7 2019 Exams

In 2019, a joint Pandas/SQL five week course was taught which had four exams. These are all four exams. Note that each exam contained both Pandas and SQL questions.

Exam #1

The following table contains information about athletes at a college. You can assume each name uniquely defines a person and that a person only plays a single sport. There are no two rows with the same name.

- **name:** The name of the applicant (string)
- **wgt:** The athlete's weight (in kg) (float)
- **hgt:** The athlete's height (in meters) (float)
- **state:** The state that the athlete is from (string)
- **mdt:** The date that the measurement was taken (date type)
- **sport:** The sport (all lowercase) that the person plays (string)
- **sex:** Is the athlete male or female ("M" or "F") (string)
- **injury:** Injuries (all lowercase) are listed here (if Null that means no injury). There will be only a SINGLE injury listed (string)
- The name of the table / DataFrame is **ath**. No need to use a schema or load the DataFrame.
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.

Figure D.3: *Ath* Table: 12,435 Rows

name	wgt	hgt	state	mdt	sport	sex	injury
Ringly Roberson	94.25	1.75	NY	8-1-2012	basketball	M	
Crash Bandicoot	88.25	1.62	MS	1-1-2012	rugby	M	shoulder
alligator reynolds	66.1	1.88	PA	8-5-2012	softball	F	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Using SQL, write a query which returns the names (name only) of the top-6 tallest athletes who play basketball.

```
SELECT
        name
FROM
        ath
where sport = 'basketball'
order by hgt desc limit 6;
```

2. Using SQL, write a query which returns all basketball players (name only) shorter than 1.65 m from either New York ('NY') or Alabama ('AL'). Only include those athletes who are *not* injured.

```
select name from ath where
        state in ('NY', 'AL')
        and hgt < 1.65
        and sport = 'basketball'
        and injury is null;
```

3. Write an SQL query which returns the number of athletes from each state who *are* injured. This should be two columns by fifty rows.

```
SELECT
        state, count(1) as ct
from
        ath
where injury is not null
group by 1;
```

4. Write an SQL query which returns all rows and columns for female athletes.

```
select * from ath where sex = 'F';
```

5. We say that a sport is injury prone if 10% or more of the sport has injuries. Write an SQL query which returns a list of sports which are injury prone.

```
select sport from ath
group by 1
having sum( case when injury is not null then 1 else 0 end) ::float / sum(1) >= .10;
```

OR:

```
select sport from
(select sport
       , sum( case when injury is not null then 1 else 0 end)::float / sum(1) as rat
from ath
group by 1 ) as innerQ
where rat >= .1
```

6. We calculate the BMI ("Body Mass Index") of a person by taking their weight and dividing it by the height *squared*. Write a query which returns all rows and three columns: BMI, name and sport.

```
select wgt / hgt / hgt as bmi, name, sport
from ath;
```

7. Return a table with three columns: name, sport, and BMIFlag. BMIFlag should be equal to "0" if the BMI is less than or equal to 20, "1" if the BMI is greater than 20 and less than or equal to 30 and "2" otherwise.

```
select
        name, sport, wgt / hgt / hgt as BMI
        , case
                when wgt / hgt / hgt <= 20 then 0
                when wgt / hgt / hgt <= 30 then 1
                else 2 end as bmiflag
from
        ath;
```

8. If a person's BMI is greater than or equal to 30 they are defined as obese. Write a query which returns the *percentage* of athletes of each sport who are obese.

```
select
        sport,
        sum( case when wgt / hgt / hgt > 30 then 1 else 0 end )::float / count(1) as pctObese
from
        ath
group by 1;
```

9. Write a query which returns the sport, average height and average weight (by sport) for everyone whose name begins with the letter "A". Do not assume that the first letter of the person's name is always uppercase (there could be an "ann mitchell" in the table).

```
select
        sport
        , avg( hgt) as agh
        , avg(wgt) as agw
from ath
where upper(left(name,1)) = 'A'
group by 1;
```

10. Volleyball and basketball are known to be hard on the knees. Write a query which returns the percent of athletes who have "knee" injuries who play "volleyball" or "basketball" (combined) vs. the percent of knee injuries for sports which are NOT "volleyball" or "basketball". In other words, this should return two rows (one for volleyball / basketball and one for other) and two columns (one with a label for the sports included and one for the percent).

```
select
        case
                when sport = 'basketball' or sport = 'volleyball' then 'VB'
                else 'not VB'
        end as sportsType
, sum( case when injury = 'knee' then 1 else 0 end)::float / sum(1) as pctKnee
from ath
group by 1;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *ath* is already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Using Pandas, return an object (Series, DataFrame or array) of the names, and names only, of the top-6 tallest athletes who play basketball.

