions beyond just describing statisctics of your sample data.Allows us to make claims abt data that we don’t have access to.
2. **Central Limit theorem** – states that given a sufficiently large sample size, the sampling distribution of the sample mean will be approximately normally distributed regardless of the orginal population’s distribution.

allows us to treat non-normal distributions as normal distributions and provides ways for us to estimate paramaters about a population

**CENTAL LIMIT THEOREM**

* **Plot** the data to check **distribution** eg sns.kdeplot(bimodal\_data) OR sns.distplot()





* Also use a **statistical** **test** to check normality

stat,p\_value = st.normaltest(data)

print(stat)

print(p\_value)

alpha = 0.05

#nterpret the result

if p\_value > alpha:

    print('Fail to reject the null hypothesis(data is likely normally distributed)')

else:

    print('Reject the null hypothesis(data is likely not normally distributed)')



Since our dataset is non-normal, that means we'll need to use the **\*\****\_Central Limit Theorem.\_***\*\***

* We **sample** our data

sample\_30 = np.random.choice(bimodal\_data,30)

sample\_30.mean()

* Create a sampling distribution of sample means

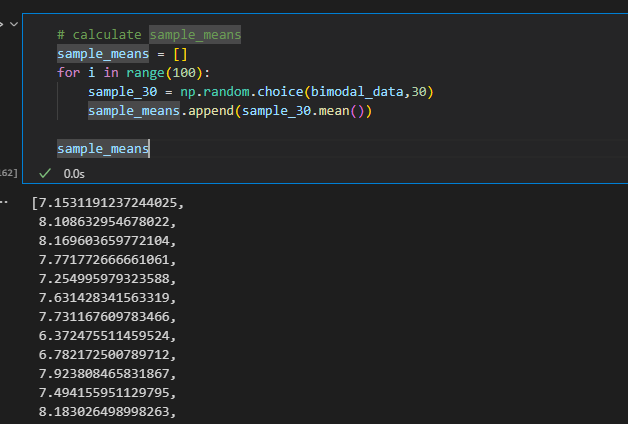
sample\_means = []

for i in range(100):

    sample\_30 = np.random.choice(bimodal\_data,30)

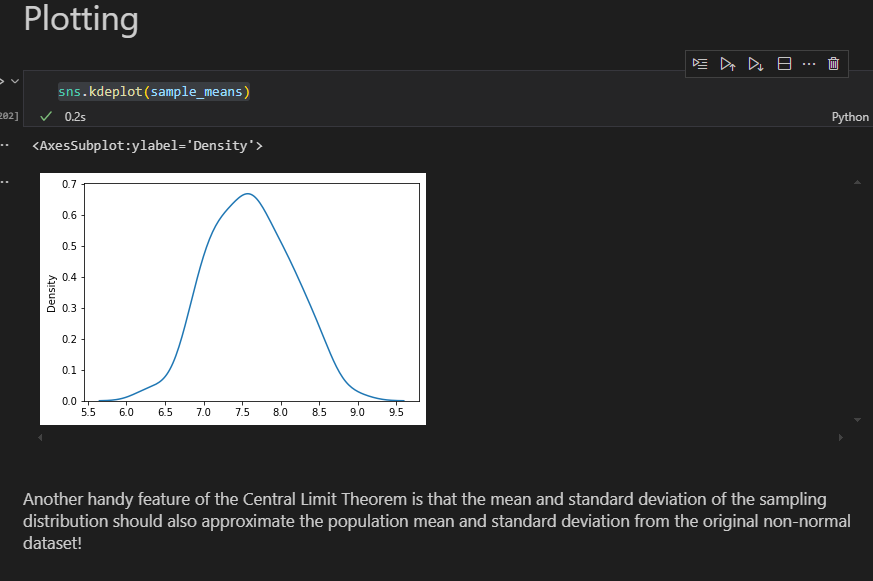
    sample\_means.append(sample\_30.mean())

