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('''

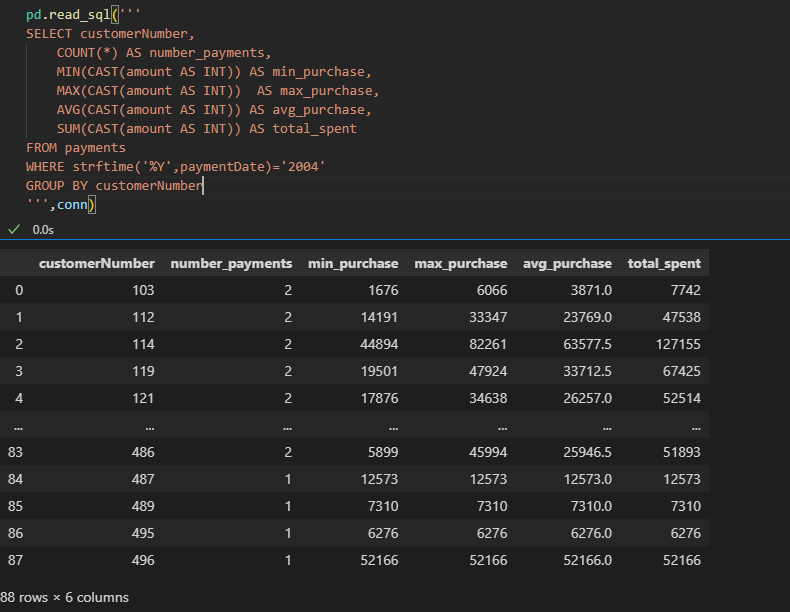
select cast(round(priceEach) as INTEGER) as rounded\_price\_int

        from orderDetails

            ''',conn)

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

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



pd.read\_sql('''

select orderDate,

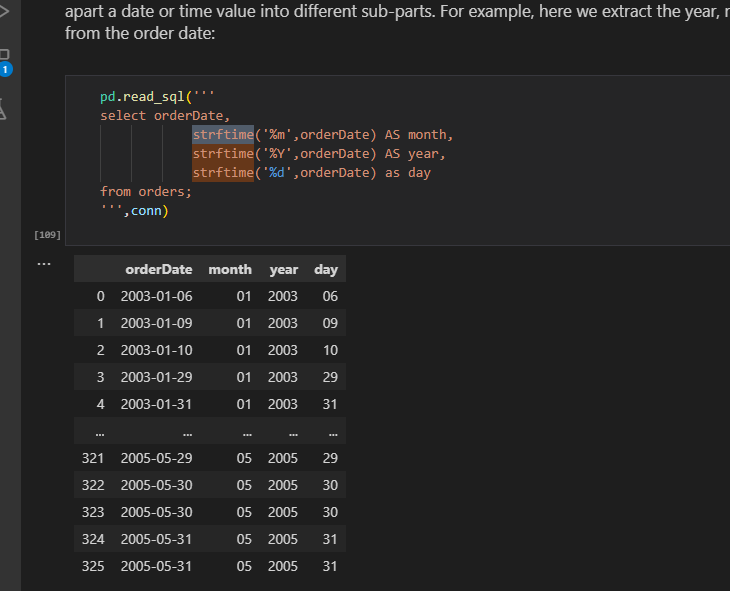
            strftime('%m',orderDate) AS month,

            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,

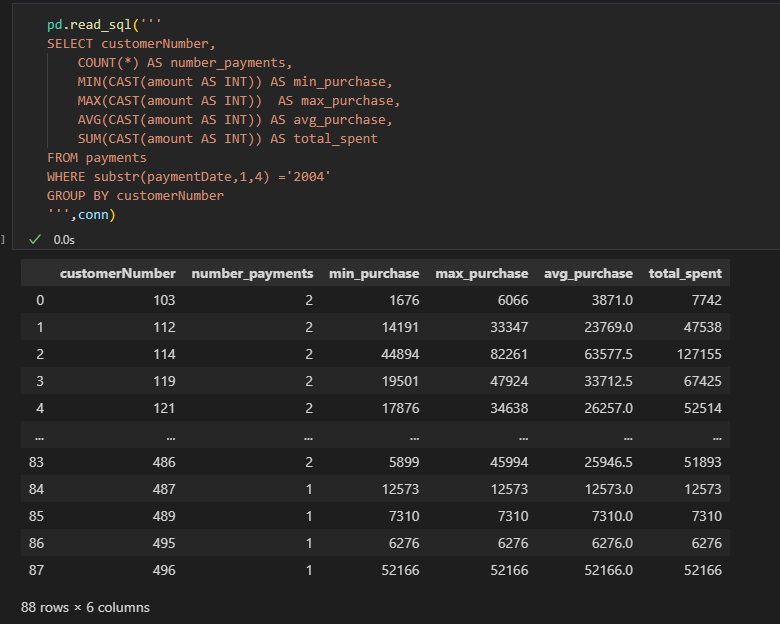
    SUM(CAST(amount AS INT)) AS total\_spent

FROM payments

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

GROUP BY customerNumber

''',conn)



**3.Convert select statement to dataframe**

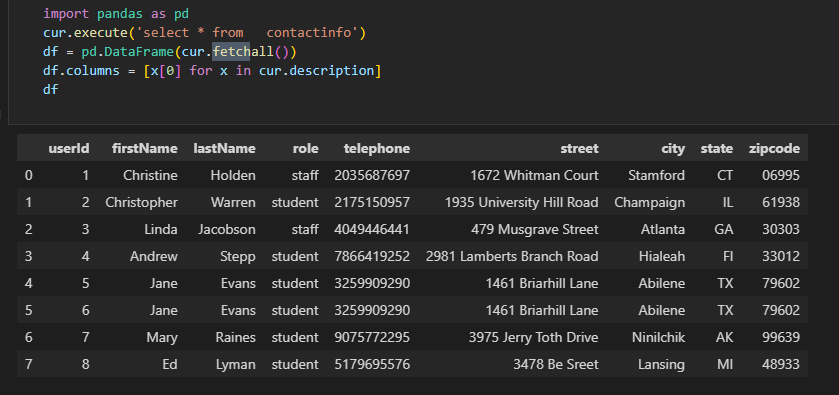
import pandas as pd

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).**

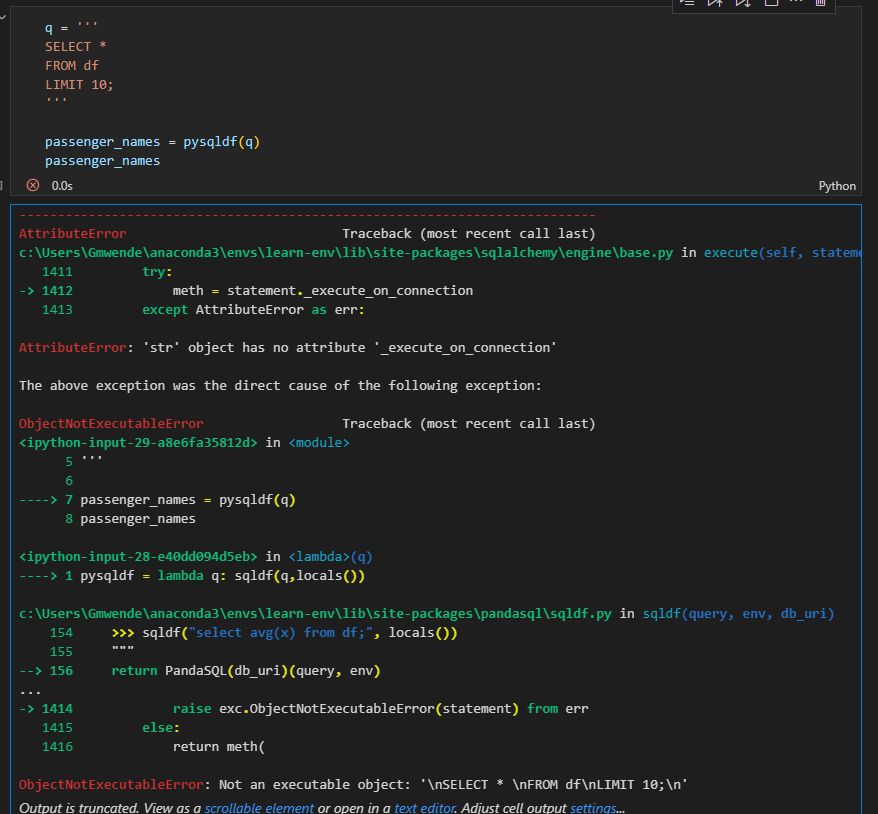
**TO update use**

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:')

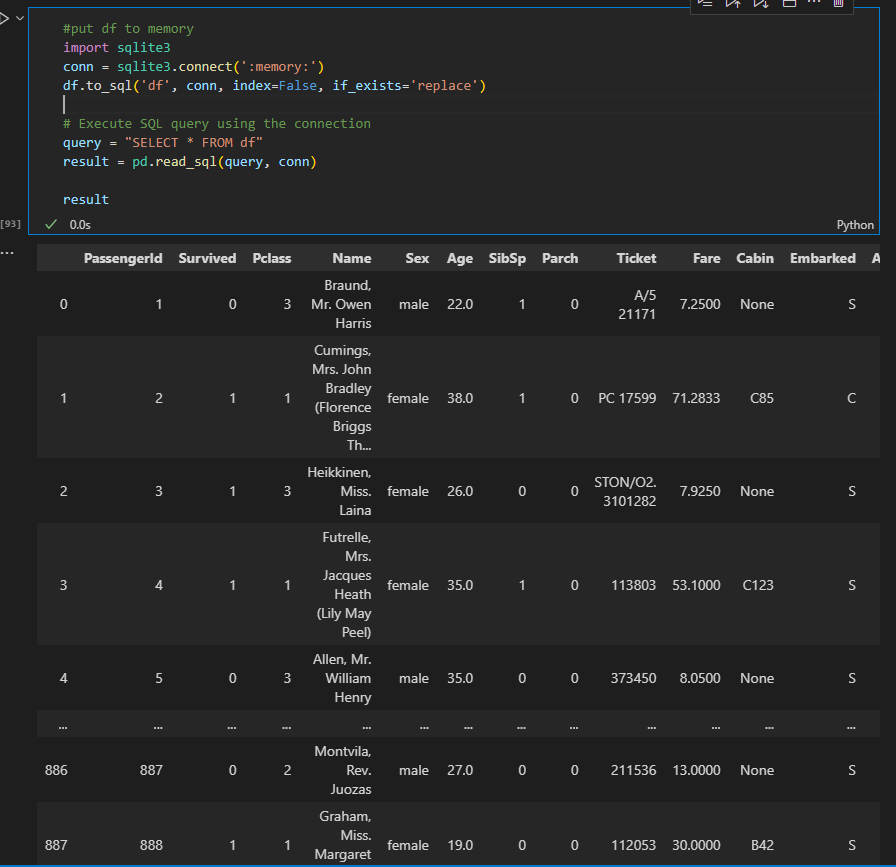
df.to\_sql('df', conn, index=False, if\_exists='replace')