```
ath.loc[(ath.sport == 'basketball')].nlargest(6, hgt)['name']
```

2. Using Pandas, return all basketball players (name only) shorter than 1.65 m from either New York ('NY') or Alabama ('AL'). Only include those athletes who are not injured.

```
ath.loc[
(ath.sport=='basketball') & (ath.hgt < 1.65) & ((ath.state == 'AL') | (ath.state == 'NY'))
& (ath.injury.isna() == True), 'name']
```

3. Using Pandas, return an object which contains the names of sports (this should be without duplicates) which have a player with an injury to their "shoulder". You may assume that all the injuries in the table are lowercase.

```
ath.loc[(ath.injury=='shoulder')].sports.unique()
```

4. Female soccer players who have had a knee injury are going to be put on a special training program. Please return an object which contains *their names and only their names* sorted by state (A to Z).

```
ath.loc[ (ath.sport == 'soccer' ) & (ath.injury == 'knee') & (ath.sex == 'F')]
        .sort_values('state')['name']
```

5. We calculate the BMI ("Body Mass Index") of a person by taking their weight and dividing it by the height squared. In Pandas, return a DataFrame with three columns: BMI, name and sport for all rows.

```
ath = ath.assign(bmi = ath.wgt / ath.hgt / ath.hgt).loc[:, ['name', 'sport', 'bmi']]
```

6. Return a DataFrame with three columns: name, sport and BMIFlag. BMIFlag should be equal to "0" if the BMI is less than or equal to 20, "1" if the BMI is greater than 20 and less than or equal to 30 and "2" otherwise.

```
ath = ath.assign(bmi=ath.wgt / ath.hgt / ath.hgt).loc[:, ['name', 'sport', 'bmi']]
ath.loc[ (ath.bmi >= 30), 'bmiflag'] = 2
ath.loc[ (ath.bmi >= 20) & (ath.bmi < 30), 'bmiflag'] = 1
ath.loc[ (ath.bmi < 20), 'bmiflag'] = 0
```

7. There was an error with the weight machine and all weight-ins done in the month of March of 2012 were 10% too high. Please return an updated DataFrame with the information corrected. Note that this should include all rows and columns from the original dataset with wgt set 10% lower for miss-measured observations.

```
ath.loc[(ath.mdt.dt.year == 2012) & (ath.mdt.dt.month == 3), 'wgt']
        = .9* ath.loc[(ath.mdt.dt.year == 2012) & (ath.mdt.dt.month == 3), 'wgt']
```

8. Return a DataFrame which contains all information on anyone from Rhode Island ("RI") or who has a "knee" injury. Return this data sorted first by sport (A to Z), then by state (A to Z) and then by name (*Z to A*). Finally, upper case all returned names.

```
ath = ath.loc[(ath.state == "RI") | (ath.injury == "knee")]
        .assign(name=ath.name.str.upper())
        .sort_values(['sport', 'state', 'name'], ascending=[True, True, False])
```

9. Return a DataFrame which contains name, state and a flag which is equal to 1 if they play soccer and weight less than 70 kg or play basketball and weight less than 80 kg. The flag should be zero otherwise.