sample\_means

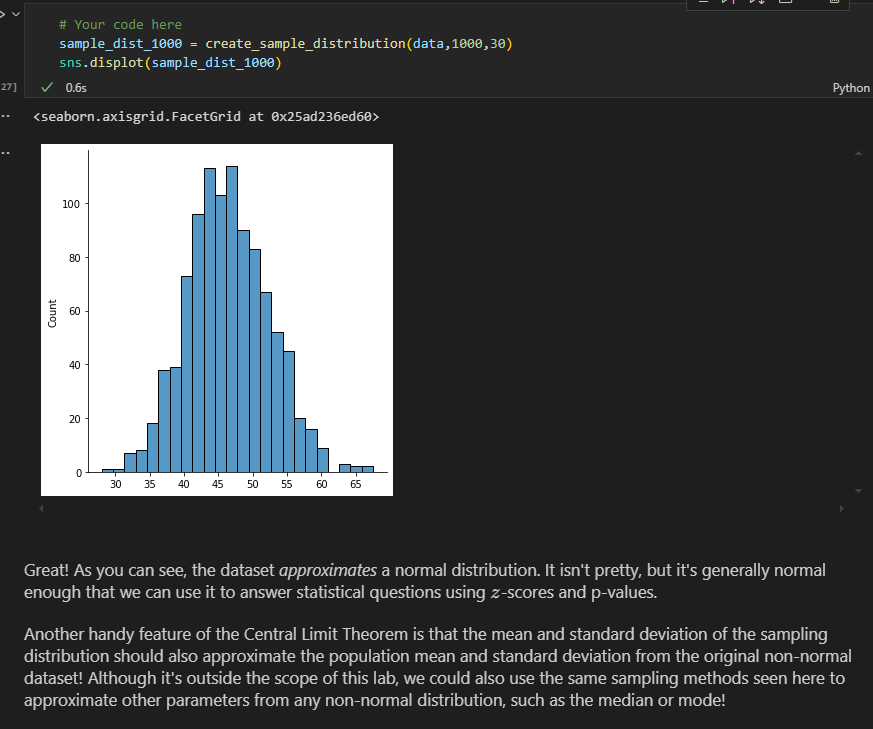


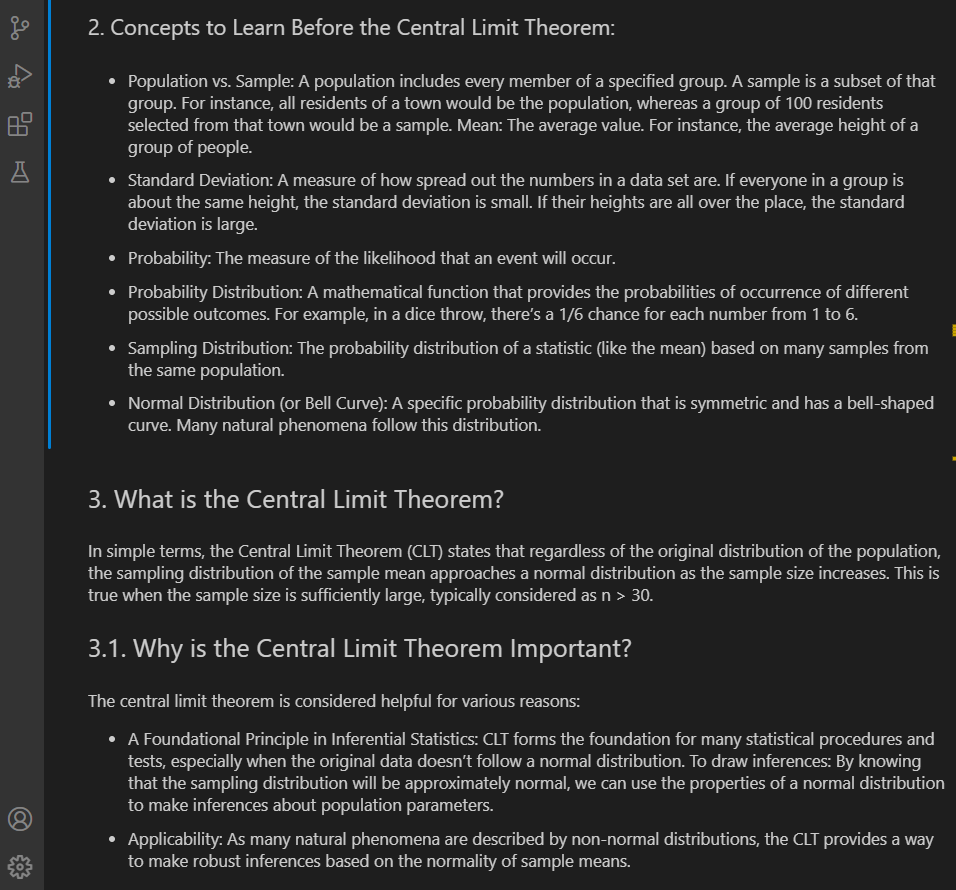
* Plot the sample means

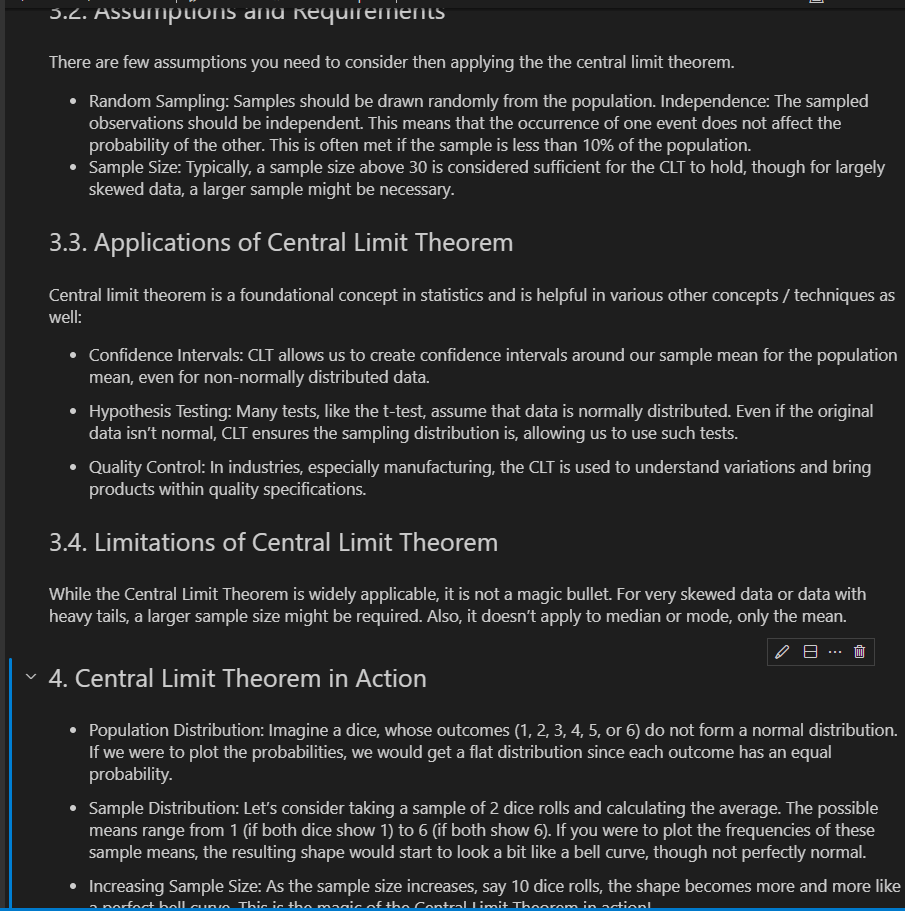
sns.kdeplot(sample\_means)



* Data almost look normal







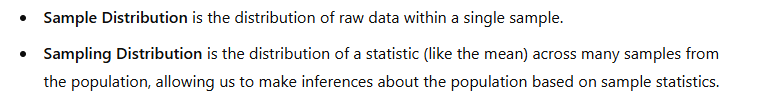
1. **Confidence intervals** – range of values above and below the point estimate that captures the true population parameter at some predetermined confidence level

range of values surrounding an estimated parameter eg 12-18 inches is the height of chairs required for 12 yr-olds

**(If we pulled 100 samples and constructed confidence intervals in the same manner, we would expect that 95 of the intervals would contain the true mean of population age. )**

**Sample Distribution -** The sample distribution is the distribution of values in a single sample taken from a population.

**Sampling Distribution -** The sampling distribution is the distribution of a sample statistic (e.g., the mean, proportion, or variance) across multiple samples taken from the same population

