Parch & Posey Data Analysis with SQL & Pandas

Importing Required Libraries

```
import pandas as pd
import psycopg2
```

Connecting to Database

```
hostname= 'localhost'
port_id= 5432
database= 'Parch_and_Posey'
username= 'postgres'
pwd= 'Seriously?!!!'
```

```
try:
    conn = psycopg2.connect(host= hostname,
                            dbname = database,
                            user = username,
                            password = pwd,
                            port = port_id
                            )
except Exception as error:
    print(error)
```

Preparing the Pandas Dataframe

```
accounts_script = """select * from accounts"""
orders_script = """select * from orders"""
regions_script = """select * from region"""
salesPeople_script = """select * from sales_reps"""
```

```
accounts = pd.read_sql_query(accounts_script,con=conn)
orders = pd.read_sql_query(orders_script,con=conn)
regions = pd.read_sql_query(regions_script,con=conn)
sr = pd.read_sql_query(salesPeople_script,con=conn)
```

```
accounts.head(3)
```

	id	name	website	lat	long	primary_poc	sales_rep_id
0	1001	Walmart ...	www.walmart.com ...	40.238496	-75.103297	Tamara Tuma ...	321500
1	1011	Exxon Mobil ...	www.exxonmobil.com ...	41.169156	-73.849374	Sung Shields ...	321510
2	1021	Apple ...	www.apple.com ...	42.290495	-76.084009	Jodee Lupo ...	321520

```
orders.head(3)
```

	id	account_id	occurred_at	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_amt_usd
0	1	1001	2015-10-06 17:31:14	123	22	24	169	613.77	164.78	194.88
1	2	1001	2015-11-05 03:34:33	190	41	57	288	948.10	307.09	462.84
2	3	1001	2015-12-04 04:21:55	85	47	0	132	424.15	352.03	0.00

```
regions.head(3)
```

	id	name
0	1	Northeast ...
1	2	Midwest ...
2	3	Southeast ...

```
sr.head(3)
```

	id	name	region_id
0	321500	Samuel Racine ...	1
1	321510	Eugena Esser ...	1
2	321520	Michel Averette ...	1

```
tables = [orders,accounts,sr,regions]
for table in tables:
    print(table.info(),'\n')
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6911 entries, 0 to 6910
Data columns (total 11 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   id                6911 non-null   int64
 1   account_id        6911 non-null   int64
 2   occurred_at       6911 non-null   datetime64[ns]
 3   standard_qty      6911 non-null   int64
 4   gloss_qty         6911 non-null   int64
 5   poster_qty        6911 non-null   int64
 6   total             6911 non-null   int64
 7   standard_amt_usd  6911 non-null   float64
 8   gloss_amt_usd     6911 non-null   float64
 9   poster_amt_usd    6911 non-null   float64
 10  total_amt_usd     6911 non-null   float64
dtypes: datetime64[ns](1), float64(4), int64(6)
memory usage: 594.0 KB
None
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 349 entries, 0 to 348
Data columns (total 7 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   id            349 non-null    int64  
 1   name          349 non-null    object 
 2   website       349 non-null    object 
 3   lat           349 non-null    float64
 4   long          349 non-null    float64
 5   primary_poc   349 non-null    object 
 6   sales_rep_id  349 non-null    int64  
dtypes: float64(2), int64(2), object(3)
memory usage: 19.2+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   id         50 non-null     int64 
 1   name       50 non-null     object
 2   region_id  50 non-null     int64 
dtypes: int64(2), object(1)
memory usage: 1.3+ KB
None

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4 entries, 0 to 3
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   id      4 non-null      int64 
 1   name    4 non-null      object
dtypes: int64(1), object(1)
memory usage: 192.0+ bytes
None
```

```
orders.rename(columns={'occurred_at':'order_date_time','id':'order_id'}, inplace=True)
accounts.rename(columns={'id':'account_id','sales_rep_id':'sales_person_id','name':'account'},inplace=T
sr.rename(columns = {'id':'sales_person_id','name':'sales_person'}, inplace = True)
regions.rename(columns = {'id':'region_id','name':'region'}, inplace = True)
```

```
# merge the 4 dataframes
df = (orders.merge(accounts, 'inner', left_on = 'account_id', right_on = 'account_id').
      merge(sr, 'inner', left_on = 'sales_person_id', right_on = 'sales_person_id').
      merge(regions, 'inner', left_on = 'region_id', right_on = 'region_id')
      )
df.head()
```