```
ath = ath.assign(flag = 0)
ath.loc[((ath.sport == "soccer") & (ath.wgt < 70))
        | ((ath.sport == "basketball") & (ath.wgt < 80)), 'flag'] =1
ath[['name', 'state', 'flag']]
```

Exam #2

The following table contains information about customer service interactions at a company. In particular, this has information about customers coming in and asking questions about their computer laptops.

- **serviceid:** This is an incrementing integer (int)
- **custid:** This is the ID for the customer (int)
- **pid:** This is the ID of the reported problem (e.g. battery problems are when pid = 2) (int)
- **servicedt:** This is the date that the service took place (date)
- **location:** This is the city and state of the service center (string)
- **result:** This is the diagnosis code for the device (e.g. result = 1 means solved) (int)
- **followup:** This contains information about if there was a follow up to the customer service (string)
- The name of the table / DataFrame is **cust**. No need to use a schema or load the DataFrame.
- The only column with Null values is "followup".
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.

Figure D.4: *cust* Table: 12,435 Rows

serviceid	custid	pid	servicedt	location	result	followup
1	21	12	1-1-2012	Livermore, CA	1	
2	21	23	1-5-2014	Livermore, CA	1	
3	26	11	1-15-2018	Livermore, CA	110	Refund
4	53	18	3-3-2011	Yuma, AZ	1	

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns all rows and columns for services which occur in March or December of *any* year.

```
select
        *
from
        cust
where date_part('month', servicedt ) = 3
        or date_part('month', servicedt) = 12;
```

2. All customers with problem 22 (pid = 22) in California (location ends with 'CA') had the wrong result. Please write a query which returns a list of customers (no duplicates, just their IDs) who need to be notified.

```
select distinct custid
from cust
where right(location,2) = 'CA'
and pid = 22;
```

3. Write a query which returns a time series of the number of services which have *no* followup. This should be aggregated to the month/year level, so that all no-followup service of the same month/year are combined. Make sure to return the data sorted from earliest to latest date. In other words, there should be two columns: one indicating the year / month (as a date) and one with the number of services without a followup.

```
select
        date_trunc('month', servicedt) as monyear
        , sum(1) as ct
from
        cust
where followup is null
group by 1
order by 1 asc;
```

4. We say that a service request is solved if there is no followup *and* the result is equal to 1. For each problem type (pid), report the total number of solved service requests.

```
select
        pid
        , count(1) as solved
from
        cust
where followup is null and result = 1
group by 1;
```

5. We say that a service request is solved if there is no followup *and* the result is equal to 1. Generate a dataset which has one row per location and three columns. The first column is location, the second is the *total* number of solved interactions at that location (over all time) and the third is the total number of all customer service interactions in August of 2014 at that location.

```
select
        location
        , sum( case when result = 1 and followup is null then 1 else 0 end) as tst2
        , sum( case when date_trunc('month', servicedt)::date = '08-01-2014'
                then 1 else 0 end ) as tstaug
from
        cust
group by 1;
```

6. Write a query which returns the locations which have more than 10 different types of problems (unique pid). For those locations, return two columns: one with the original location and one *with just the state abbreviation*. You can assume that all locations are of the form "cityname, state abbreviation" and that all state abbreviations are TWO characters long.

```
select
        location, right(location, 2) as state
from
        cust
group by 1
having count(distinct pid) > 10
```

NOTE THE ABOVE CAN BE GROUP BY 1 OR GROUP BY 1,2

7. We are trying to figure out how effective each service center is at solving different problems. Create a dataset with 3 columns: the first should be location, the second should be problem (pid) and the third should be the *percent* of problems of that type and at that location which are "solved" (result = 1 and no followup). Make sure to exclude any problem/location group with less than or equal to 10 rows.

```
select
        pid
        , location
        , sum( case when result = 1 and followup is null
                then 1 else 0 end)::float / sum(1) as pct
from
        cust
group by 1,2
having count(1) > 10;
```

8. Write a query which returns the frequency distribution of different problem types. This should return two columns. The first, called num, should be the number of times a problem appears in the dataset and the second, called "val" should be number of times this frequency occurs. For example, let's say that problems (pid) 27, 35 and 115 each appear 8 times in the table (and no other problem appears exactly 8 times in the table), then there should be a row which is (8,3). Note that each problem (pid) should only be tallied once.