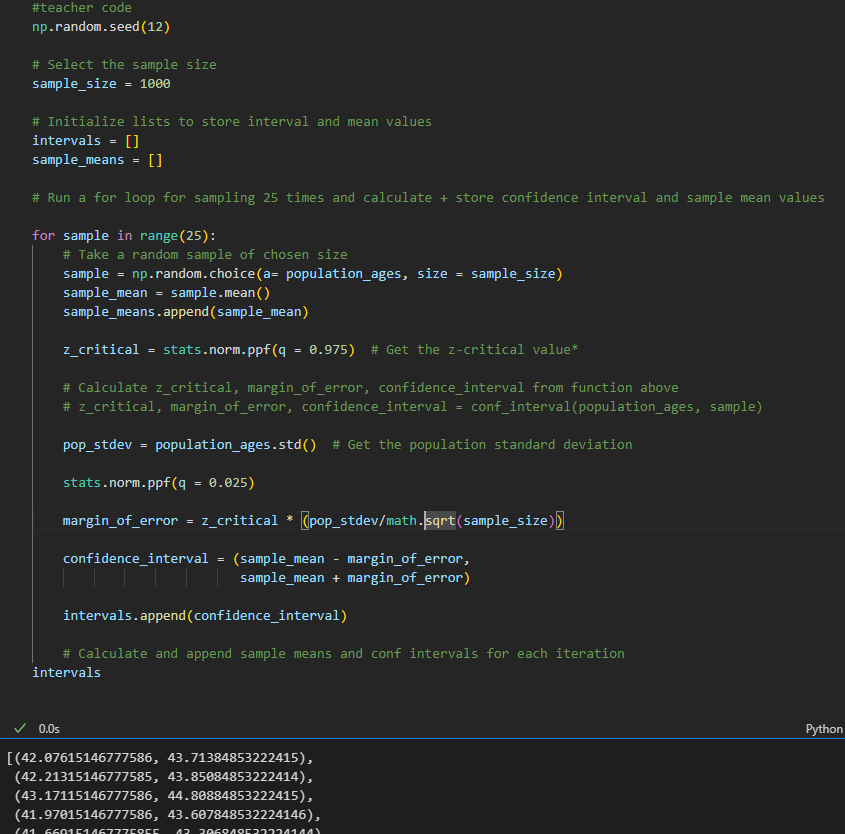


Calculate the Confidence Interval: Using the formula:

**CI=xˉ±z∗⋅SE**

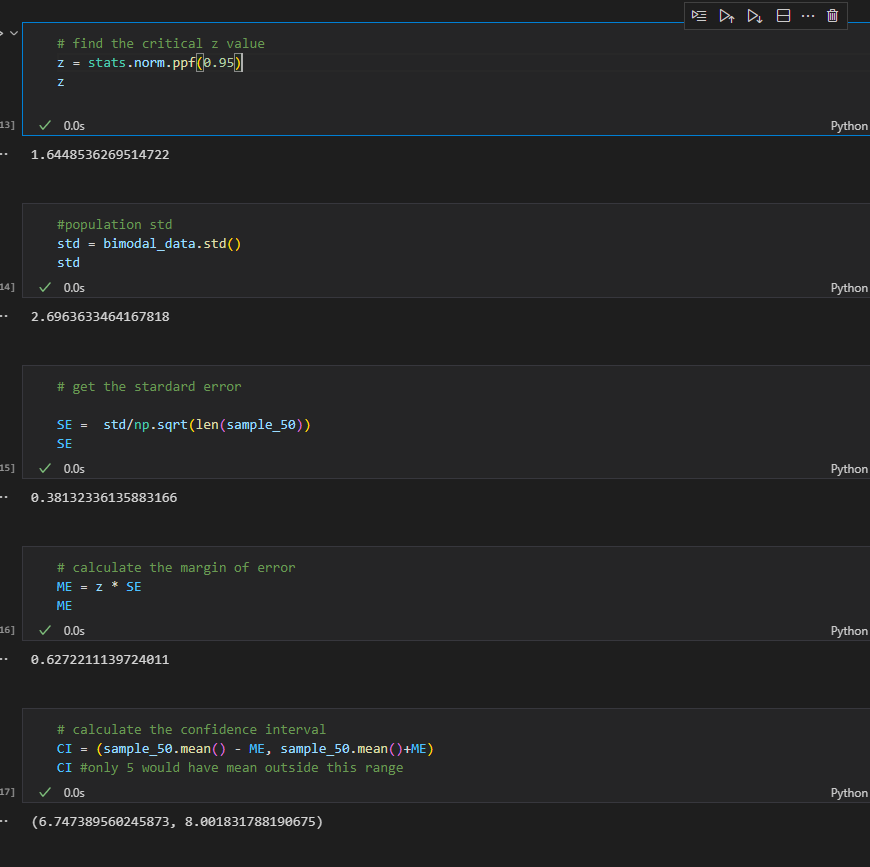
**Margin of Error = z ∗ σ / √n #** where SE = ∗ σ / √n meaning MOE = z\*SE

Example:



**If we pulled 100 samples and constructed confidence intervals in the same manner, we would expect that 95 of the intervals would contain the true mean of population age.**

**Example 2:**

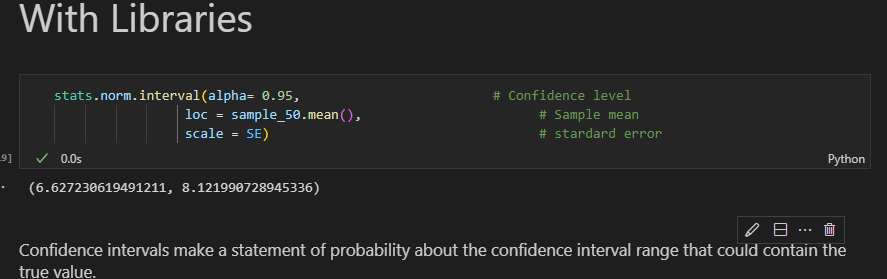


**With library**

stats.norm.interval(alpha= 0.95,                         # Confidence level

                 loc = sample\_50.mean(),                       # Sample mean

                 scale = SE)