	order_id	account_id	order_date_time	standard_qty	gloss_qty	poster_qty	total	standard_amt_usd	gloss_amt_usd	poster_
0	1	1001	2015-10-06 17:31:14	123	22	24	169	613.77	164.78	
1	2	1001	2015-11-05 03:34:33	190	41	57	288	948.10	307.09	
2	3	1001	2015-12-04 04:21:55	85	47	0	132	424.15	352.03	
3	4	1001	2016-01-02 01:18:24	144	32	0	176	718.56	239.68	
4	5	1001	2016-02-01 19:27:27	108	29	28	165	538.92	217.21	

```
df['region'] = df['region'].str.lstrip().str.rstrip()
df['account'] = df['account'].str.lstrip().str.rstrip()
df['sales_person'] = df['sales_person'].str.lstrip().str.rstrip()
```

```
len(orders), len(df)
```

```
(6911, 6911)
```

Answering the HR Department Business Questions

HR Department Business Request

Q1

Provide a table with the region for each sales representative along with their associated accounts. Your final table should include three columns: the region name, the sales rep name, and the account name. Sort the accounts alphabetically (A-Z) according to account name?

A1 SQL

```
a1_script = """SELECT TRIM(r.name) AS region,TRIM(sr.name) as sales_person, TRIM(a.name) AS Account
               FROM region AS r
               INNER JOIN sales_reps AS sr
                   ON r.id = sr.region_id
               INNER JOIN accounts AS a
                   ON a.sales_rep_id = sr.id"""
a1_sql = pd.read_sql_query(a1_script,con=conn)
a1_sql = a1_sql.sort_values(['region','sales_person','account']).reset_index(drop=True)

a1_sql.to_csv('results/a1_sql.csv')
a1_sql.head(3)
```

	region	sales_person	account
0	Midwest	Carletta Kosinski	Danaher
1	Midwest	Carletta Kosinski	Dollar General
2	Midwest	Carletta Kosinski	International Paper

A1 Pandas

```
a1_pandas = df[['region','sales_person','account']].drop_duplicates()
a1_pandas = a1_pandas.sort_values(['region','sales_person','account']).reset_index(drop=True)

a1_pandas.to_csv('results/a1_pandas.csv')
a1_pandas.head(3)
```

	region	sales_person	account
0	Midwest	Carletta Kosinski	Danaher
1	Midwest	Carletta Kosinski	Dollar General
2	Midwest	Carletta Kosinski	International Paper

Comparison

```
(a1_sql==a1_pandas).sum()
```

```
region          349
sales_person    349
account         349
dtype: int64
```

Q2

Provide a table with the region for each sales representative along with their associated accounts. This time only for accounts where the sales rep has a first name starting with S and in the Midwest region. Your final table should include three columns: the region name, the sales representative name, and the account name. Sort the accounts alphabetically (A-Z) according to account name?

A2 SQL

```
a2_script = """SELECT TRIM(r.name) AS region, TRIM(sr.name) AS sales_person, TRIM(a.name) AS account
               FROM region AS r
               INNER JOIN sales_reps AS sr
                  ON r.id = sr.region_id
               INNER JOIN accounts AS a
                  ON sr.id = a.sales_rep_id
               WHERE r.name = 'Midwest' AND sr.name LIKE 'S%'
               ORDER BY account, sales_person"""
a2_sql = pd.read_sql_query(a2_script,con=conn)
a2_sql = a2_sql.drop_duplicates().\
            sort_values(['account','region','sales_person']).\
            reset_index(drop=True)
a2_sql.head()
```

	region	sales_person	account
0	Midwest	Sherlene Wetherington	Community Health Systems
1	Midwest	Sherlene Wetherington	Progressive
2	Midwest	Sherlene Wetherington	Rite Aid
3	Midwest	Sherlene Wetherington	Time Warner Cable
4	Midwest	Sherlene Wetherington	U.S. Bancorp