```
select ct as num, count(1) as val
from
        (select count(1) as ct
        from cust group by pid) as innerq
group by 1;
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrame named *cust* is already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Generate a DataFrame which contains all columns and only those rows which have problem #3 (pid = 3) and are solved (result = 1 and there is no followup).

```
cust.loc[ (cust.result == 1 ) & (cust.pid == 3) & (cust.followup.isna())]
```

2. Generate a dataset which contains (a) location (b) the earliest date that a service occurred for that location, (c) the latest date that a service occurred at that location, (d) the number of unique problem's (pid) that the location experienced and (d) the total number of services that occurred at that location. Don't worry about column names, but make sure that location is a *column*.

```
(cust
        .groupby(['location'])
        .agg( {'servicedt' : ['max', 'min'], 'pid' : ['nunique', 'sum']})
        .reset_index()
)
```

3. Generate a DataFrame with three columns: location, day of the week ("dow", as an integer) and the number of rows with *any, non-null* followup that were at that location on that day-of-the-week. Note that it doesn't matter if this returns columns or indexes for any value.

```
cust['dow'] = cust.servicedt.dt.dayofweek
cust['flag'] = 0
cust.loc[ ~(cust.followup.isna), 'flag'] = 1
cust.groupby(['location', 'dow']).agg({ 'flag' : 'sum'})
```

4. Generate a dataset which has one row per location and three columns. The first column is location, the second is the *total* number of solved (result = 1 and followup is empty) interactions at that location (over all time) and the third is the total number of customer service interactions in August of 2014 at that location. Make sure that this is three *columns*.

```
cust.assign(succ=0, Aug2014=0)
cust.loc[(cust.result == 1) & (cust.followup.isna()), 'succ'] =  1
cust.loc[(cust.servicedt.dt.month == 8) & (cust.servicedt.dt.year == 2014), 'Aug2014'] = 1
cust.groupby('location').agg({'succ' : ['sum'], 'Aug2014' : ['sum']}).reset_index()
```

5. We are trying to figure out how effective each service center is at each location at solving different problems. Create a DataFrame with three *columns*: the first should be location ("location"), the second should be problem ("pid") and the third ("succ") should be equal to 1 or 0, depending on if the outcome was solved (result = 1 and no followup) or not. This should have the same number of rows as the original DataFrame. Name this DataFrame "tst".

```
cust['succ'] = 0
cust.loc[(cust.result == 1) & (cust.followup.isna()), 'succ'] = 1
tst = cust.loc[ :, ['location', 'pid', 'succ']]
```

6. Assume that you have the "tst" DataFrame from the problem above. We now want to calculate

the percent of customer interactions which are solved, aggregated to the problem (pid) and location level. In other words, using the DataFrame from the previous problem, generate a new DataFrame consisting of three *columns*: location, pid and the percent of interactions which were solved. Specifically this is the sum of "succ" divided by the number of rows. Make sure to remove any row which has less than 10 observations in the original DataFrame (as there is not enough data to conclude anything from them).

```
tst = tst.groupby(['location', 'pid']).agg({'succ' : ['sum', 'count']})
tst.columns = ['s1', 'c1']
tst = tst.loc[ (tst.c1 > 10) ]
tst.assign(pct=tst.s1/tst.c1)[['pct']].reset_index()
```

Exam #3

The following table contains information about doctors and their patients. At most, each patient has one doctor.

- Columns in the patients table
  - **patientid:** This is an auto-incrementing integer for the patient (int)
  - **doctorid:** This is the ID for the doctor that they see (int)
  - **hgt:** This the height of the patient in meters (float)
  - **birthdt:** This is the date of birth of the patient (date)
  - **wgt:** This is the weight of the patient in kg (float)
  - **city:** This the city that the person lives in (string)
  - **state:** This is the state that the person lives in (string)
  - **sex:** The sex of the patient (M/F) (string)
- Columns in the doctors table
  - **doctorid:** This is an auto-incrementing integer for the doctor (int)
  - **speciality:** This is the type of doctor (string)
  - **surgeon:** Is the doctor a surgeon (Y/N) (string)
  - **sex:** The sex of the doctor (M/F) (string)
- The names of each table are "patients" and "doctors". No need to refer to any schema or load a DataFrame.
- **There are some patients who have not yet been assigned doctors, so doctorid could be Null in the patients table.**
- **There are some doctors who were just hired who have not been assigned patients yet.**
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.
- Any two columns with the same name can be assumed to match.