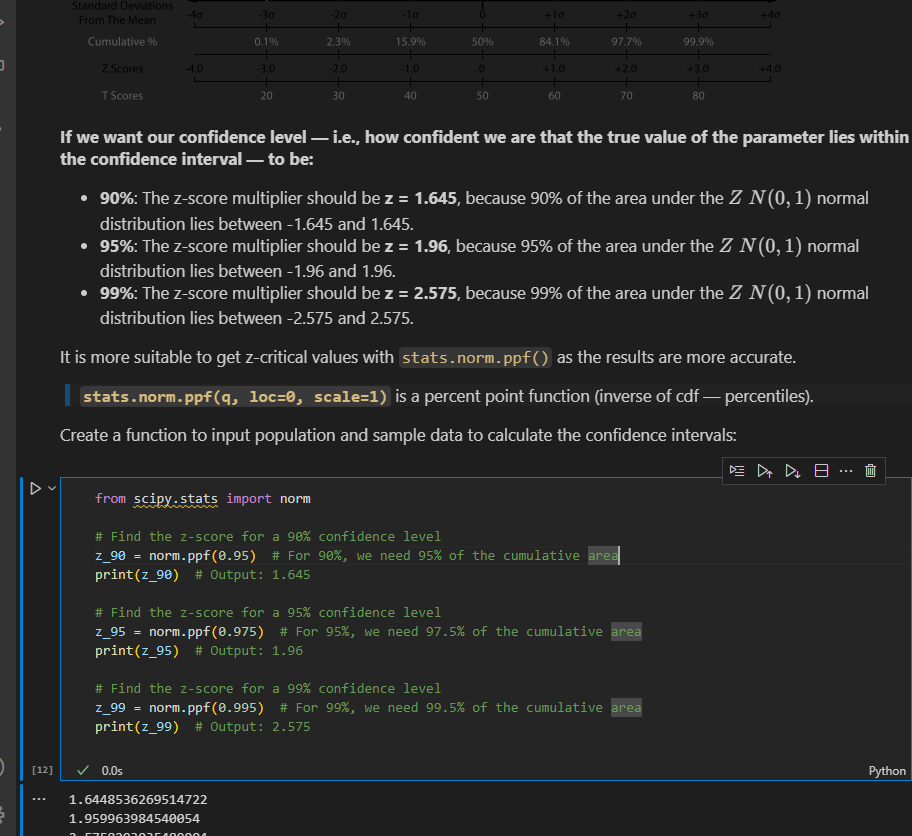


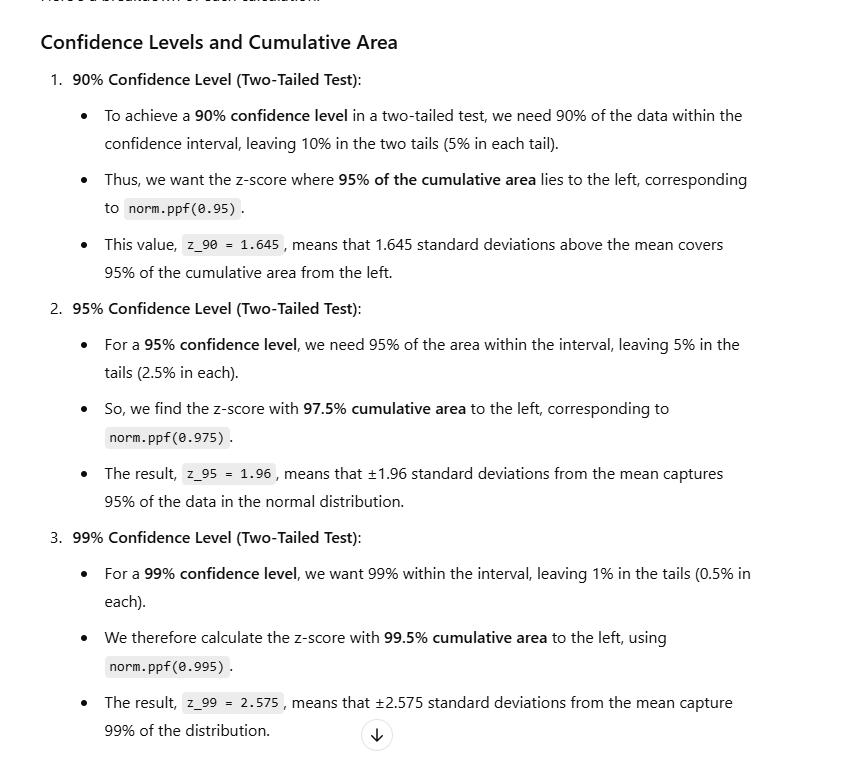
**Zcore for a two-tailed test**

90% = norm.ppf(95)

Stats.norm.ppf()

z\_80 = stats.norm.ppf(0.9)



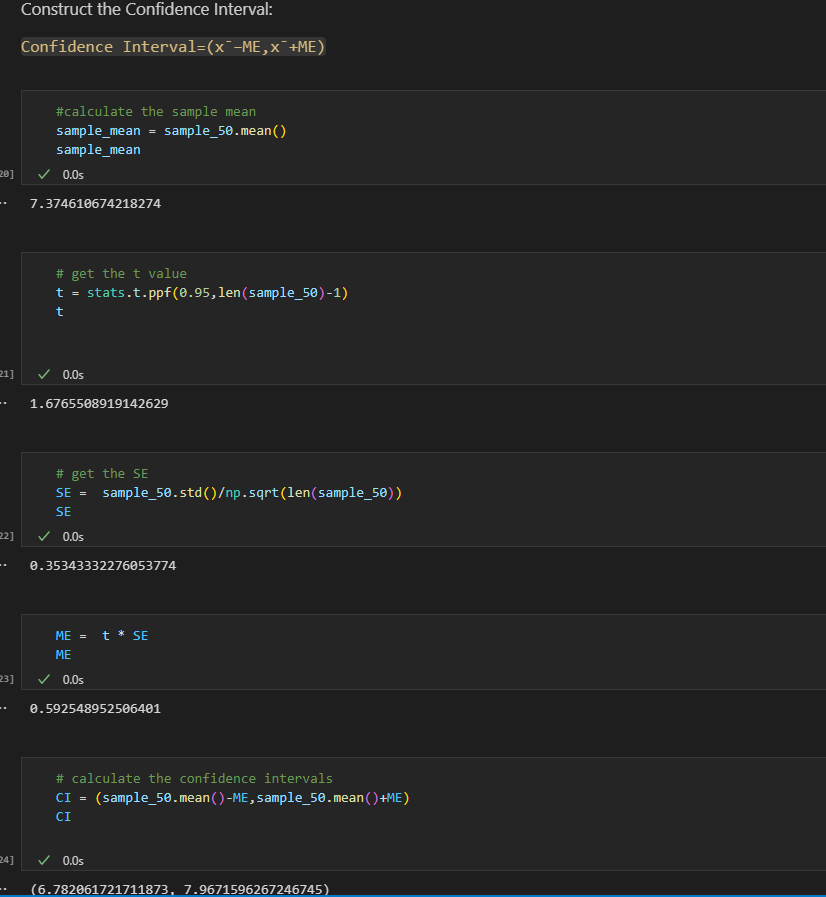


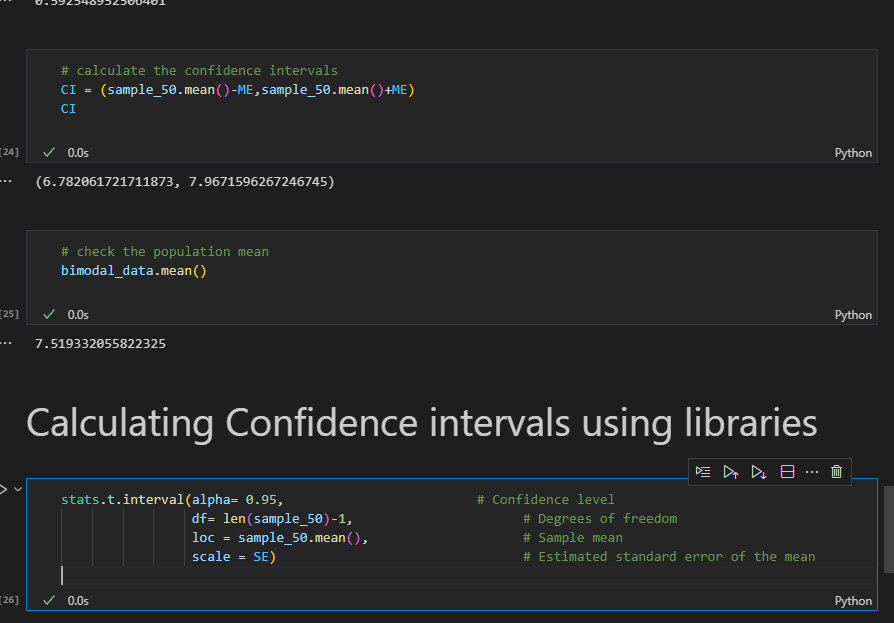
**CONFIDENCE INTERVALS WITH T DISTRIBUTION**