# 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**

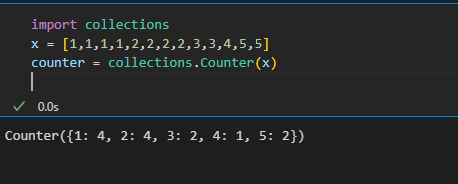
df[(df['Sex'] == 'female') | (df['Age'] <= 15)]

**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')*

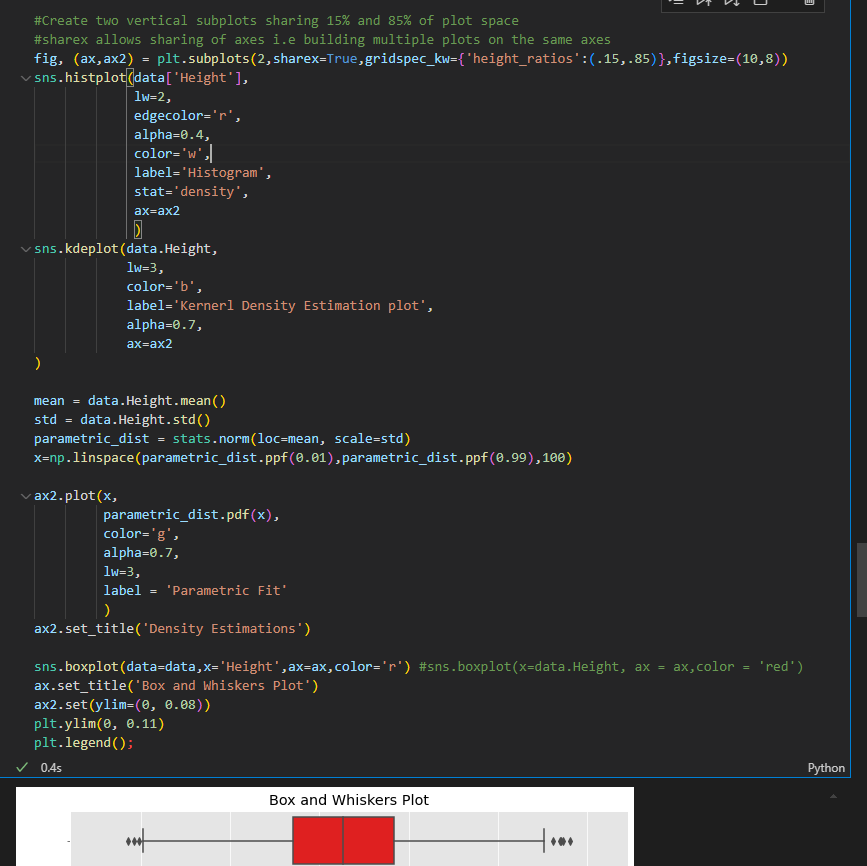
*sns.boxplot(data=data,x='Height',ax=ax,color='r') #sns.boxplot(x=data.Height, ax = ax,color = 'red')*

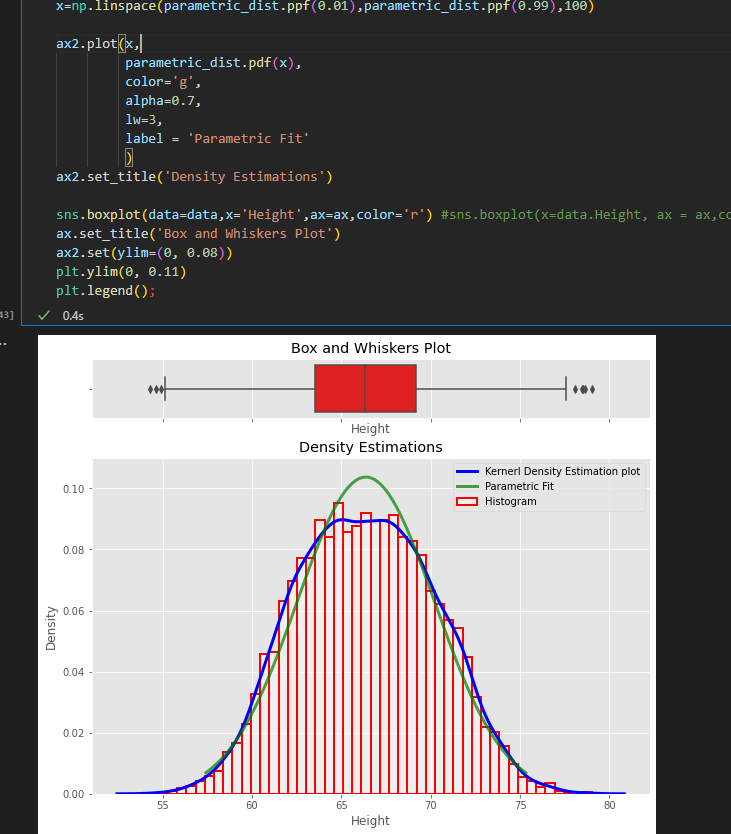
*ax.set\_title('Box and Whiskers Plot')*

*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]

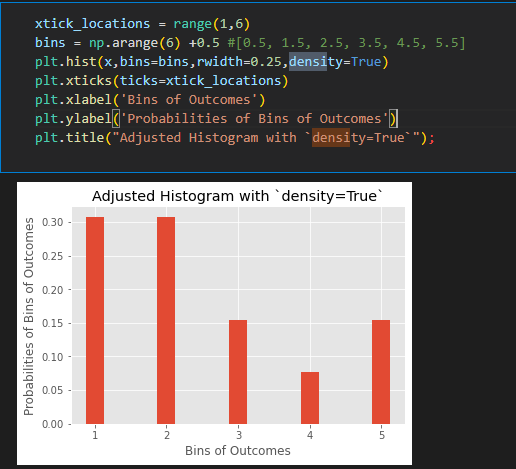
plt.hist(x,bins=bins,rwidth=0.25,density=True)

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))

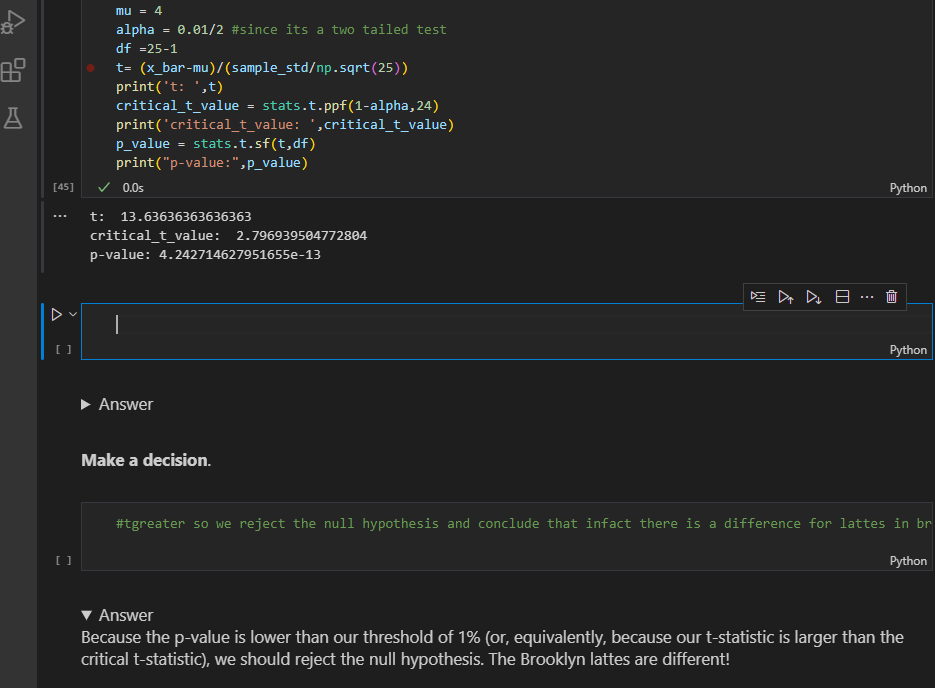
print('t: ',t)

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))

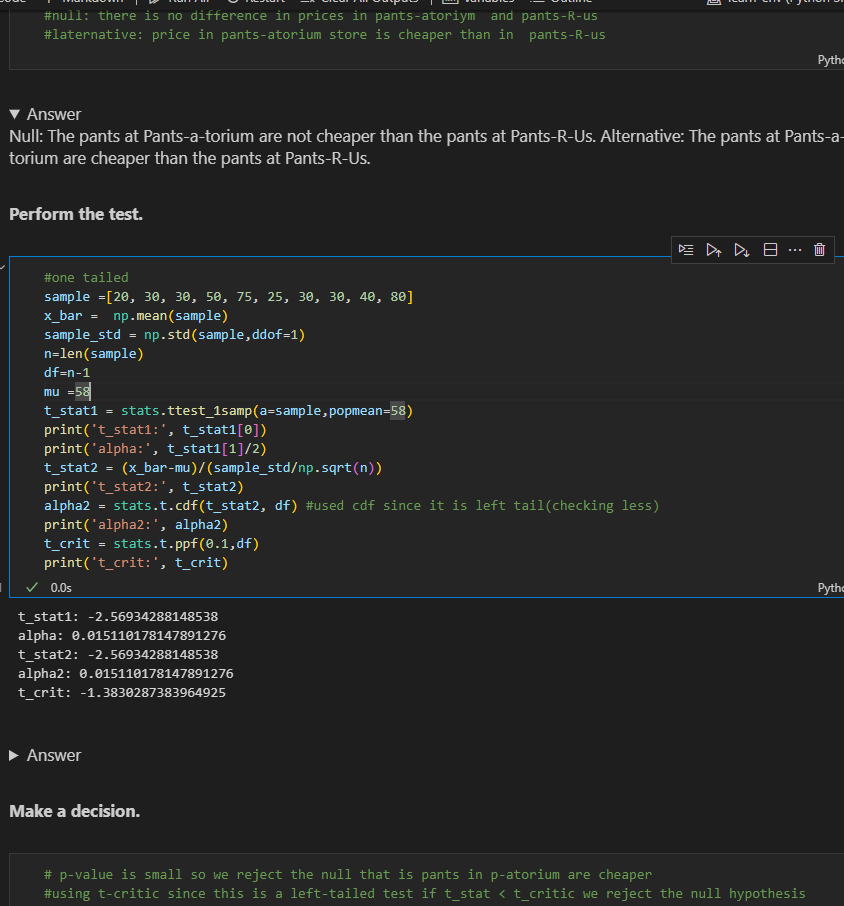
print('t\_stat2:', t\_stat2)