Figure D.5: *Patients* Table: 12,435 Rows

patientid	doctorid	hgt	birthdt	wgt	city	state	sex
1	2	1.7	1-1-1997	58.2	Livermore	CA	M
2	18	1.65	1-5-1975	56.5	Livermore	CA	F
3	18	1.8	1-15-1994	67.3	Livermore	CA	M
4	7	1.93	3-3-1964	66.0	Yuma	AZ	M

Figure D.6: *Doctor* Table: 277 Rows

doctorid	speciality	surgeon	sex
1	Oncology	Y	M
2	ENT	N	F
3	General	N	M
4	Pediatric	Y	F

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns two columns. The first should be the doctorid and the second should be the number of patients that that doctor sees. This should be sorted from most to least patients seen. Make sure to *not* include doctors who have no patients and patients who have no doctors.

```
select
        doctorid, count(1) as ct
        patients
where doctorid is not null
group by 1
order by 2 desc;
```

2. A high-usage doctor is one that sees strictly more than 50 patients. Write a query which returns four columns. The first should be state, the second and third should be the average height and average weight of patients from that state and the fourth should be the number of patients from that state. Note that this should *only* include patients who have a high-usage doctor. There should be one row returned per state.

```
select
        state
        , avg( wgt) as aw
        , avg(hgt) as ah
        , count(1) as ct
from
        (select doctorid from patients
        group by doctorid
        having count(1) > 50) as lhs
left join
        patients
using(doctorid)
group by 1 ;
```

OR

```
select
        state
        , avg( wgt) as aw
        , avg(hgt) as ah
        , count(1) as ct
from
        patients
where doctorid in
        (select doctorid from patients
                group by doctorid
                having count(1) > 50)
group by 1 ;
```

3. Write a query which returns three columns. The first should be the speciality, the second should be the number of surgeons (surgeon = 'Y') within that speciality and the third should be the number of non-surgeons (surgeon = 'N') of that speciality.

```
select
        speciality
        , sum( case when surgeon = 'Y' then 1 else 0 end) as numsurg
        , sum( case when surgeon = 'N' then 1 else 0 end) as numnonsurg
from
        doctor
group by 1;
```

4. We say that a doctor is the same-sex as their patient if they are the same-sex as the patient (e.g. both Male or both Female). Write a query which returns two columns. The first should be the patientid and the second should be a flag which is equal to 1 if the patient and doctor are the same sex, zero otherwise. If a patient does not yet have a doctor the flag should be set to -1. This should have the same number of rows as the patients table.

```
select
        patientid
        , case
                when lhs.sex = rhs.sex then 1
                when rhs.sex is null then -1
                else 0
        end as flag
from
        patients as lhs
left join
        doctor as rhs
using(doctorid);
```

5. Create a dataset with five columns: year, speciality, number of patients who were born that year and have a doctor with that speciality, the average weight of patients who were born that year and have a doctor of that speciality and the average height of patients who were born that year and have a doctor of that speciality. Note that year should be returned as a number, not a date. Only include those patients who have been assigned doctors.

```
select
        date_part('year', birthdt) as yr
        , speciality
        , count(1) as numpatients
        , avg( hgt ) as avghgt
        , avg( wgt) as avgwgt
from
        patients
inner join
        doctor
using(doctorid)
group by 1,2;
```

6. We calculate the Body Mass Index of a person (BMI) as weight divided by height *squared*. Generate a table which has two columns: speciality and the max BMI of patients who see doctors of that speciality. Be careful to exclude doctors without patients and patients without doctors.

```
select
        speciality
        , max( wgt / hgt / hgt ) as maxbmi
from
        patients
inner join
        doctor
using(doctorid)
group by 1;
```