* Used in statisctics particularly when dealing with small sample sizes or when the population standard deviation is unknown

**Example 1:**

Stats.t.ppf(0.95,ddof) where ddof =len(sample)-1(get t-value) for 90%CI



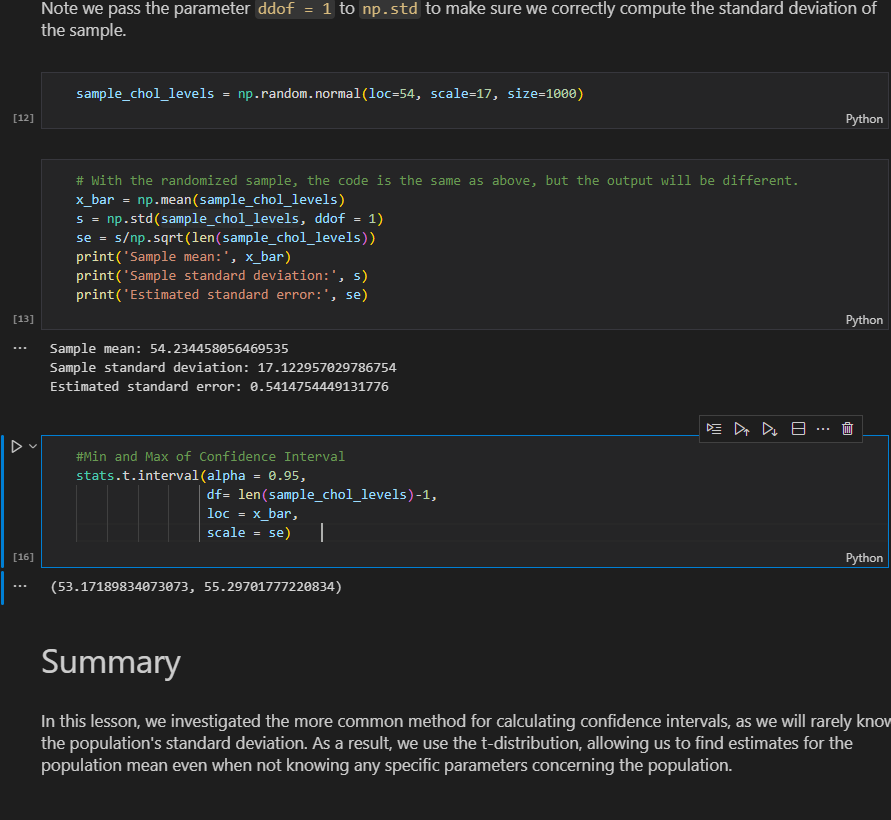


**EXAMPLE 2:**

**NB: added** ddof when calculating standard deviation



**Example 3: Using libraries**



**Example 4: Using a real life scenario**

n = 30

mean = 4.8

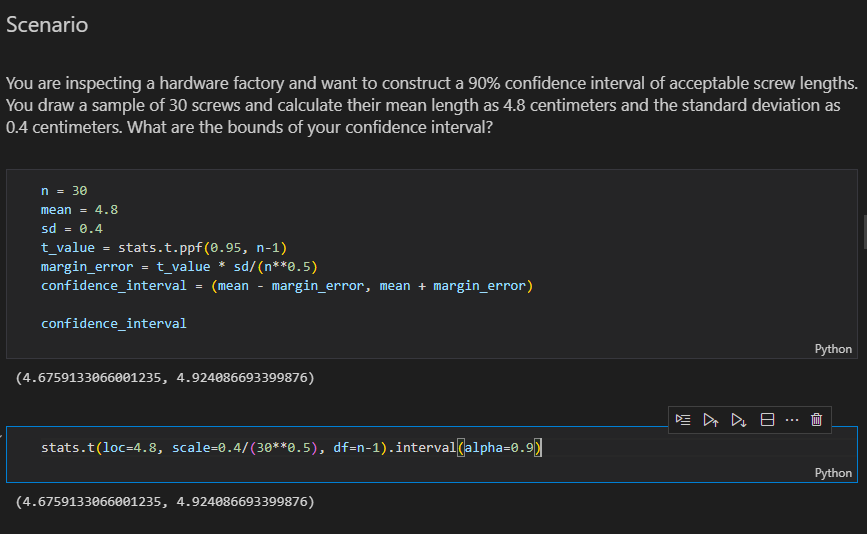
sd = 0.4

t\_value = stats.t.ppf(0.95, n-1)

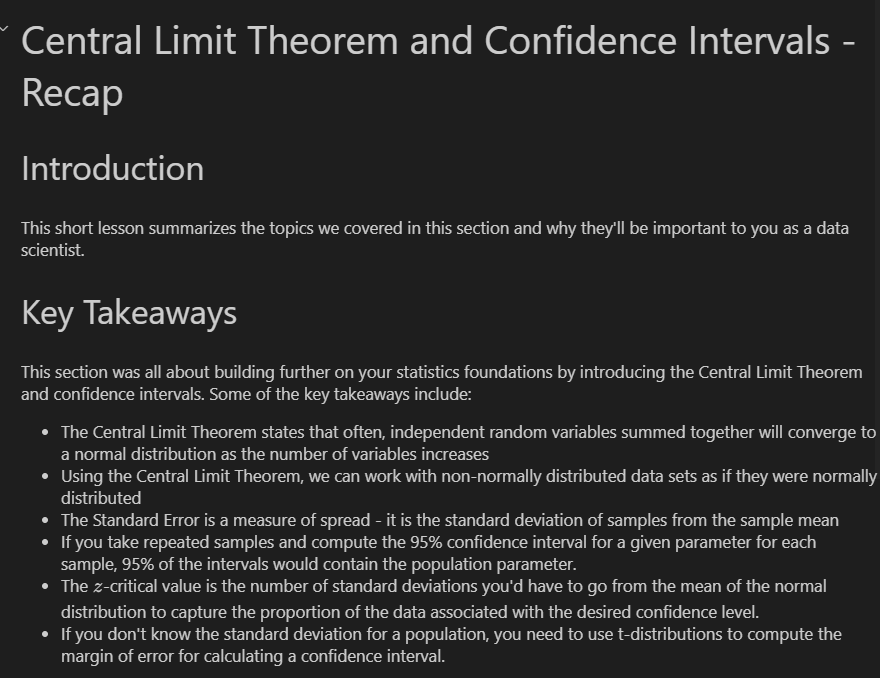
margin\_error = t\_value \* sd/(n\*\*0.5)

confidence\_interval = (mean - margin\_error, mean + margin\_error)