A2 Pandas

```
a2_pandas = df[(df['region']=='Midwest') & (df['sales_person'].\
                str.startswith('S'))]\
                [['region','sales_person','account']]
a2_pandas = a2_pandas.drop_duplicates().\
                sort_values(['account','region','sales_person']).\
                    reset_index(drop=True)
a2_pandas.head()
```

	region	sales_person	account
0	Midwest	Sherlene Wetherington	Community Health Systems
1	Midwest	Sherlene Wetherington	Progressive
2	Midwest	Sherlene Wetherington	Rite Aid
3	Midwest	Sherlene Wetherington	Time Warner Cable
4	Midwest	Sherlene Wetherington	U.S. Bancorp

Comparison

```
(a2_sql == a2_pandas).sum()
```

```
region          5
sales_person    5
account         5
dtype: int64
```

Q3

Find the number of sales reps in each region. Your final table should have two columns - the region and the number of sales representative. Order from fewest reps to most reps?

A3 SQL

```
a3_script = """SELECT TRIM(r.name) AS region, COUNT(*) AS sales_person_count
               FROM region AS r
               INNER JOIN sales_reps AS sr
                   ON r.id = sr.region_id
               GROUP BY TRIM(r.name)
               ORDER BY sales_person_count """
a3_sql = pd.read_sql_query(a3_script,con=conn)

a3_sql.to_csv('results/a3_sql.csv')
a3_sql
```

	region	sales_person_count
0	Midwest	9
1	Southeast	10
2	West	10
3	Northeast	21

A3 Pandas

```
a3_pandas = df[['region','sales_person']].\
                drop_duplicates().\
                groupby('region').\
                agg({'sales_person':'count'}).\
                rename(columns={'sales_person':'sales_person_count'}).\
                sort_values('sales_person_count').\
                reset_index()

a3_pandas.to_csv('results/a3_pandas.csv')
a3_pandas
```

	region	sales_person_count
0	Midwest	9
1	Southeast	10
2	West	10
3	Northeast	21

Comparison

```
(a3_sql == a3_pandas).sum()
```

```
region               4
sales_person_count    4
dtype: int64
```

Q4

We would like to identify top performing sales reps, which are sales reps associated with more than 200 orders or more than 750000 in total sales. The middle group has any rep with more than 150 orders or 500000 in sales. Create a table with the sales rep name, the total number of orders, total sales across all orders, and a column with top, middle, or low depending on this criteria. Place the top sales people based on dollar amount of sales first in your final table.

A4 SQL

```
a4_script = """SELECT
                    TRIM(sr.name) AS sales_person
                    ,COUNT(*) AS total_orders
                    ,SUM(o.total_amt_usd) AS total_revenue
                    ,CASE
                        WHEN COUNT(*) > 200 OR SUM(O.total_amt_usd) > 750000
                            THEN 'top'
                        WHEN COUNT(*) > 150 OR SUM(O.total_amt_usd) > 500000
                            THEN 'middle'
                        ELSE
                        'not'
                    END AS sales_person_level
                FROM orders AS o
                INNER JOIN accounts a
                    ON o.account_id = a.id
                INNER JOIN sales_reps AS sr
                    ON sr.id = a.sales_rep_id
                GROUP BY sr.name
                ORDER BY total_revenue DESC """
a4_sql = pd.read_sql_query(a4_script,con=conn)

a4_sql.head()
```

	sales_person	total_orders	total_revenue	sales_person_level
0	Earlie Schleusner ...	335	1098137.72	top
1	Tia Amato ...	267	1010690.60	top
2	Vernita Plump ...	299	934212.93	top
3	Georgianna Chisholm ...	256	886244.12	top
4	Arica Stoltzfus ...	186	810353.34	top