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**

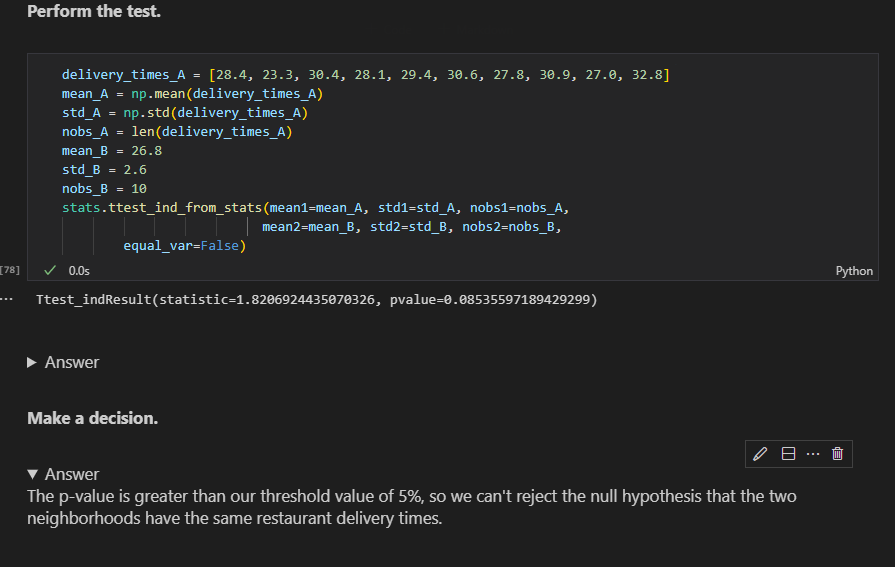
**std\_B = 2.6**

**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)**

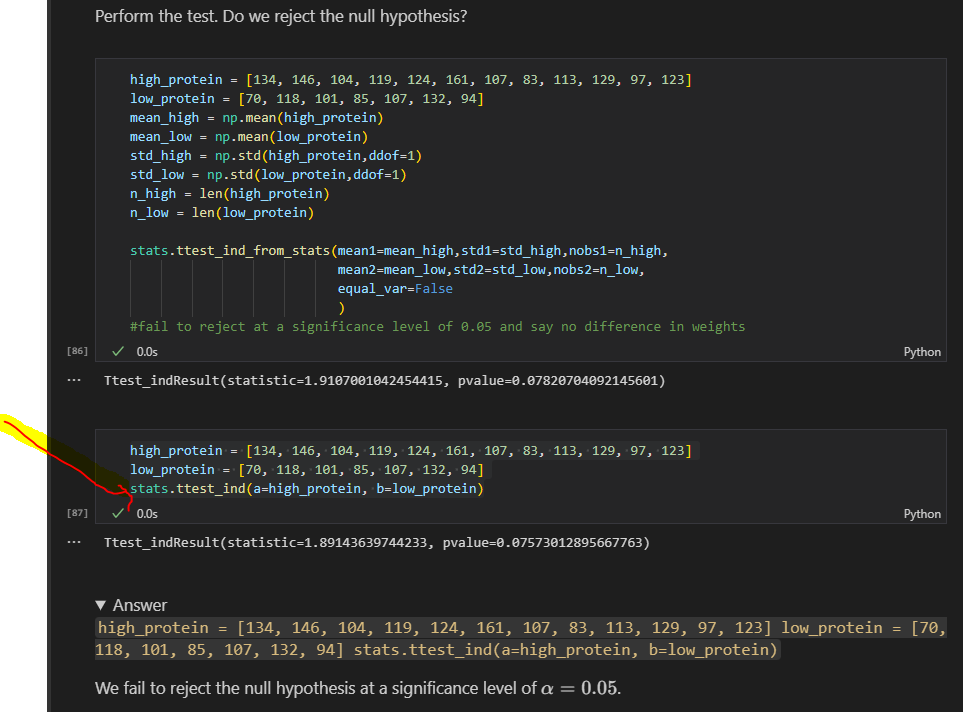


**c)2 sample again two tailed**

**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

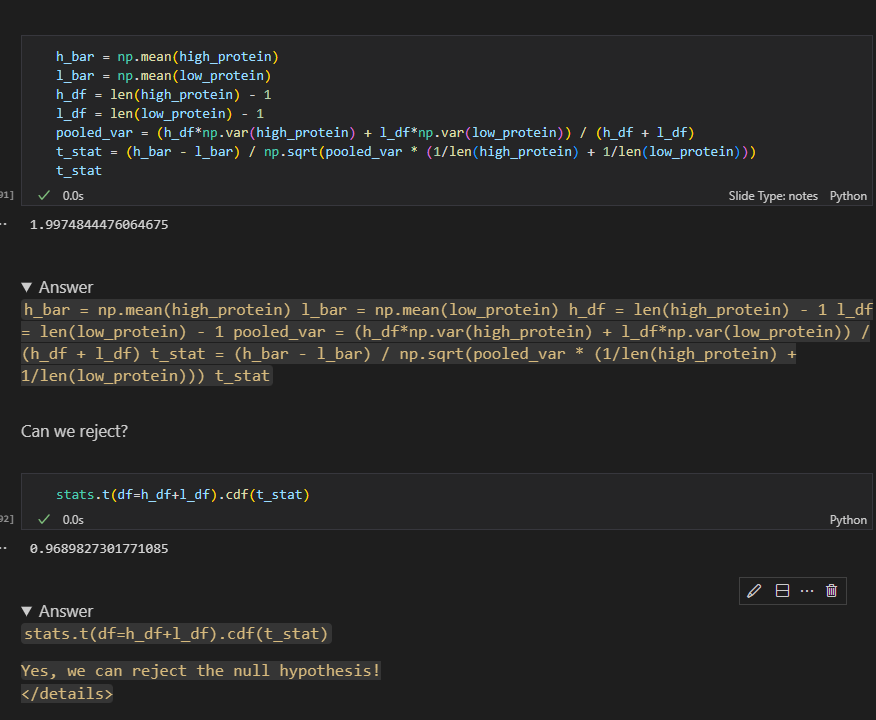
l\_df = len(low\_protein) - 1

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)))

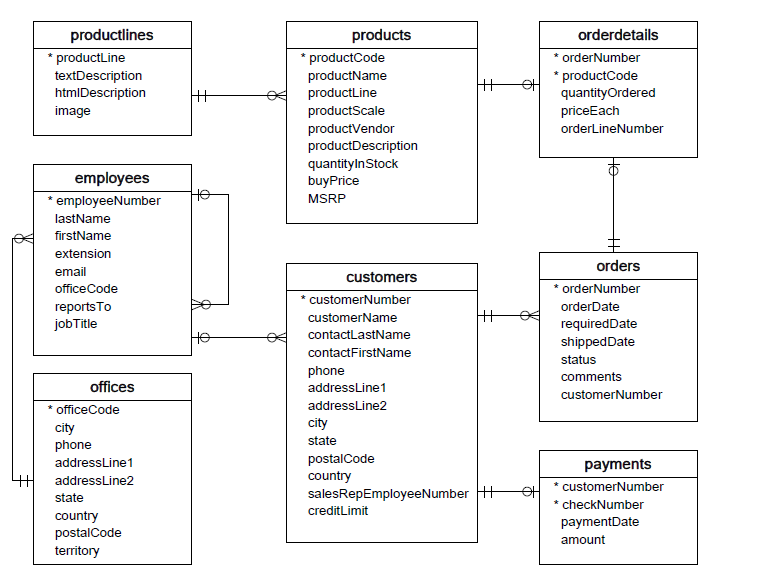
t\_stat

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**

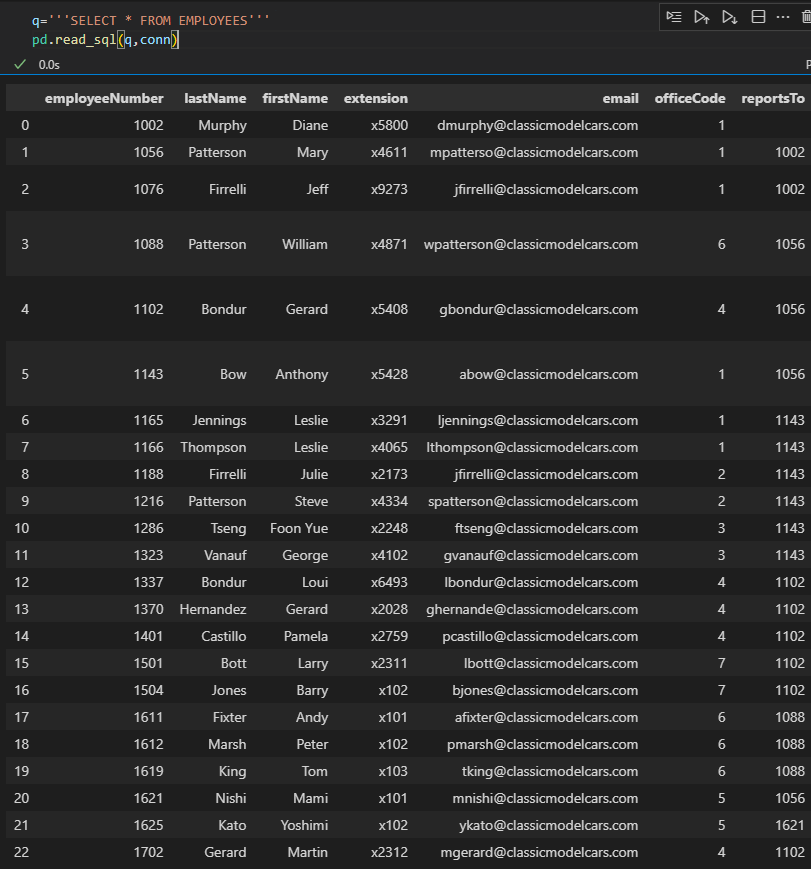
import sqlite3

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

1. **Select Records**

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

pd.read\_sql(q,conn)



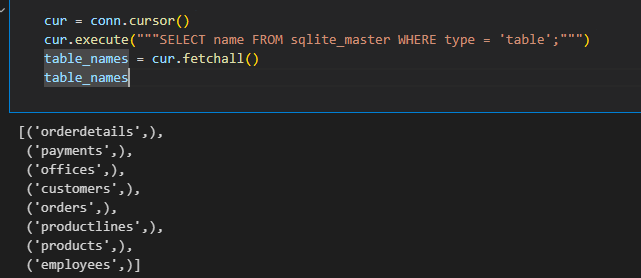
1. Check table in a database(Note that the .execute() method didn't actually return our data. The data is now just available in our cursor object. We'll use the .fetchall() method to get all the rows from our query.)