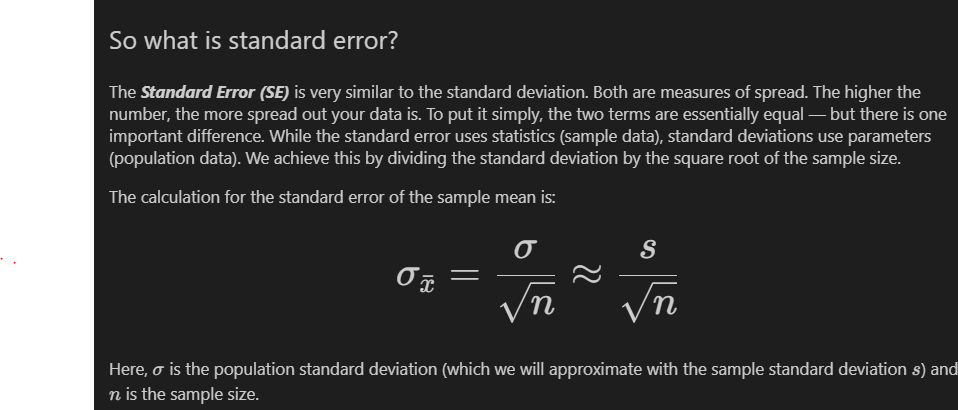
confidence\_interval



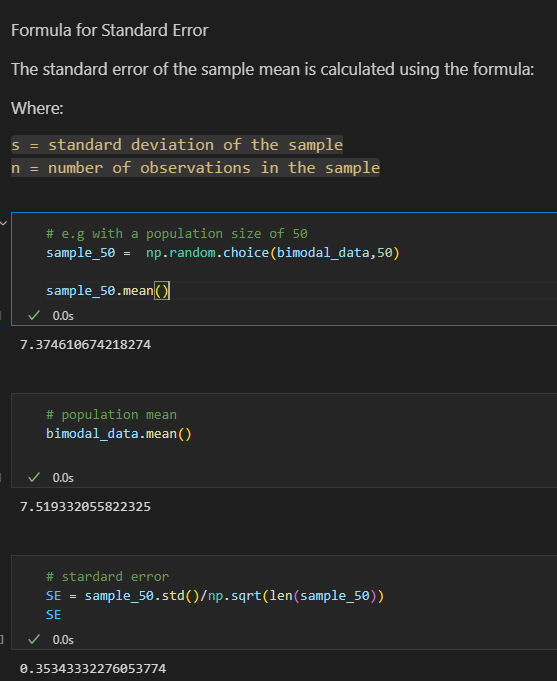
**RECAP ON CENRTAL LIMIT THEOREM AND CONFIDENCE LEVEL**



1. **Z distribution** – requires that we know the standard deviation of the population while **t-distribution** allows us to work with samples where the population standard deviation is unkown(as well as as smaller samples) in order to form confidence intervals
2. **Standard Error** – measure of the variability or dispersion of a sample mean estimate relative to the true population mean.it quantifies how much the sample mean is expected to fluctuate from the true population mean due to sampling variability



Example:



1. Generate data points using poisson distribution

population\_ages1 = stats.poisson.rvs(loc=18,mu=35,size=150000)

population\_ages2 = stats.poisson.rvs(loc=18,mu=10,size=100000)

population\_ages = np.concatenate((population\_ages1,population\_ages2))

1. Create normal distribution data from scipy

population = list(stats.norm.rvs(size=1000,

                          random\_state=42))

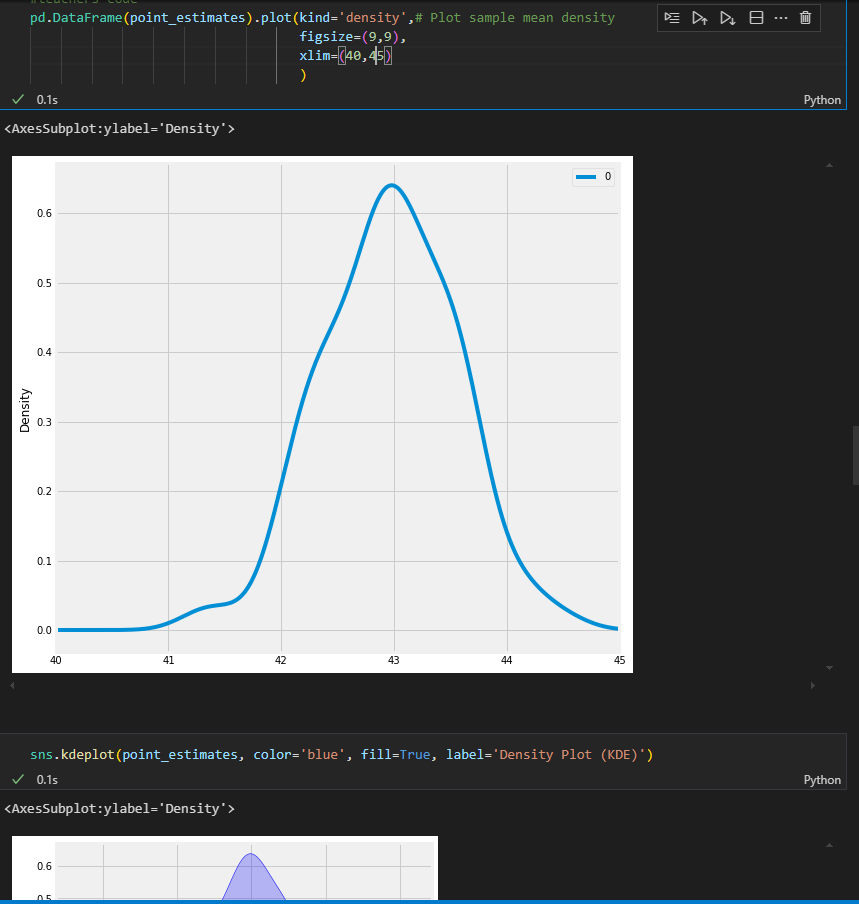
1. Convert single column to a dataframe and plot

pd.DataFrame(point\_estimates).plot(kind='density',# Plot sample mean density

                                   figsize=(9,9),

                                   xlim=(40,45)

                                   )