A4 Pandas

```
a4_pandas = df[['order_id','sales_person','total','total_amt_usd']].\
            groupby('sales_person').\
            agg({'order_id':'count','total_amt_usd':'sum'}).\
            rename(columns={'order_id':'total_orders','total_amt_usd':'total_revenue'}).\
            sort_values('total_revenue', ascending=False).\
            reset_index()

condition1 = (a4_pandas['total_orders']>150) | (a4_pandas['total_revenue']>500000)
condition2 = (a4_pandas['total_orders']>200) | (a4_pandas['total_revenue']>750000)

a4_pandas.loc[:,'sales_person_level'] = 'not'
a4_pandas.loc[condition1 ,'sales_person_level'] = 'middle'
a4_pandas.loc[condition2,'sales_person_level'] = 'top'

a4_pandas.head()
```

	sales_person	total_orders	total_revenue	sales_person_level
0	Earlie Schleusner	335	1098137.72	top
1	Tia Amato	267	1010690.60	top
2	Vernita Plump	299	934212.93	top
3	Georgianna Chisholm	256	886244.12	top
4	Arica Stoltzfus	186	810353.34	top

Comparison

```
(a4_sql==a4_pandas).sum()
```

```
sales_person          50
total_orders          50
total_revenue         50
sales_person_level    50
dtype: int64
```

Q5

Provide the name of the sales represntative in each region with the largest amount of total_amt_usd sales?

A5 SQL

```
a5_script = """SELECT
                    TRIM(T3.sales_person) as sales_person
                    ,T3.region
                    ,T3.total_revenue
                FROM
                    (SELECT
                        region
                        ,MAX(total_revenue) AS maximum_total_revenue
                     FROM
                        (SELECT
                            sr.name AS sales_person
                            ,TRIM(r.name) region
                            ,SUM(o.total_amt_usd) AS total_revenue
                         FROM sales_reps AS sr
                         INNER JOIN accounts AS a
                             ON sr.id = a.sales_rep_id
                         INNER JOIN orders AS o
                             ON a.id = o.account_id
                         INNER JOIN region AS r
                             ON r.id = sr.region_id
                         GROUP BY sr.name,r.name)
                         AS T1
                    GROUP BY Region)
                    AS T2
```

```
                INNER JOIN
                     (SELECT
                         sr.name AS sales_person
                         ,TRIM(r.name) region
                         ,SUM(o.total_amt_usd) AS total_revenue
                      FROM sales_reps AS sr
                      INNER JOIN accounts AS a
                          ON sr.id = a.sales_rep_id
                      INNER JOIN orders AS o
                          ON o.account_id = a.id
                      INNER JOIN region AS r
                          ON r.id = sr.region_id
                      GROUP BY sr.name,r.name)
                      AS T3
                ON T3.region = T2.Region AND T3.total_revenue = T2.maximum_total_revenue
                ORDER BY region ASC"""

a5_sql = pd.read_sql_query(a5_script,con=conn)[['region','sales_person','total_revenue']]

a5_sql
```

	region	sales_person	total_revenue
0	Midwest	Charles Bidwell ...	675637.19
1	Northeast	Tia Amato ...	1010690.60
2	Southeast	Earlie Schleusner ...	1098137.72
3	West	Georgianna Chisholm ...	886244.12

A5 Pandas

```
q5_list = []
regions = df['region'].value_counts().index.values.tolist()

agg_df = df.groupby(['region','sales_person']).\
            agg({'total_amt_usd':'sum'}).\
            reset_index().\
            rename(columns={'total_amt_usd':'total_revenue'}).\
            sort_values(['region','total_revenue'], ascending=[True, False])

for region in regions:
    new_df = agg_df[agg_df['region']==region]
    new_df = new_df.sort_values('total_revenue', ascending=False).head(1)
    q5_list.append(new_df)

a5_pandas = pd.concat(q5_list, axis=0).sort_values('region', ascending=True).reset_index(drop=True)
a5_pandas
```

Finance Department Business Request

Q6

What are the average quantity & average revenue for each paper type (standard, gloss, poster)?