cur = conn.cursor()

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

table\_names = cur.fetchall()

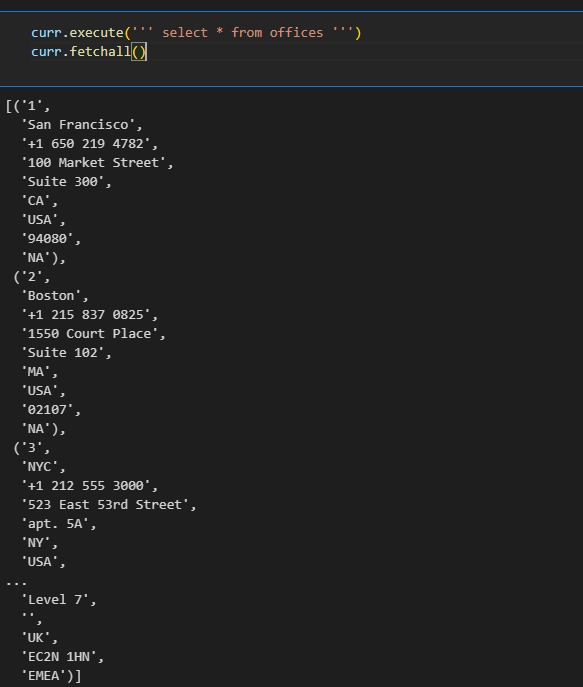
table\_names



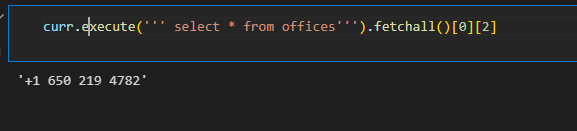
1. Select \* using cursor or can use ‘curr.execute(''' select \* from offices''').fetchall(),

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

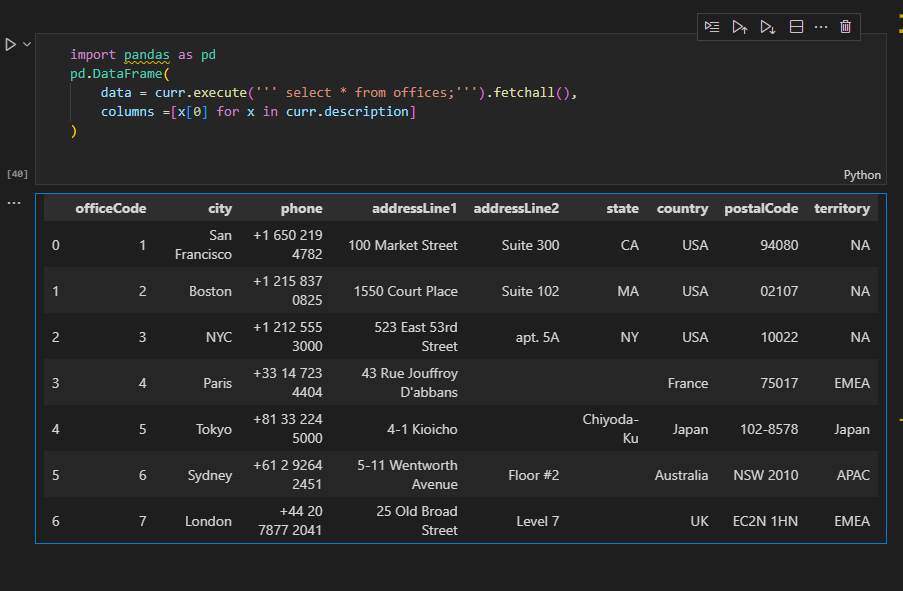
import pandas as pd

pd.DataFrame(

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

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

)



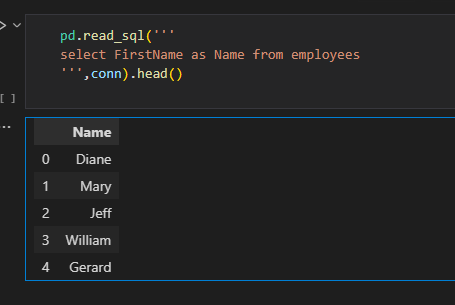
1. Retrieving a subset of columns

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)

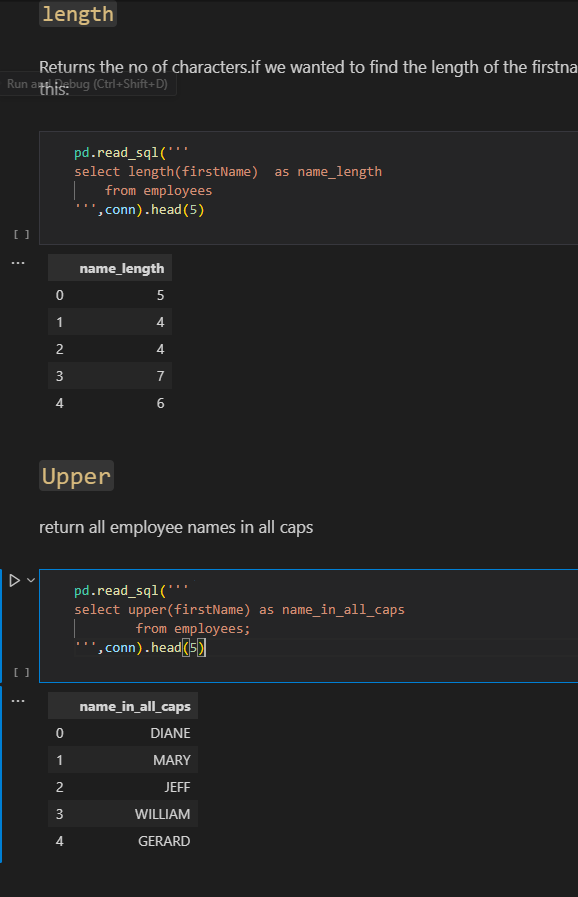
b)**Convert to upper**

pd.read\_sql('''

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

        from employees;

''',conn).head(5)



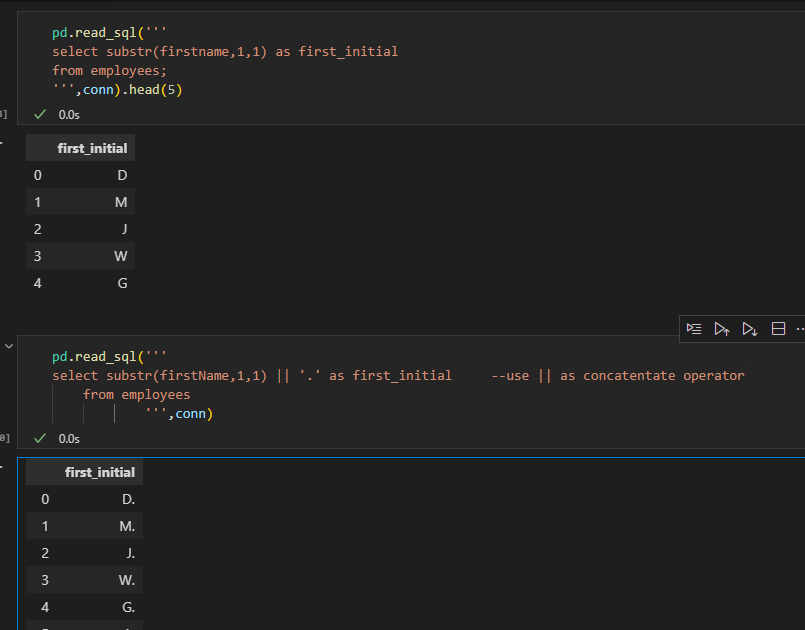
**c)Slicing**

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

from employees;

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

    from employees



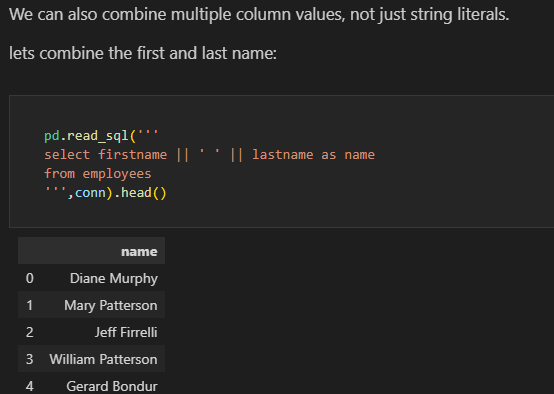
d) We can also **combine multiple column values**, not just string literals

pd.read\_sql('''

select firstname || ' ' || lastname as name

from employees

''',conn).head()



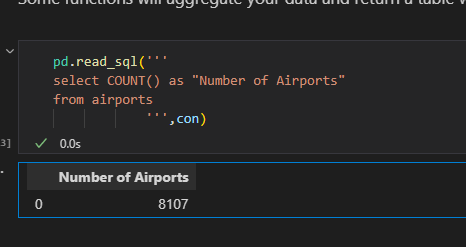
e)**Aggregations eg count**

pd.read\_sql('''

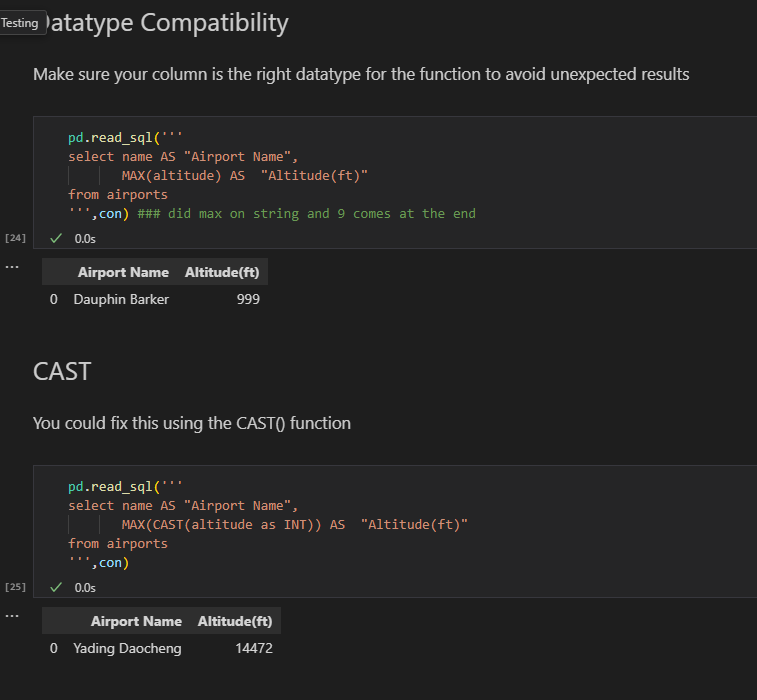
select COUNT() as "Number of Airports"