1. Adding mean formula labels(x-bar using code)

import matplotlib.pyplot as plt

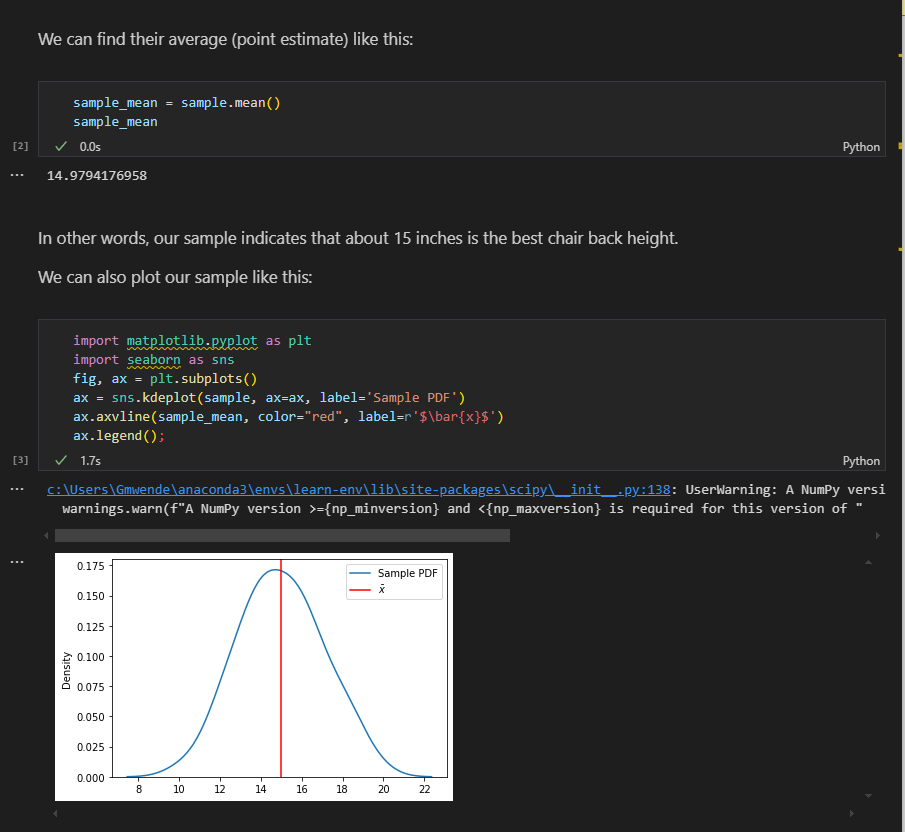
import seaborn as sns

fig, ax = plt.subplots()

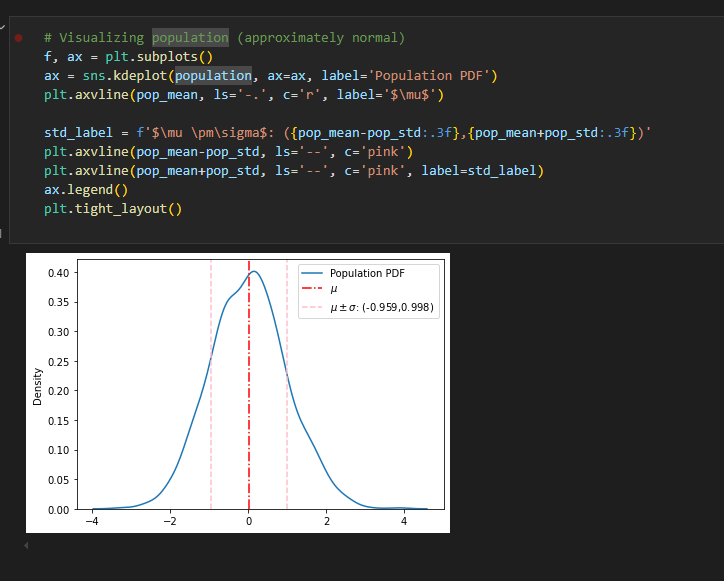
ax = sns.kdeplot(sample, ax=ax, label='Sample PDF')

ax.axvline(sample\_mean, color="red", label=r'$\bar{x}$')

ax.legend();



1. Add mu and sigma label using code
2. f, ax = plt.subplots()
3. ax = sns.kdeplot(population, ax=ax, label='Population PDF')
4. plt.axvline(pop\_mean, ls='-.', c='r', label='$\mu$')
5. std\_label = f'$\mu \pm\sigma$: ({pop\_mean-pop\_std:.3f},{pop\_mean+pop\_std:.3f})'
6. plt.axvline(pop\_mean-pop\_std, ls='--', c='pink')
7. plt.axvline(pop\_mean+pop\_std, ls='--', c='pink', label=std\_label)
8. ax.legend()
9. plt.tight\_layout()

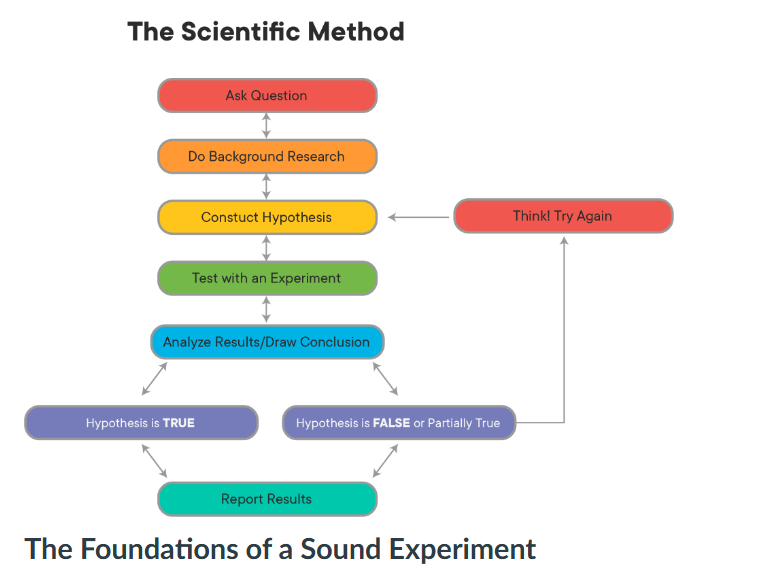


1. **Statistical significance –** determination about whether the observed effect in the data is unlikely to have occurred by random chance alone.
2. **Significance level** = alpha eg **0.05(5%) 95%** confidence level or **0.01(1%)99%** confidence level
3. **HYPOTHESIS TESTING**

**a).General structure of an experiment**

* **Make an Observation**- observe sth you want to test. During this step, you must observe phenomena to help refine the question that you want to answer. This might be anything from "does this drug have an effect on headaches?" to "does the color of this button affect the number of sales a website makes in a day?". Before testing these ideas, you need to observe that there might be some phenomena occurring and then come up with a specific question to answer.
* **Examine the research -** Good data scientists work smart before they work hard. In the case of the scientific method, this means seeing what research already exists that may help you answer your question, directly or indirectly. It could be that someone else has already done an experiment that answers your question--if that's the case, you should be aware of that experiment before starting your own, as it could inform your approach to structuring your experiment, or maybe even answer your question outright!
* **Form an hypothesis -** you'll formulate 2 hypotheses to test--your educated guess about the outcome is called the *Alternative Hypothesis*, while the opposite of it is called the *Null Hypothesis*.
* **Conduct an experiment -** This step is the part of the scientific method that will be the focus of this section. You can only test a hypothesis by gathering data from a well-structured experiment. A well-structured experiment is one that accounts for all of the mistakes and randomness that could give you false signals relating to the effect of an intervention. Just because you're running an experiment doesn't prove that A causes B, or that there's even a relationship between A and B! A poorly designed experiment will lead to false conclusions that you haven't considered or controlled for. A well-designed experiment leaves you no choice but to acknowledge that the effects seen in a dependent variable are related to an independent variable. The world is messy and random. You have to account for this messiness and randomness in experiments so that you can filter it out and be left only with the things you're actively trying to measure.
* **Analyze experimental Results-**  All the work you've done with statistics is usually in service of this goal--looking at the data and understanding what happened. During this step, you will tease out relationships, filter out noise, and try to determine if something that happened is ***statistically significant*** or not.
* **Draw conclusions -** This step is the logical endpoint for an experiment. You've asked a question, looked at experimental results from others that could be related to your question, made an educated guess, designed an experiment, collected data, and analyzed the results.