A6 SQL

```
a6_script = """SELECT
                    ROUND(AVG(standard_qty),0) AS Average_Standard_Quantity
                    ,ROUND(AVG(standard_amt_usd),2) AS Average_Standard_Revenue
                    ,ROUND(AVG(gloss_qty),0) AS Average_Gloss_Quantity
                    ,ROUND(AVG(gloss_amt_usd),2) AS Average_Gloss_Revenue
                    ,ROUND(AVG(poster_qty),0) AS Average_Poster_Quantity
                    ,ROUND(AVG(poster_amt_usd),2) AS Average_Poster_Revenue
                FROM orders"""

a6_sql = pd.read_sql_query(a6_script,con=conn)

a6_sql.head()
```

	average_standard_quantity	average_standard_revenue	average_gloss_quantity	average_gloss_revenue	average_poster_quanti
0	280.0	1399.56	147.0	1098.68	105

A6 Pandas

```
a6_pandas = pd.DataFrame()

a6_pandas.loc[0,'average_standard_quantity'] = df['standard_qty'].mean().round(0)
a6_pandas.loc[0,'average_standard_revenue'] = df['standard_amt_usd'].mean().round(2)
a6_pandas.loc[0,'average_gloss_quantity'] = df['gloss_qty'].mean().round(0)
a6_pandas.loc[0,'average_gloss_revenue'] = df['gloss_amt_usd'].mean().round(2)
a6_pandas.loc[0,'average_poster_quantity'] = df['poster_qty'].mean().round(0)
a6_pandas.loc[0,'average_poster_revenue'] = df['poster_amt_usd'].mean().round(2)
a6_pandas
```

	average_standard_quantity	average_standard_revenue	average_gloss_quantity	average_gloss_revenue	average_poster_quanti
0	280.0	1399.56	147.0	1098.68	105

Comparison

```
(a6_sql==a6_pandas).sum()
```

```
average_standard_quantity    1
average_standard_revenue     1
average_gloss_quantity       1
average_gloss_revenue        1
average_poster_quantity      1
average_poster_revenue       1
dtype: int64
```

Q7

What is median of (total_amt_usd) values?

A7 SQL

```
a7_script = """
               SELECT
                    AVG(1.0*total_amt_usd) as medain_revenue
               FROM (SELECT
                        total_amt_usd
                     FROM orders
                     ORDER BY total_amt_usd
                     OFFSET ((SELECT COUNT(*) FROM orders) - 1)/2
                     FETCH NEXT (1 + (1-(SELECT COUNT(*) FROM orders)%2)) ROWS ONLY
                    )  AS T1
                    """

a7_sql = pd.read_sql_query(a7_script,con=conn)

a7_sql
```

	medain_revenue
0	2483.16

A7 Pandas

```
a7_pandas = df['total_amt_usd'].median()
a7_pandas
```

```
2483.16
```

Comparison

```
a7_sql == a7_pandas
```

	medain_revenue
0	True

Q8

In which month of which year did Walmart spend the most on gloss paper in terms of dollars?

```
a8_script = """SELECT
                    CAST(DATE_PART('month', o.occurred_at) AS INTEGER) AS month
                    ,CAST(DATE_PART('year', o.occurred_at) AS INTEGER) AS year
                    ,SUM(o.gloss_amt_usd) AS gloss_total_revenue
               FROM orders AS o
               INNER JOIN accounts AS a
                   ON a.id = o.account_id
               WHERE a.name = 'Walmart'
               GROUP BY DATE_PART('month', o.occurred_at), DATE_PART('year', o.occurred_at)
               ORDER BY  gloss_total_revenue DESC
               LIMIT(1)
               """

a8_sql = pd.read_sql_query(a8_script,con=conn)[['year','month','gloss_total_revenue']]

a8_sql
```

	year	month	gloss_total_revenue
0	2016	5	9257.64

A8 Pandas

```
q8_pandas = df[df['account']=='Walmart'].\
                groupby([df['order_date_time'].dt.year,df['order_date_time'].dt.month]).\
                agg(gloss_total_revenue = ('gloss_amt_usd','sum')).\
                sort_values('gloss_total_revenue', ascending=False).\
                head(1)

q8_pandas['year'] = q8_pandas.index.get_level_values(0)
q8_pandas['month'] = q8_pandas.index.get_level_values(1)

q8_pandas.reset_index(drop=True, inplace=True)
q8_pandas = q8_pandas[['year','month','gloss_total_revenue']]

q8_pandas
```

	year	month	gloss_total_revenue
0	2016	5	9257.64

Comparison

```
(a8_sql == q8_pandas).sum()
```

```
year                1
month               1
gloss_total_revenue 1
dtype: int64
```