7. Write a query which returns four columns. The first should be speciality, the second should be sex (of the doctor), the third should be surgeon (the 'Y'/'N' flag) and the fourth should be the number of doctors of that type (where type is defined as speciality, sex and surgeon combination). If there is no doctor of that type then the count should be set to zero.

```
select lhs1.sex, lhs2.speciality, lhs3.surgeon, count(rhs.doctorid)
from
        (select distinct sex from doctor) as lhs1
cross join
        (select distinct speciality from doctor) as lhs2
cross join
        (select distinct surgeon from doctor) as lhs3
left join
        doctor as rhs
on lhs1.sex = rhs.sex
        and lhs2.speciality = rhs.speciality
        and lhs3.surgeon = rhs.surgeon
group by 1,2,3
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that a DataFrames named *patients* and *doctor* are already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Generate a DataFrame with two columns. The first should be the doctorid and the second should be the number of patients that the doctor sees. This should be sorted from most to least patients seen. Make sure to *not* include doctors who have no patients and that it returns two *columns*.

```
(pd.merge(doctor, patients, on='doctorid', how='inner')
 .groupby( 'doctorid' )
 .agg({'patientid' : 'count'})
 .sort_values('patientid', ascending=False)
 .reset_index())

 This can be done without a merge, but you need to filter out the rows which don't match
```

2. Return a DataFrame which contains patientid, doctorid, birthdate and the speciality of the that patient's doctor. Only include those patients from California "CA" who have been assigned doctors.

```
lhs = patients.loc[ (patients.state == 'CA'), ['patientid', 'doctorid', 'birthdate'] ]
rhs = doctors.loc[ :, ['doctorid' , 'mtype']]
mrg = pd.merge( lhs, rhs, on='doctorid', how='inner')
```

3. A high-usage doctor is one that sees strictly more than 50 patients. Create a DataFrame which returns four *columns*. The first should be state, the second and third should be the average height and average weight of patients from that state and the fourth should be the number of patients from that state. Note that this should *only* include patients who have a high-usage doctor. There should be one row returned per state.

```
p1 = patients[['doctorid']].groupby('doctorid').agg({'doctorid' : ['count']}).reset_index()
p1.columns = ['doctorid', 'ct']
p1 = p1.loc[(p1.ct > 50)]

mrg = (pd.merge(p1, patients, on = ['doctorid'], how = 'inner')
        .groupby('state')
        .agg({'wgt' : ['mean'], 'hgt' : ['mean'], 'doctorid' : ['count']})
        .reset_index()
        )
```

OR

```
lst = patients[['doctorid']].groupby('doctorid').agg({'doctorid' : ['count']}).reset_index()
lst.columns = ['doctorid', 'ct']
lst = lst.loc[(lst.ct > 50), 'doctorid']

p1 = (patients.loc[(patients.doctorid.isin(lst)), :]
        .groupby('state')
        .agg({'wgt' : ['mean'], 'hgt' : ['mean'], 'doctorid' : ['count']})
        .reset_index()
        )
```

4. We calculate the Body Mass Index of a person (BMI) as weight divided by height *squared*. Generate a DataFrame which has two columns: speciality and the max BMI of patients who see doctors of that speciality. Be careful to exclude doctors without patients and patients without doctors. Make sure to return two *columns*

```
patients['bmi'] = patients.wgt / patients.hgt / patients.hgt

mrg = (pd.merge( patients, doctor, on='doctorid', how = 'inner')
       .groupby('speciality')
       .agg({'bmi' : ['max']})
       .reset_index()
      )
```

5. Create a DataFrame with three *columns*. The first should be state, the second should be number of patients from that state (total), the third should be the number of patients from that state which do NOT have doctors.

```
mrg = pd.merge( patients, doctor, on ='doctorid', how='left').assign(nodoc=0)
mrg.loc[ mrg.doctorid.isna(), 'nodoc'] = 1
mrg = mrg.groupby('state').agg({'doctorid' : ['count'], 'nodoc' : ['sum']}).reset_index()
```

6. How many patients do not have a doctor assigned?