from airports

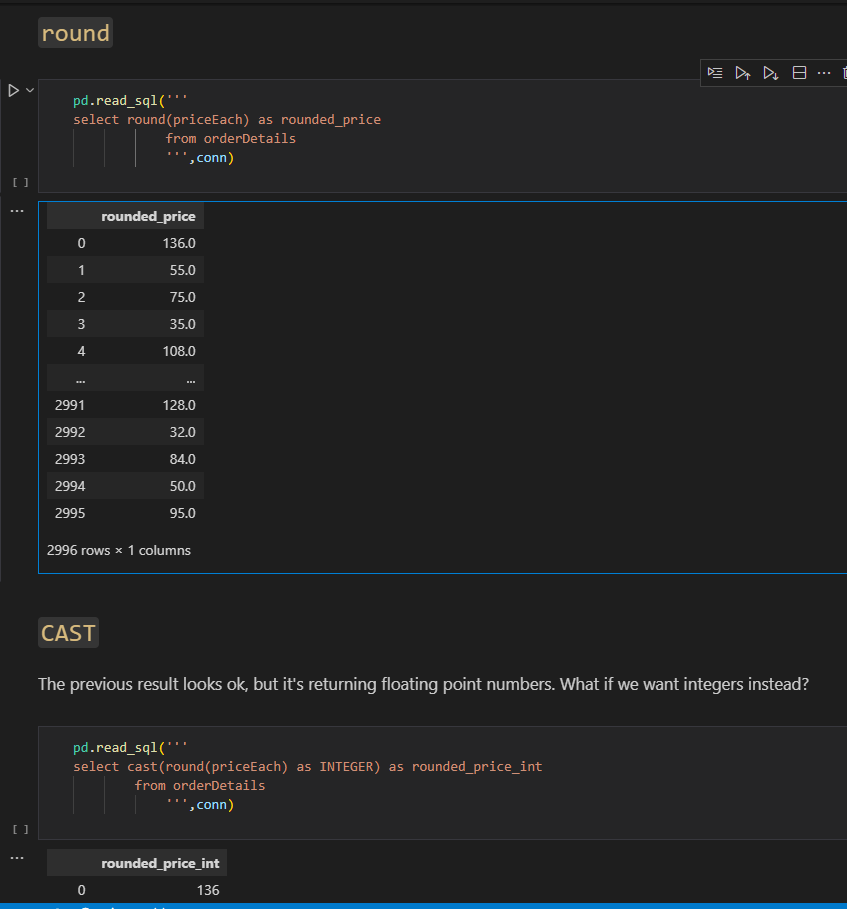
            ''',con)



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

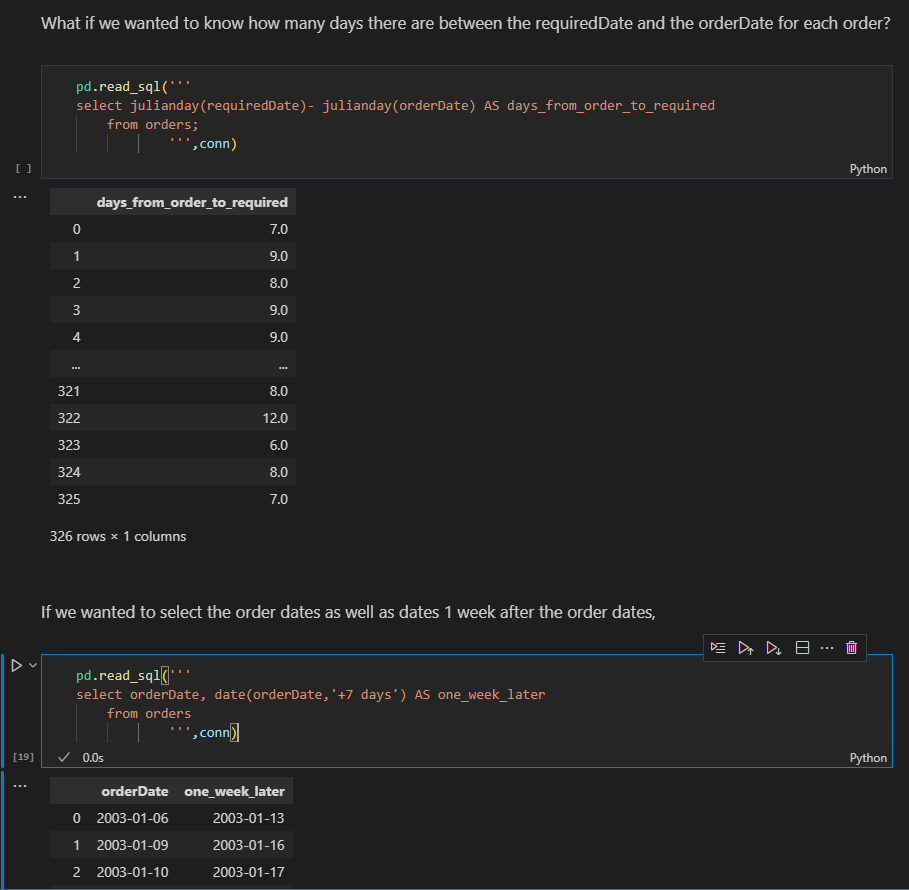


1. **Built in SQL for Math Functions**



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

(JulianDay and Date)



1. strftime function

pd.read\_sql('''

select orderDate,

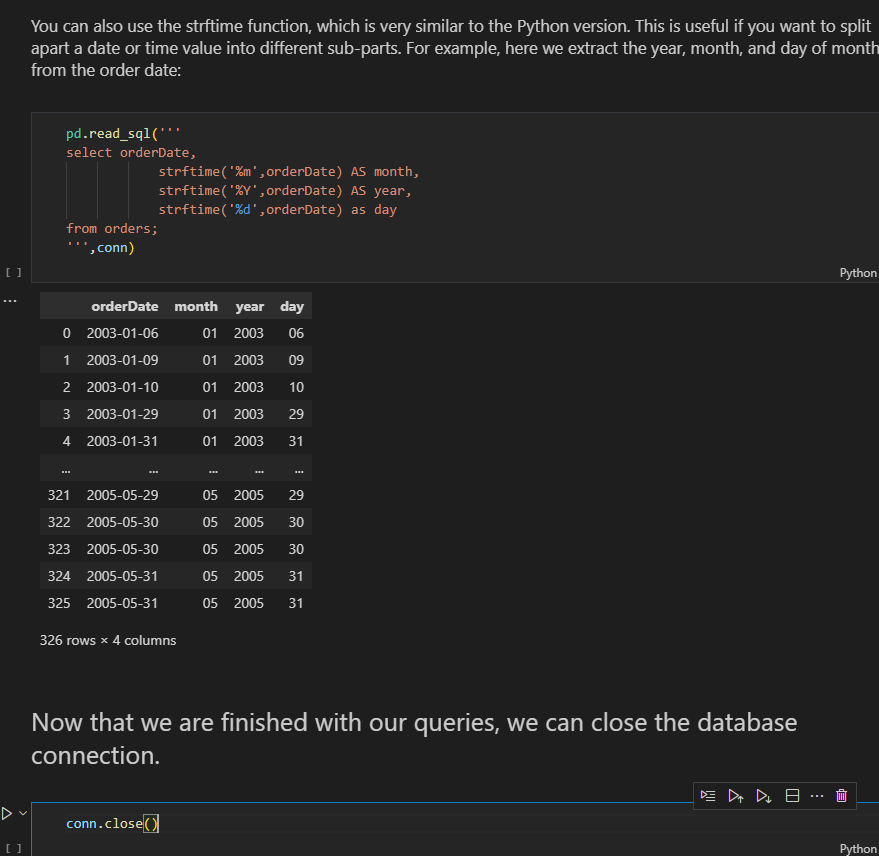
            strftime('%m',orderDate) AS month,

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

            strftime('%d',orderDate) as day

from orders;

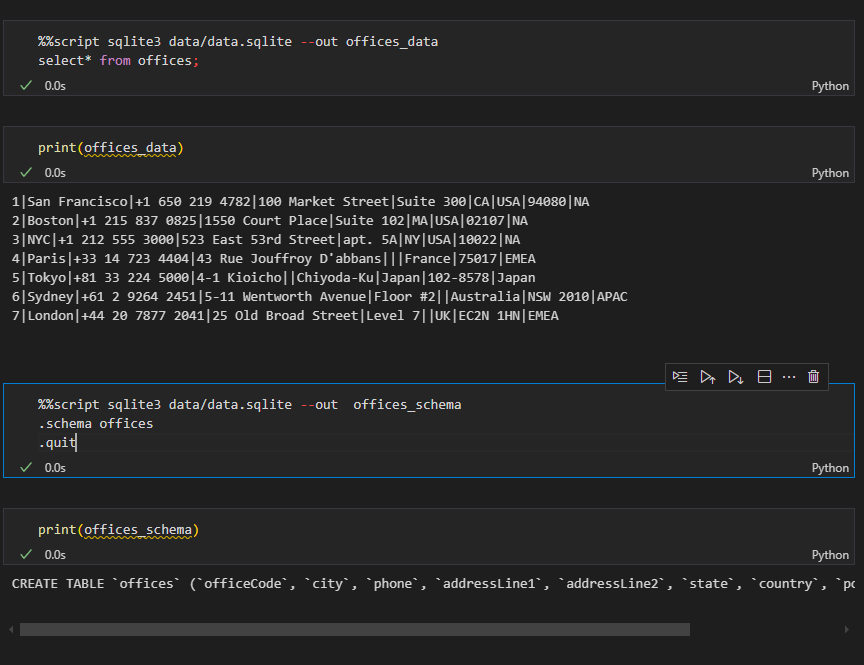
''',conn)



**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

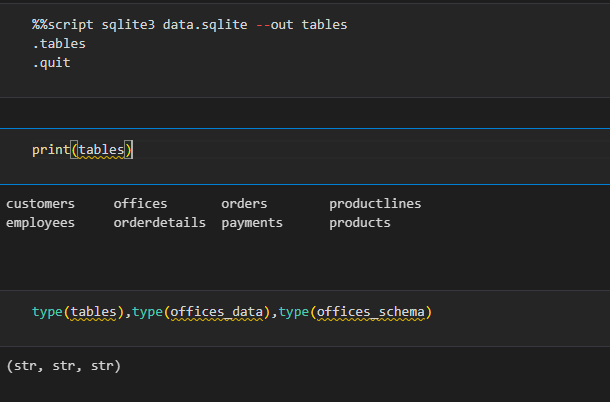
select\* from offices;