Q9

What is the lifetime average amount spent in terms of total_amt_usd, including only the companies that spent more per order, on average, than the average of all orders?

A9 SQL

```
a9_script = """SELECT
                    ROUND(AVG(average_revenue),0) AS top10_companies_average_of_average_revenue
               FROM (SELECT
                        o.account_id AS account
                        ,AVG(o.total_amt_usd) AS average_revenue
                      FROM orders AS o
                      GROUP BY o.account_id
                      HAVING AVG(o.total_amt_usd) > ( SELECT
                                                          AVG(o.total_amt_usd) AS average_spent
                                                      FROM orders AS o )
                       ) AS T1 """

a9_sql = pd.read_sql_query(a9_script,con=conn)

a9_sql
```

	top10_companies_average_of_average_revenue
0	4721.0

A9 Pandas

```
average_spent = df['total_amt_usd'].mean()

a9_pandas = df[['account','total_amt_usd']].groupby('account').\
                agg(average_revenue=('total_amt_usd','mean'))

a9_pandas = a9_pandas[a9_pandas['average_revenue']>average_spent]

a9_pandas = a9_pandas['average_revenue'].mean().round(0)

a9_pandas
```

```
4721.0
```

Comparison

```
(a9_sql.values==a9_pandas).sum()
```

```
1
```

Q10

What is the lifetime average total amount spent in terms of total_amt_usd for the top 10 total spending accounts?

A10 SQL

```
a10_script = """SELECT
                    AVG(total_revenue) AS top10_companies_average_total_revenue
                FROM (SELECT
                        a.id AS account_id
                        ,a.name AS Account
                        ,SUM(o.total_amt_usd) AS total_revenue
                      FROM orders AS o
                      INNER JOIN accounts AS a
                          ON a.id = o.account_id
                      GROUP BY a.id, a.name
                      ORDER BY total_revenue DESC
                      LIMIT(10)
                     ) AS T1"""

a10_sql = pd.read_sql_query(a10_script,con=conn)

a10_sql
```

	top10_companies_average_total_revenue
0	304846.969

A10 Pandas

```
a10_pandas = df[['account','total_amt_usd']].groupby('account').\
                agg(total_revenue=('total_amt_usd','sum')).\
                sort_values('total_revenue',ascending=False).\
                head(10).\
                mean()

a10_pandas
```

```
total_revenue    304846.969
dtype: float64
```

Comparison

```
(a10_sql.values==a10_pandas.values).sum()
```

```
1
```

Q11

How many accounts spent more than 30,000 usd total across all orders?

A11 SQL

```
a11_script = """WITH T1 AS
                    (SELECT
                        a.id AS account_id
                        ,SUM(o.total_amt_usd) AS total_revenue
                     FROM accounts AS a
                     INNER JOIN orders AS o
                       ON a.id = o.account_id
                     GROUP BY a.id
                     HAVING SUM(o.total_amt_usd) > 30000
                    )
                SELECT
                    COUNT(*)
                FROM T1
                """

a11_sql = pd.read_sql_query(a11_script,con=conn)

a11_sql
```

	count
0	204

A11 Pandas

```
a11_pandas = df[['account_id','total_amt_usd']].\
                groupby('account_id').\
                agg(total_revenue=('total_amt_usd','sum'))

a11_pandas = len(a11_pandas[a11_pandas ['total_revenue']>30000])
a11_pandas
```

```
204
```

comparison

```
(a11_sql.values==a11_pandas).sum()
```

```
1
```

Q12

Provide a table to show the number of orders in each of three categories, based on the total number of items in each order. The three categories are: 'At Least 2000', 'Between 1500 and 2000' and 'Less than 1500'?