The reality of this step is that you use your analysis of the data to do one of two things: either ***reject the null hypothesis or fail to reject the null hypothesis***



***b).*Foundations of a sound experiment**

All experiments are not created equal--simply following the steps outlined above does not guarantee that the results of any experiment will be meaningful. For instance, there's nothing stopping a person from testing the hypothesis that "wearing a green shirt will make it rain tomorrow!", seeing rain the next day, and rejecting the null hypothesis, thereby incorrectly "proving" that their choice of wardrobe affected the weather. Good experiments demonstrate that independent variables {X} have an effect on the dependent variables {Y} because you control for all the other things that could be affecting {Y}, until you are forced to conclude that the only thing that explains what happened to {Y} is {X}!

Although there are many different kinds of experiments, there are some fundamental aspects of experimental design that all experiments have:

* **Control group/Random controlled Trials** -

One of the most important aspects of a sound experiment is the use of a ***Control Group***. A Control Group is a cohort that receives no treatment or intervention--for them, it's just business as usual. In a medical test, this might be a ***placebo***, such as a sugar pill. In the example of testing the color of a button on a website, this would be customers that are shown a version of the website with the button color unchanged. Using a control group allows researchers to compare the results of doing nothing (the control) with the effects of doing something (the ***intervention***). Without a control group, you have no way of knowing how much of the results you see can be attributed to the intervention, and how much would have happened anyway

**NB:** **A control group is only a control group if they are sampled from the same population as the treatment groups!**

The main way scientists deal with this is through ***Random Controlled Trials***. In a Random Controlled Trial, there is a control group and an intervention (also called treatment) group, where subjects are ***randomly assigned to each***.

A ***Single-Blind*** or ***Blind Trial*** is one where the participant does not know if they are receiving the treatment or a placebo.

A ***Double-Blind Trial*** is one where the participant does not know if they are receiving the treatment or a placebo, and neither does the person administering the experiment (because their bias could affect the outcomes, too!). Instead, knowing whether someone received the treatment or a placebo is kept hidden from everyone until after the experiment is over (obviously, *someone* has to know for recordkeeping purposes, but that person stays away from the actual experiment to avoid contaminating it with bias from that knowledge).

* **Appropriate Sampling techniques and sample size**

When data scientists are performing experiments, they rarely have the opportunity to work with an entire population of data. Rather, they must obtain a sample that is representative of the population. In order to get a high quality sample, you should follow these four assumptions related to sampling techniques and sample size.

* **Sample is independent**

Independence means the value of one observation does not influence or affect the value of other observations. Independent data items are not connected with one another in any way (unless you account for it in your model). This includes the observations in both the “between” and “within” groups of your sample. Non-independent observations introduce bias and can make your statistical test give too many false positives.

* **Sample is collected randomly**

A sample is random when each data point in your population has an equal chance of being included in the sample; therefore, the selection of any individual observation happens by chance, rather than by choice. This reduces the chance that differences in materials or conditions strongly bias results. Random samples are more likely to be representative of the population; therefore, you can be more confident with your statistical inferences with a random sample.

* **The sample is approximately normally distributed**

The normal distribution assumption is that the sampling distribution of the mean is normal. That is, if you took a sample, calculated its mean, and then you took another (independent) sample (from the same population) and got its mean (and repeated this an infinite number of times), then the distribution of the values that you wrote down would always be a perfect bell curve. This is the principle behind the Central Limit Theorem, and it is this idea that allows us to perform hypothesis tests. While maybe surprising, this assumption turns out to be relatively uncontroversial, at least when each of the samples is large, such as N ≥ 30.

* **Appropriate Sample Size**

Randomness is a big problem in experiments. It can lead you to false conclusions by making you think that something doesn't matter when it does, or vice versa. Small sample sizes make experiments susceptible to the problem of randomness; whereas, large sample sizes protect experiments from it. The following scenario illustrates this point:

A person tells you that they can predict the outcome of a fair coin flip. You flip a coin, they call "tails", and they are correct. Is this enough evidence to accept or reject this person's statement? What if they got it right 2 times in a row? 5 times in a row? 55 times out of 100?

This situation illustrates two things that are important for us to understand and acknowledge:

1. No matter how large your sample size, there's always a chance that your results can be attributed to randomness or luck.
2. At some point, you would cross a threshold where random chance is small enough that you'd say "this probably isn't random", and are okay with accepting the results as the result of something other than randomness or luck.

With the situation above, you probably wouldn't assume that this person can predict coin flips after only seeing them get 1 correct. However, if this person got 970 out of 1000 correct, you would probably believe very strongly that this person *can* predict coin flips because the odds of guessing randomly and getting 970/1000 correct are very, very small--but not 0!

Large sample sizes protect us from randomness and variance. A more realistic example would be testing a treatment for HIV. Less than 1% of the global population carries a protective mutation that makes them resistant to HIV infection. If you took a randomly selected sample of 1 person from the population, there is a ~1% chance that you may mistakenly attribute successful prevention to the drug you're testing, when the results really happened because you randomly selected a person with this mutation. However, if your sample size was 100 people per sample, your odds of randomly selecting 100 people with that mutation are *0.01100* . The larger your sample size, the more unlikely it is that you randomly draw people that happen to affect your study in a way that is not reflected by the general population.

* **Reproducibility**
* This one is a big one, and it represents a bit of a crisis in some parts of the scientific community right now. Good scientific experiments have **Reproducible Results**! This means that if someone else follows the steps you outline for your experiment and performs it themselves, they should get pretty much the same results as you did (allowing for natural variance and randomness). If many different people try reproducing your experiment and don't get the same results, this might suggest that your results are due to randomness, or to a **lurking variable** that was present in your samples that wasn't present in others. Either way, a lack of reproducibility often casts serious doubts on the results of a study or experiment.
* This is less of a problem for data scientists, since reproducibility usually just means providing the dataset you worked with and the corresponding Jupyter notebook. However, this isn't always the case! Luckily, you can use code to easily run your experiments multiple times and show reproducibility. When planning experiments, consider running them multiple times to ensure to really help show that your results are sound, and not due to randomness!