%%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

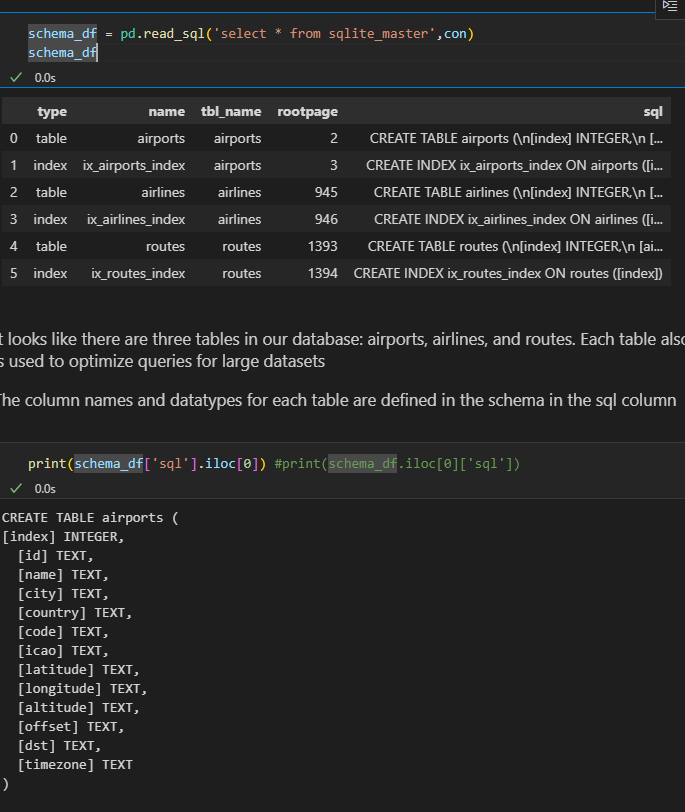
Database **Schema** – defines the structure of the database, including the tables and relationships between tables

**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",

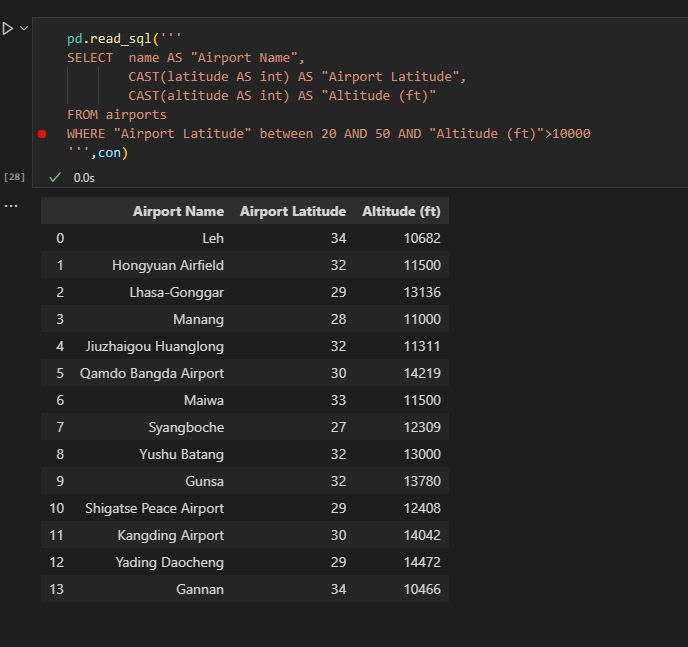
        CAST(latitude AS int) AS "Airport Latitude",

        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('''

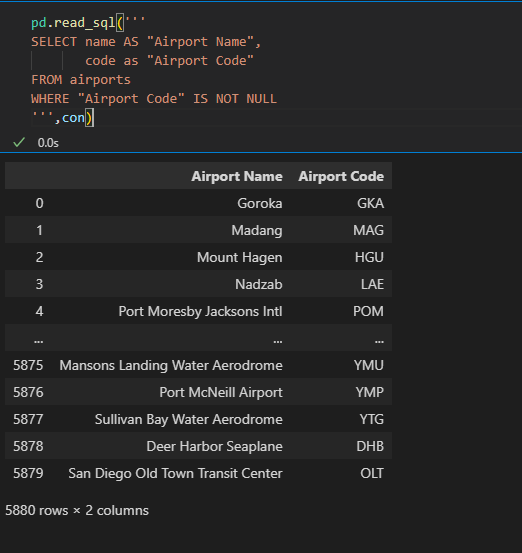
SELECT name AS "Airport Name",

       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)"

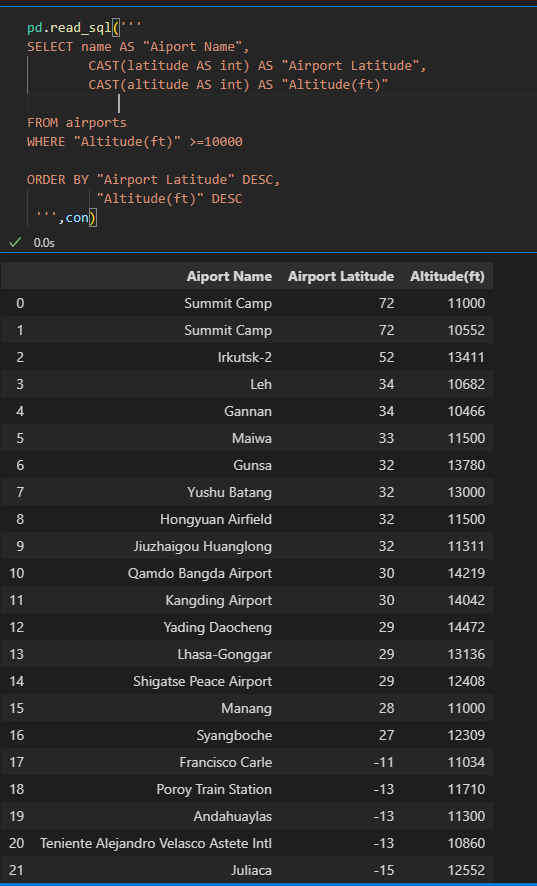
FROM airports

WHERE "Altitude(ft)" >=10000

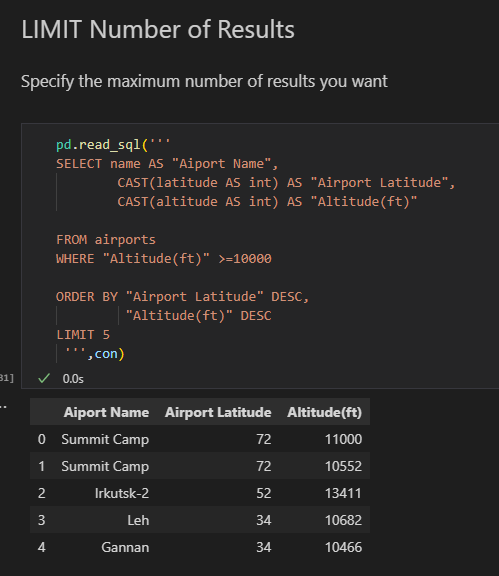
ORDER BY "Airport Latitude" DESC,

         "Altitude(ft)" DESC

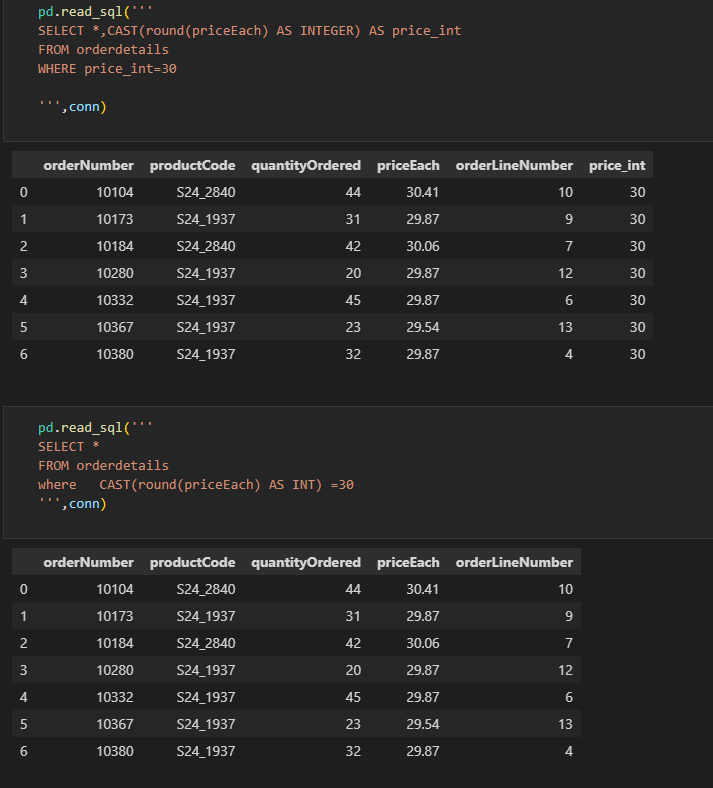
 ''',con)



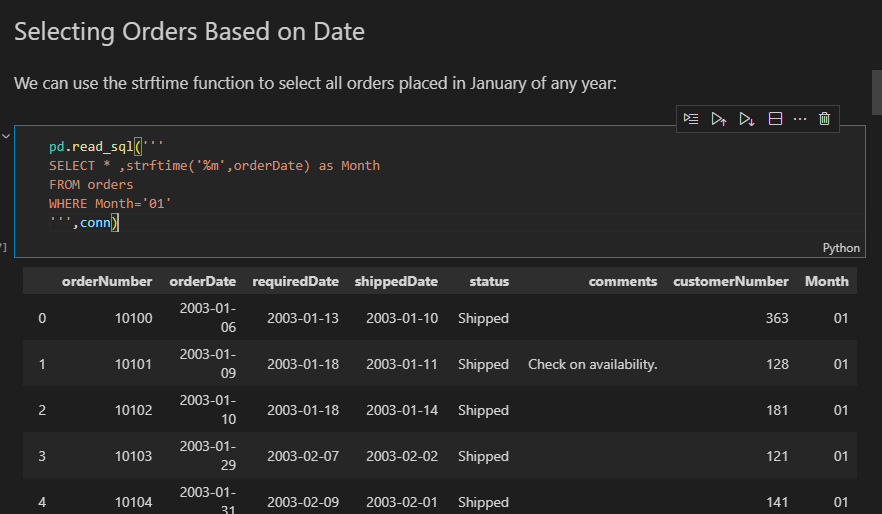
1. **LIMIT**



1. **a)Filtering by price**



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



**C)Filter late orders**

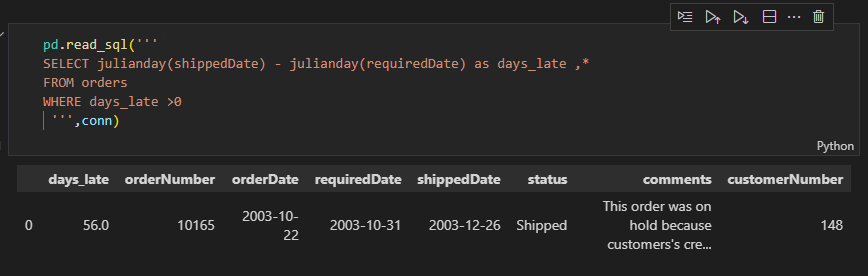
pd.read\_sql('''