A12 SQL

```
a12_script = """with T1 AS
                    (SELECT
                        id,
                        CASE
                            WHEN total >= 2000
                                THEN 'At Least 2000'
                            WHEN total >= 1500
                                THEN 'Between 1500 and 2000'
                            ELSE
                                'Less than 1500'
                        END AS order_category
                      FROM orders
                     )

                SELECT order_category, COUNT(*) AS order_count
                FROM T1
                INNER JOIN orders o
                    ON T1.ID = O.id
                GROUP BY order_category
                 """

a12_sql = pd.read_sql_query(a12_script,con=conn)
a12_sql
```

	order_category	order_count
0	At Least 2000	70
1	Between 1500 and 2000	60
2	Less than 1500	6781

A12 Pandas

```
new_df = df[['account_id','total']].copy()

less_1500_conddition = new_df['total']<1500
between_1500_2000_condition = (new_df['total']>=1500) & (new_df['total']<2000)
at_least_2000_condition = new_df['total']>=2000


new_df.loc[less_1500_conddition,'order_category'] = 'Less than 1500'
new_df.loc[between_1500_2000_condition,'order_category'] = 'Between 1500 and 2000'
new_df.loc[at_least_2000_condition,'order_category'] = 'At Least 2000'


a12_pandas = new_df.groupby('order_category').agg(order_count=('account_id','count')).reset_index()
a12_pandas
```

	order_category	order_count
0	At Least 2000	70
1	Between 1500 and 2000	60
2	Less than 1500	6781

Comparison

```
(a12_sql==a12_pandas).sum()
```

```
order_category    3
order_count       3
dtype: int64
```

A13 SQL

```
a13_script = """
            SELECT
                TRIM(r.name) AS region
                ,COUNT(o.id) AS total_orders
            FROM orders AS o
            INNER JOIN accounts AS a
                ON o.account_id = a.id
            INNER JOIN sales_reps AS sr
                ON a.sales_rep_id = sr.id
            INNER JOIN region AS r
                ON r.id = sr.region_id
            GROUP BY r.name
            HAVING  SUM(o.total_amt_usd) = ( SELECT
                                                MAX(total_revenue)
                                            FROM (SELECT
                                                    r.name AS Region
                                                    ,SUM(o.total_amt_usd) AS total_revenue
                                                FROM orders AS O
                                                INNER JOIN accounts AS a
                                                    ON o.account_id = a.id
                                                INNER JOIN sales_reps AS sr
                                                    ON sr.id = a.sales_rep_id
                                                INNER JOIN region AS r
                                                    ON r.id = sr.region_id
                                                GROUP BY r.name
                                                ) AS T1
                                            )
            """

a13_sql = pd.read_sql_query(a13_script,con=conn)
a13_sql
```

	region	total_orders
0	Northeast	2356

A13 Pandas

```
a13_pandas = df.groupby('region').\
                agg(total_revenue=('total_amt_usd','sum'),total_orders=('order_id','count')).\
                sort_values('total_revenue',ascending=False).\
                head(1).reset_index()[['region','total_orders']]

a13_pandas
```

	region	total_orders
0	Northeast	2356

Comparison

```
(a13_sql==a13_pandas).sum()
```

```
region          1
total_orders    1
dtype: int64
```

Q14

How many accounts had more total purchases than the account name which has bought the most standard_qty paper throughout their lifetime as a customer?

```
a14_script = """
    SELECT
        COUNT(*) AS accounts_count
    FROM
    (SELECT
        a.name AS account
     FROM orders AS o
     INNER JOIN accounts AS a
     ON a.id = o.account_id
     GROUP BY a.name
     HAVING SUM(o.total) > (SELECT
                                total_orders
                            FROM (SELECT
                                      a.name AS Account
                                      ,SUM(o.standard_qty) AS total_standard_quantity
                                      ,SUM(o.total) AS total_orders
                                  FROM accounts AS a
                                  INNER JOIN orders AS o
                                      ON a.id = o.account_id
                                  GROUP BY a.name
                                  ORDER BY total_standard_quantity DESC
                                  LIMIT(1)
                                  ) AS T1
                             )
        ) T2"""

a14_sql = pd.read_sql_query(a14_script,con=conn)
a14_sql
```