```
patients.loc[patients.doctorid.isna(), 'patientid'].count()
```

Exam #4

The following tables contains information about Uber drivers, their rides and reviews.

- Columns in the *drivers* table:
  - **did:** This is an auto-incrementing integer ID for the driver (int)
  - **state:** This is the state that the driver lives in (string)
  - **prom:** Is the driver on a promotion? (Y/N) (string)
- Columns in the *rides* table:
  - **rid:** This is an auto-incrementing integer for the ride (int)
  - **ridets:** This is the date and time that the ride occurred (date)
  - **did:** This is ID for the driver (int)

  - **air:** This is a flag (Y/N) for if the trip went to the airport (string)
  - **length:** This is the ride length in km (float)
- Columns in the *reviews* table:
  - **rid:** This is the ride which was reviewed (int)
  - **review:** This is the review (star scale: 1 to 5) (int)
- The names of the tables and DataFrames are *drivers, rides* and *reviews*.
- Assume that there are no Null values in any of the tables.
- Overly complex queries or code will be penalized.
- Only use syntax covered in class.
- Do not create any views.
- Any two columns with the same name can be assumed to match.
- **Not all rides will have reviews. A ride can have, at most, one review.**
- **Not all Drivers may have rides. When a driver first signs up they will not have any rides.**
- **DO NOT USE CTE ("with"), but you CAN use any analytic / window function.**

did	state	prom
1	CA	Y
2	MN	N
3	CA	N
4	CA	Y

rid	ridets	did	air	length
1	1-1-2012 10:24 AM	45	N	1.25
2	12-23-2012 12:22 PM	45	N	23.45
3	7-6-2013 4:13 AM	112	Y	11.17
4	5-5-2014 1:23 PM	1125	N	.75

rid	review
1	5
23	4
35	4
45	1

Figure D.7: *driver* table (27,777 rows), *rides* table (454,123 rows) and *reviews* table (137,145)

SQL Section

Please answer the following questions making sure to return *only* the information requested.

1. Write a query which returns two columns and a row for each state in the table. The first column should be the state and the second should be the number of rides completed by drivers from that state.

```
select
        state
        , count(1)
from
        drivers
join
        rides
using( did )
group by 1;
```

2. Write a query which returns two columns and one row per driver. The first column should be the driver's ID number (did) and the second should be a Y/N flag if the driver's first ride was to the airport or not.

```
select
        distinct did, airflag
from
        (select
                did
                , first_value( air ) over(partition by did order by ridets asc) as airflag
        from
                rides ) as innerQ;

Note that you could use an aggregate function in the outer query, but there has to be an
inner query.
```

3. Write a query which returns the following: state of the driver, total rides completed by drivers from that state and the average review of drivers from that state. Make sure to sort this from highest to lowest average review. Exclude any state with strictly less than 1,000 riders served. This should have one row per state.

```
select
        state
        , sum(1) as totalrides
        , avg( review ) as avgreview
from
        drivers
join
        rides
        using( did )
left join
        reviews
        using( rid )
group by 1
having count(distinct rid ) >= 1000
order by 3 desc;
```

4. Write a query which returns three columns. The first column should be year (as an integer), the second column should be total rides from that year and the third should be the *running or cumulative total of all rides, excluding the current year.* There should be one row per year.

```
select
       yr
       , numrides
       , sum(numrides) over(order by yr asc rows between unbounded preceding and 1 preceding) as cumsum
from
       (select
              date_part('year', ridets) as yr
              , sum(1) as numrides
       from
              rides
       group by 1 ) as iq;
```

5. Write a query which returns four columns: The first should be the driver id ('did'), the second should be the total number of rides, the third should be the number of their rides with a review and the fourth should be the number of rides with 5-star reviews (review = 5). Only include rides from 2018

and make sure to sort the drivers from most to least rides with reviews. Exclude drivers without any rides.

```
select
        did
        , count(1) as numrides
        , sum( case when review is not null then 1 else 0 end) as num_with_reviews
        , sum( case when review = 5 then 1 else 0 end) as five_star_reviews
from
        rides
left join
        reviews
using(rid)
where date_part('year', ridets) = 2018
group by 1
order by 3 desc;
```

6. Write a query which returns two rows and two columns. One row contains the phrase "LT10" and have the average review for rides that were (strictly) less than 10 km and the other row should be "MT15" and should be the average review for rides which are (strictly) more than 15 km. There should only be two rows.