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

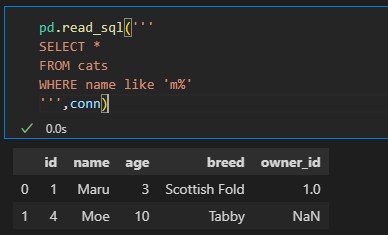
FROM orders

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\_\_**

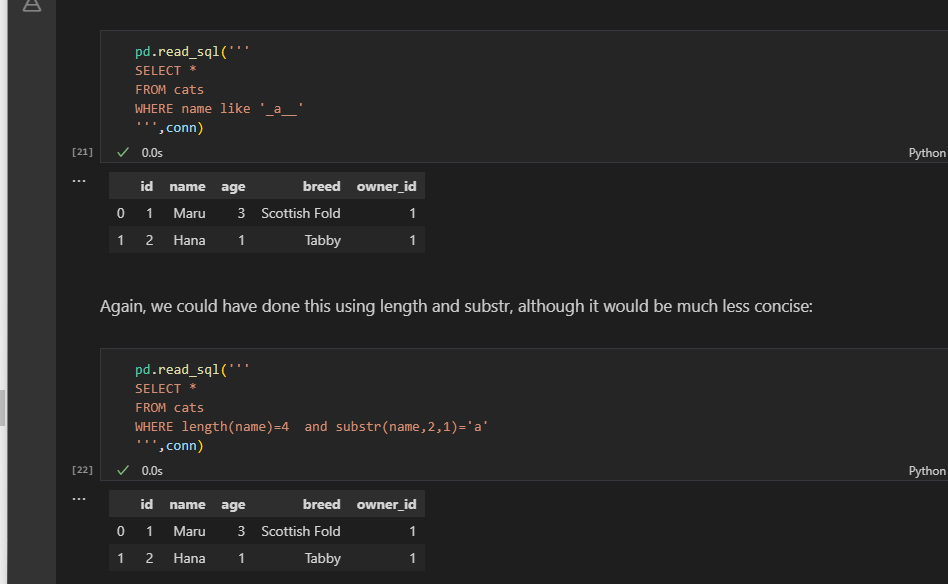
pd.read\_sql('''

SELECT \*

FROM cats

WHERE name like '\_a\_\_'

''',conn)



1. Select from 2 tables

pd.read\_sql('''

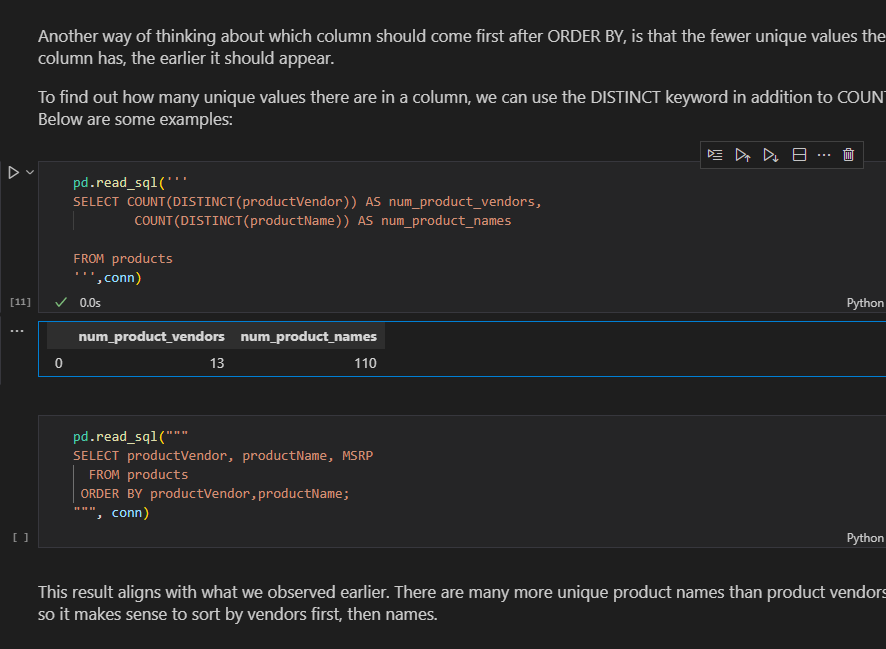
select cats.name,dogs.name

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('''

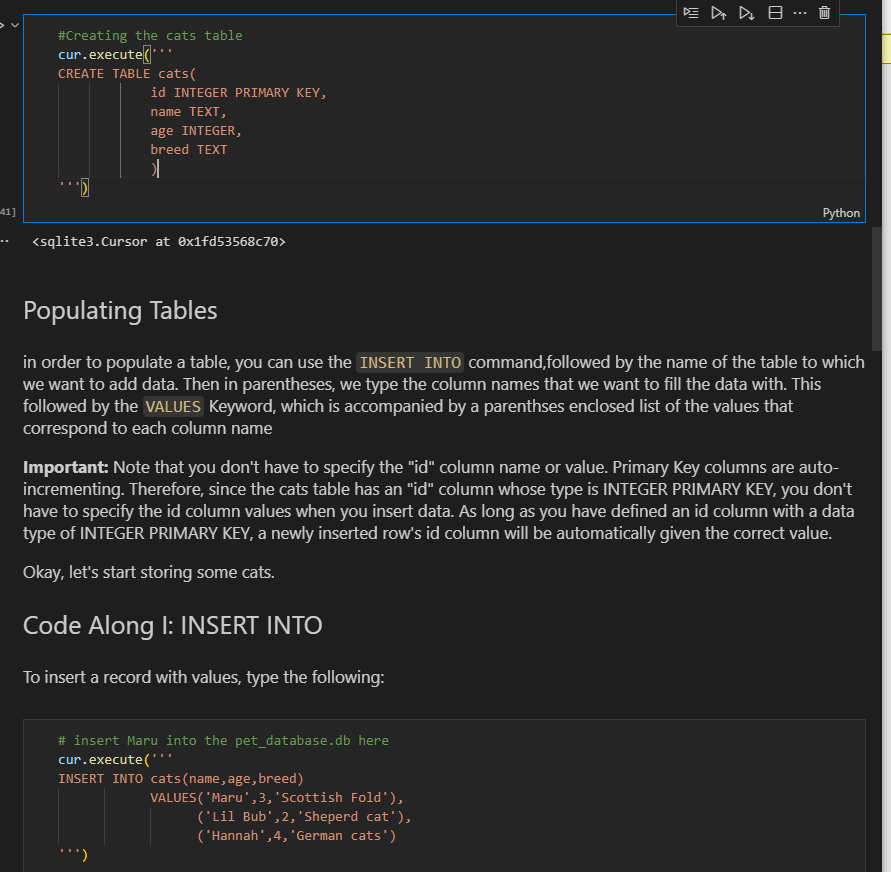
INSERT INTO cats(name,age,breed)

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

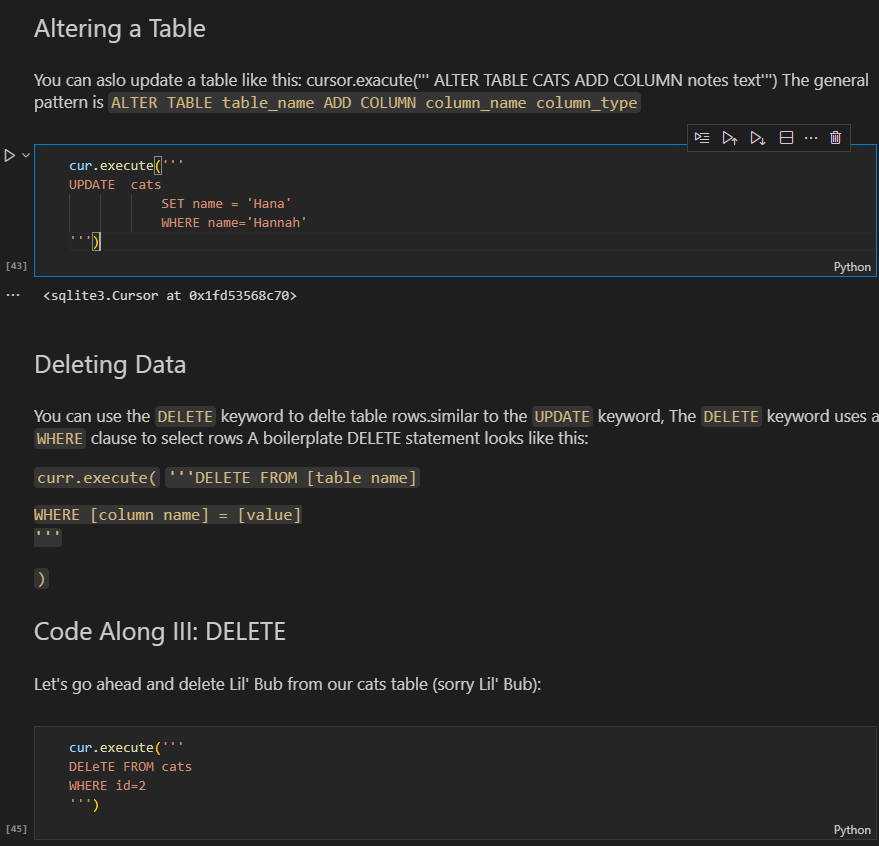
                  ('Lil Bub',2,'Sheperd cat'),