	accounts_count
0	3

A14 Pandas

```
target_total_orders = df.groupby('account').\
                         agg(total_standard_qty=('standard_qty','sum'),total_orders=('total','sum')).\
                         sort_values('total_standard_qty',ascending=False).\
                         head(1)['total_orders'].values[0]

a14_pandas = df.groupby('account').agg(total_orders=('total','sum'))
a14_pandas = len(a14_pandas[a14_pandas['total_orders']>target_total_orders])

a14_pandas
```

```
3
```

Comparison

```
(a14_sql.values==a14_pandas).sum()
```

```
1
```

Q15

Provide a record for the revenue, previous date revenue, difference from the previous date revenue for the top most spending account?

```
a15_script = """
    SELECT
        account_id
        ,occurred_at AS date
        ,total_amt_usd AS revenue
        ,LEAD(total_amt_usd) OVER (ORDER BY occurred_at)  AS previous_revenue
        ,ROUND(total_amt_usd-LEAD(total_amt_usd) OVER (ORDER BY occurred_at),2) AS difference
    FROM orders
    WHERE account_id IN (SELECT
                             account_id
                         FROM (SELECT
                                   account_id
                                   ,SUM(total_amt_usd) AS total_revenue
                               FROM orders
                               GROUP BY account_id
                               ORDER BY total_revenue DESC
                               LIMIT(1)
                              ) AS T1

                          )
    ORDER BY date
                          """

a15_sql = pd.read_sql_query(a15_script,con=conn)

a15_sql.to_csv('../results/a15_sql.csv')
a15_sql.head()
```

	account_id	date	revenue	previous_revenue	difference
0	4211	2013-12-12 09:48:16	733.89	8680.34	-7946.45
1	4211	2013-12-12 09:54:34	8680.34	8355.96	324.38
2	4211	2014-01-11 09:33:59	8355.96	1388.73	6967.23
3	4211	2014-01-11 09:42:04	1388.73	8077.66	-6688.93
4	4211	2014-02-09 09:00:48	8077.66	1421.88	6655.78

A15 Pandas

```
target_account = df.groupby('account').\
                    agg(total_revenue=('total_amt_usd','sum')).\
                    sort_values('total_revenue',ascending=False).\
                    index[0]

a15_pandas = df[df['account']==target_account][['account_id','order_date_time','total_amt_usd']].\
                rename(columns={'total_amt_usd':'revenue','order_date_time':'date'}).\
                sort_values('date').\
                reset_index(drop=True)

a15_pandas['previous_revenue'] = a15_pandas['revenue'].shift(-1)
a15_pandas['difference'] = (a15_pandas['revenue']-a15_pandas['previous_revenue']).round(2)

a15_pandas.to_csv('../results/a15_pandas.csv')
a15_pandas.head()
```

	account_id	date	revenue	previous_revenue	difference
0	4211	2013-12-12 09:48:16	733.89	8680.34	-7946.45
1	4211	2013-12-12 09:54:34	8680.34	8355.96	324.38
2	4211	2014-01-11 09:33:59	8355.96	1388.73	6967.23
3	4211	2014-01-11 09:42:04	1388.73	8077.66	-6688.93
4	4211	2014-02-09 09:00:48	8077.66	1421.88	6655.78

Comparison

```
(a15_sql==a15_pandas).sum()
```

```
account_id          62
date                62
revenue             62
previous_revenue    61
difference          61
dtype: int64
```

The reason why 'previous_revenue' and 'difference' columns are not equal between the two dataframes is that their values in the last rows are NaN because there are no previous values. Since NaN is not a number those values are not equals accoding to pandas alogorithm.

So, if NaN is not a nubmer, what is it?...it is a type of very tasty bread...owh wait..that is a different topic!

```
conn.close()
```