```
select
        case
                when length < 10 then 'LT10'
                when length > 15 then 'MT15'
        end as flag
        , avg( review)
from
        rides
join
        reviews
using(rid)
where length < 10 or length > 15
group by 1;
```

7. *Without* using an analytic function, return one row and two columns. This should be the transpose of the data in the previous question. The first column should be the average review for rides of less than (strictly) 10 km and the second should be the average review for rides of more than (strictly) 15 km. It should only have one row.

```
select
        avg( case
                when length < 10 then review
                else null
        end ) as lt10
        , avg( case
                when length > 15 then review
                else null
        end        ) as mt15
from
        rides
join
        reviews
using(rid);
```

Pandas Section

Please answer the following question, making sure to return only the information required. You can assume that DataFrames named *drivers*, *rides* and *reviews* are already loaded. Unless otherwise specified you may return either a Pandas Series or DataFrame.

1. Return a DataFrame which has two columns and a row for each state. The first column should be the state and the second should be the number of rides completed by drivers from that state.

```
pd.merge( rides, drivers, on='did', how='left')
        .groupby('state')
        .agg({'rid' : ['count']})
```

2. We are interested in studying the effect of driver promotion (prom = 'Y') on long rides (strictly greater than 10 km). Return a dataset which contains two rows (one for prom='Y' and one for prom = 'N') and has three *columns*. The first should be prom, the second should the number of long rides to the airport (air='Y') the third should be the number of long rides *not* to the airport (air='N').

```
mrg = pd.merge( drivers, rides, on='did', how='left')
mrg = mrg.loc[(mrg.loc[:, 'length'] > 10), :]

mrg.loc[ : , 'airflag' ]  = 0
mrg.loc[ (mrg.loc[:, 'air']) =='Y' , 'airflag' ]  = 1

mrg.loc[ : , 'Nairflag' ]  = 0
mrg.loc[ mrg.loc[:, 'air']=='N' , 'Nairflag' ]  = 1

mrg = (mrg
        .groupby('prom')
        .agg( { 'airflag' : ['sum'], 'Nairflag' : ['sum']})
        .reset_index()
        )
```

3. Return a DataFrame with two rows and two *columns*. One row contain the phrase "LT10" and then has the average review for rides that were (strictly) less than 10 km and the other row should be

"MT15" and should be the average review for rides which are (strictly) more than 15 km. There should only be two rows.

```
mrg = pd.merge( rides, reviews, on='rid', how='inner')

mrg = (mrg
       .loc[(mrg.loc[:, 'length'] < 10) | (mrg.loc[:, 'length'] > 15), :]
      )

mrg.loc[:, 'flag'] = 'LT10'
mrg.loc[(mrg.loc[:, 'length'] >15), 'flag'] = 'MT15'

mrg.groupby('flag').agg({'review' : ['mean']}).reset_index()
```

4. There is a worry that there is a relationship between ride length and the percentage of rides with reviews. To analyze this we will create a flag called "lflag", which is equal to 1 if the ride length < 2 km, 2 if the length is $>= 2$ and < 5 km and 3 if the length $>= 5$ km. Create a dataset which has the following columns: lflag, state, number of *rides* which are of that flag-state combination, the number of those rides with reviews and the average review from rides of that lflag-state combination.

```
mrg = pd.merge( drivers, rides, on ='did', how='inner')
mrg = pd.merge( mrg, reviews, on = 'rid', how='left')

mrg.loc[:, 'lflag'] = 1
mrg.loc[(mrg.loc[:, 'length']) >=2) & (mrg.loc[:, 'length'] <5) , 'lflag'] = 2
mrg.loc[(mrg.loc[:, 'length'] >5)  , 'lflag'] = 3

mrg.groupby( ['state', 'lflag'])
        .agg({ 'review' : ['mean', 'count'], 'rid' : ['count'] })
        .reset_index()
```

5. What is the average review for all rides from drivers which have *ever* had a ride over 60km? This should return a single number.

```
lst = rides.loc[(rides.loc[:, 'length'] > 60), 'did'].drop_duplicates()

rds = rides.loc[rides.loc[:, 'did'].isin(lst), :]

pd.merge(rds, reviews, on='rid', how='inner')['review'].mean()
```