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

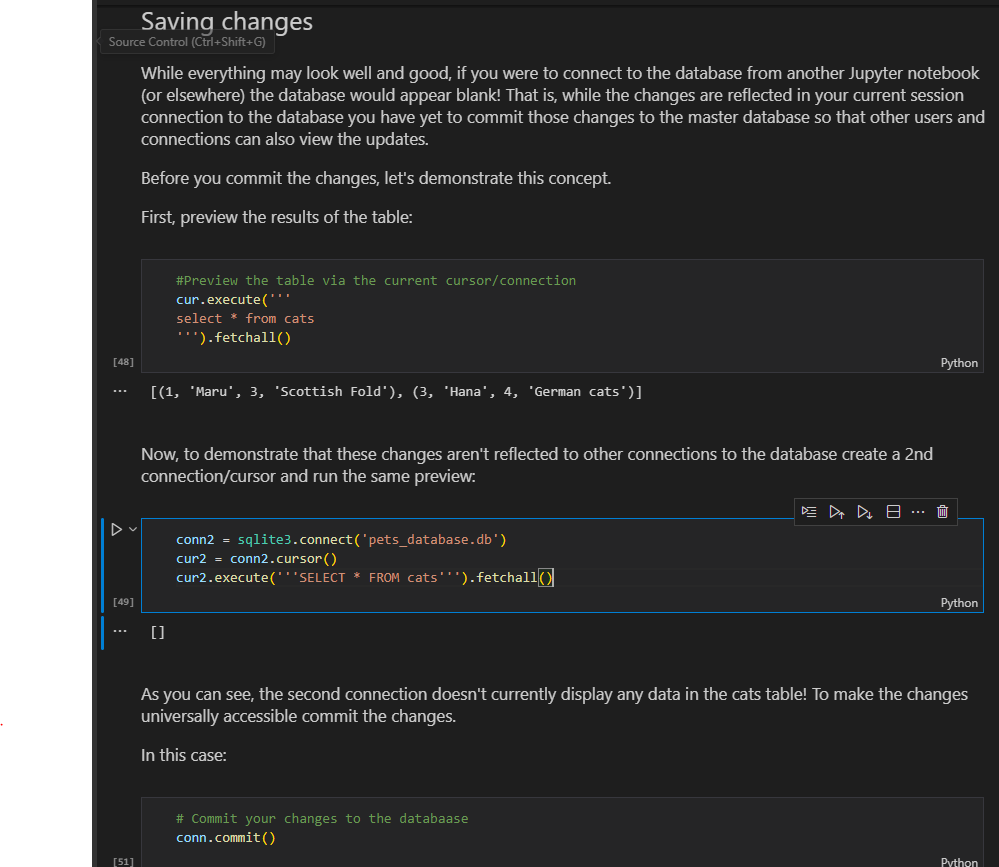
''')



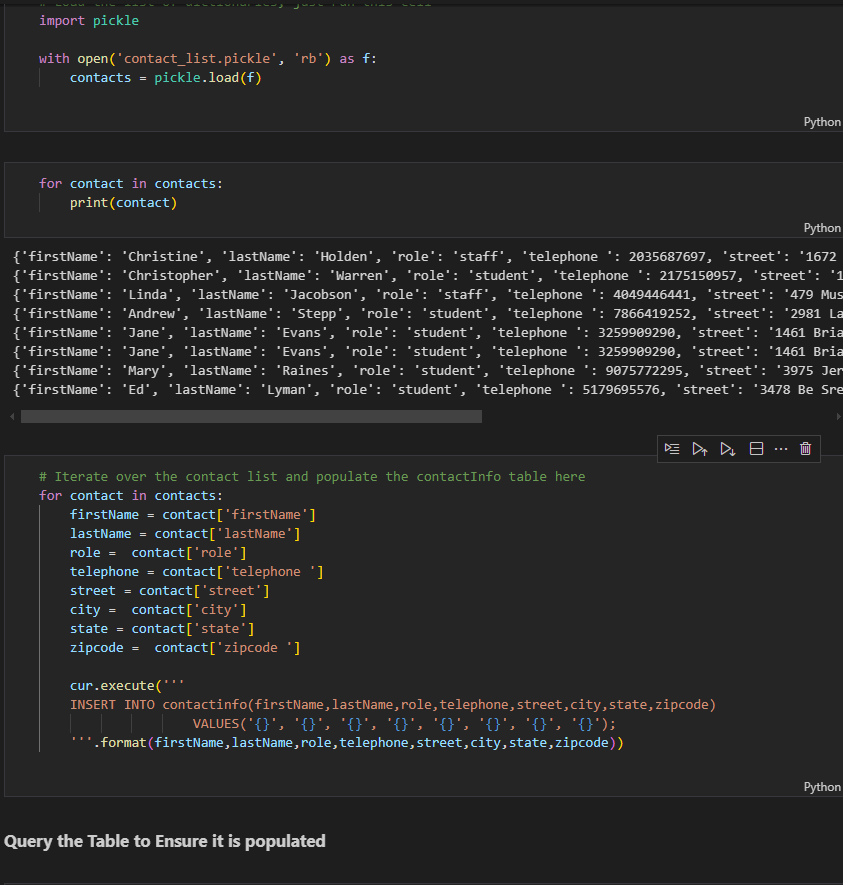
1. Altering(update) and deleting data



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,

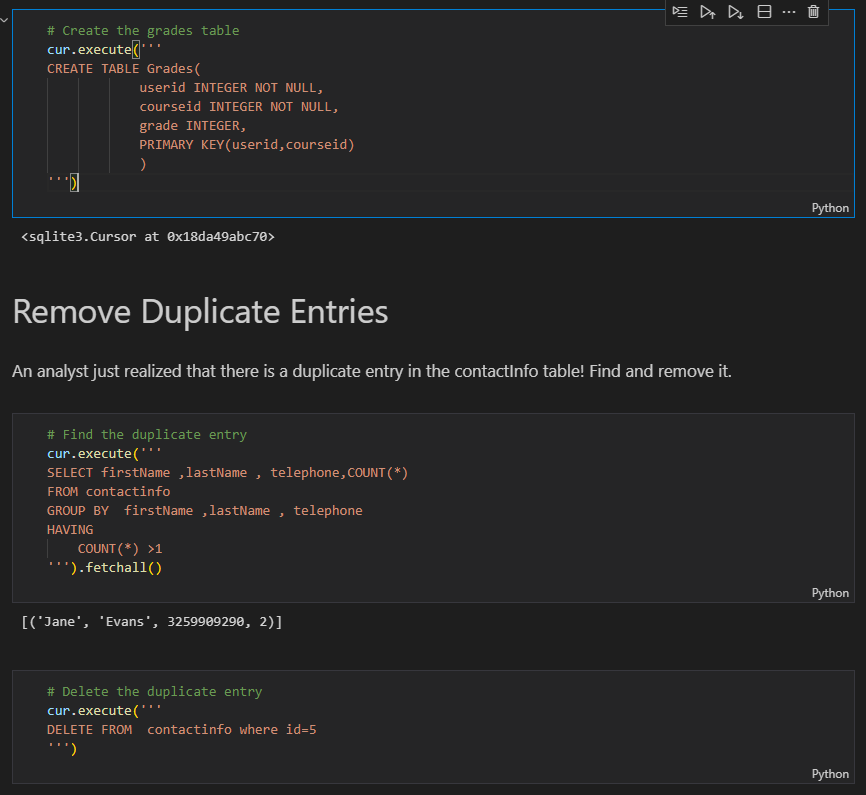
            courseid INTEGER NOT NULL,

            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

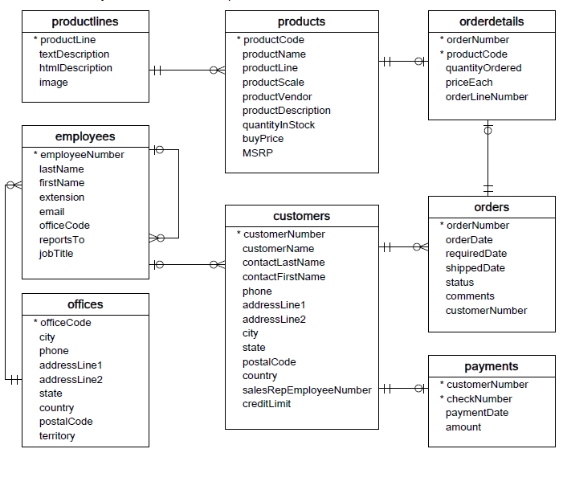
USING (productCode)

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

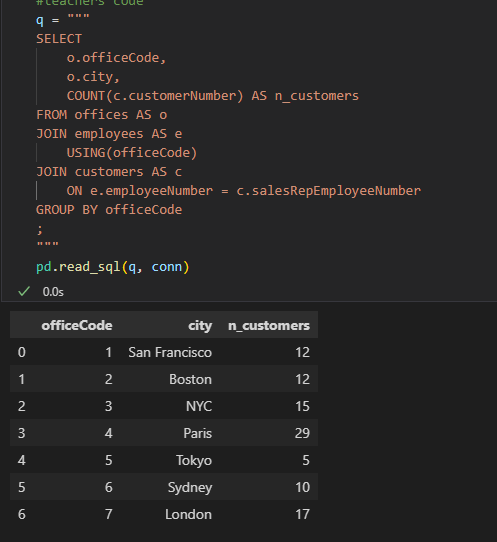
    ON e.employeeNumber = c.salesRepEmployeeNumber

GROUP BY officeCode

;

"""

pd.read\_sql(q, conn)



1. **Cast** for numeric purposes

q='''

SELECT

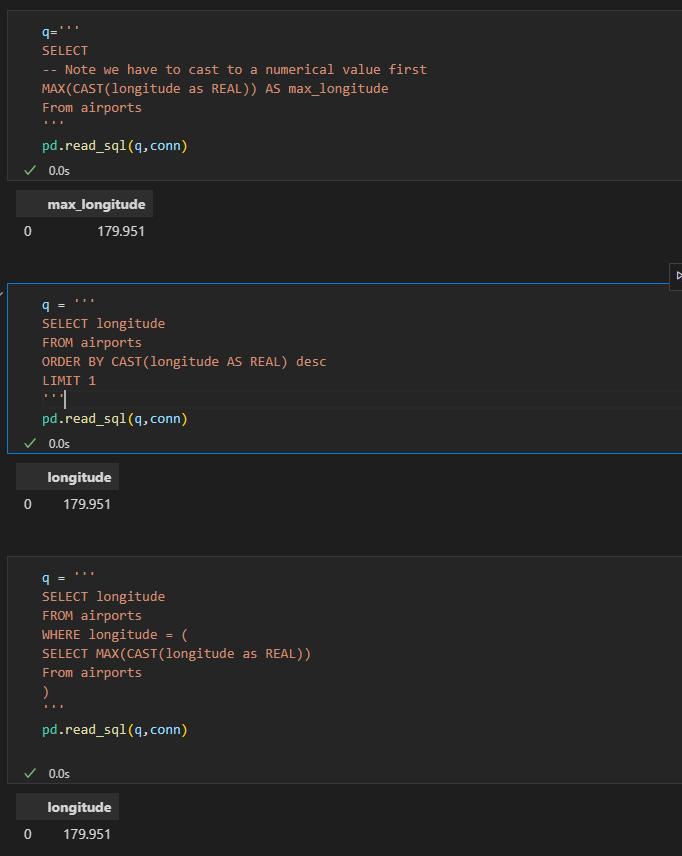
-- Note we have to cast to a numerical value first

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

