SQL Query Report

All of the SQL queries were performed using Mode (a Power BI and Data Science platform) and PgAdmin.

Dialect: PostgreSQL

All the queries that are run below are aimed at solving day-to-day problems occurring in industrial settings and focuses on how to efficiently leverage Power BI tools for strategical gain and improved interactive visualizations to collect useful and meaningful insights.

Note: I have also included queries that I have performed for my Data Science program at Thinkful as well.

Report is created by:
Sajith Gowthaman

Email: sajithgowthaman@gmail.com
Portfolio: sajithgowthaman.github.io/
LinkedIn: linkedin.com/in/SajithG/
GitHub: github.com/sajithgowthaman/

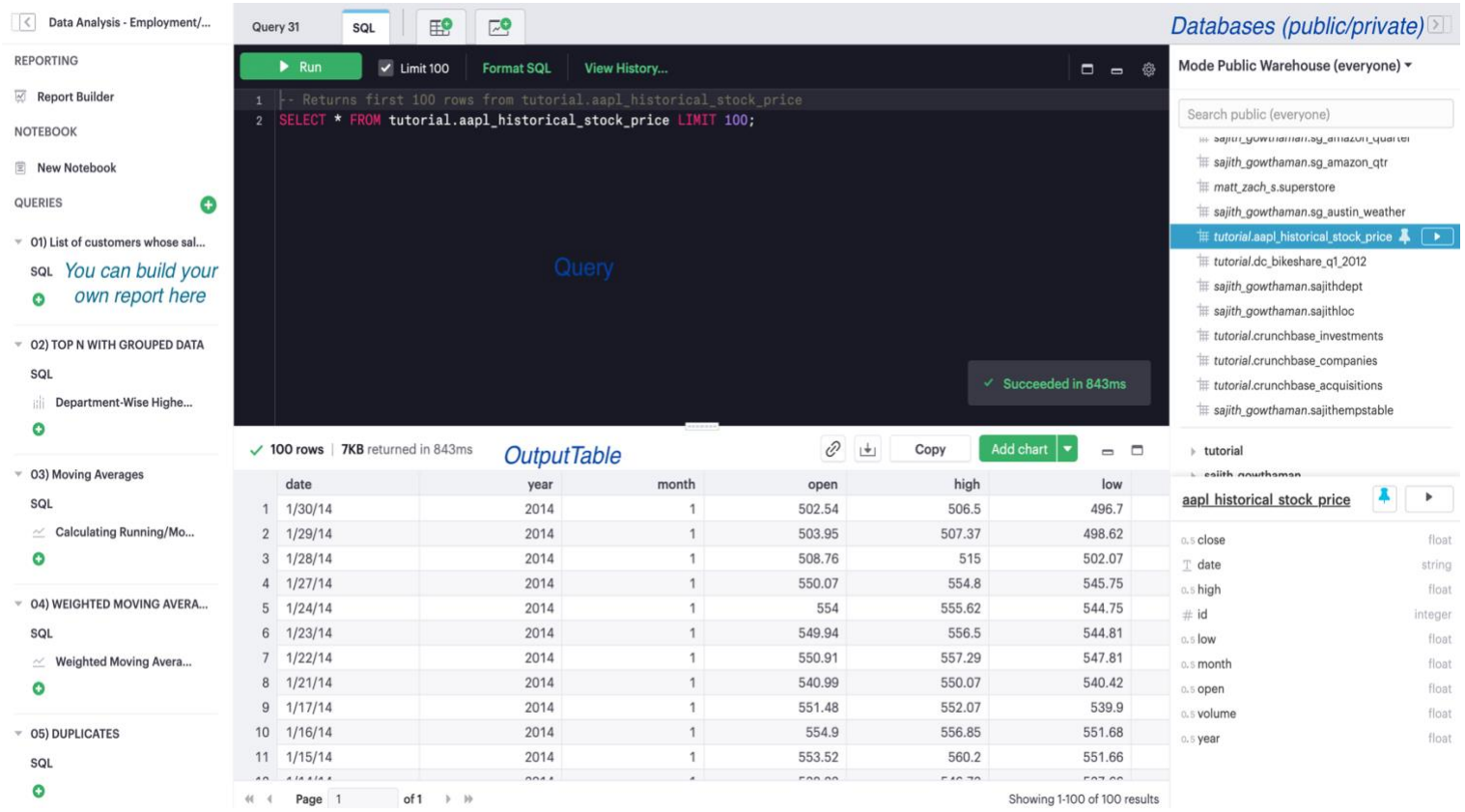
I am Engineering Management graduate with a certification in Data Science with an aim to provide data-driven decisions to solve analytical problems. I aspire to leverage Business Intelligence to provide stakeholders with actionable and profitable strategies based on accurate data analysis. This report will consist of queries that I ran to improve my database querying skills.

There are variety of online business intelligence tools that can be used for enterprise level businesses. For a student like me, I do not own an enterprise account in Mode or Silota (another BI tool). Due to that, I am unable to share my Mode reports online. Hence, I have created a PDF file containing all the important queries that I thought will come handy to me in the future.

Table of Content

MODE SQL

Before I get into SQL queries, I would like to give a short intro to Mode so you can understand the rest of the report better.



The left side of the page is to build your own report, the center of the page is to pass queries with results output shown exactly underneath it. To the right is the databases that is connected to.

Currently, I will be working with databases provided to me from Data Science program at Thinkful using PostgreSQL (PgAdmin server) and my own data that I created to fit the question.

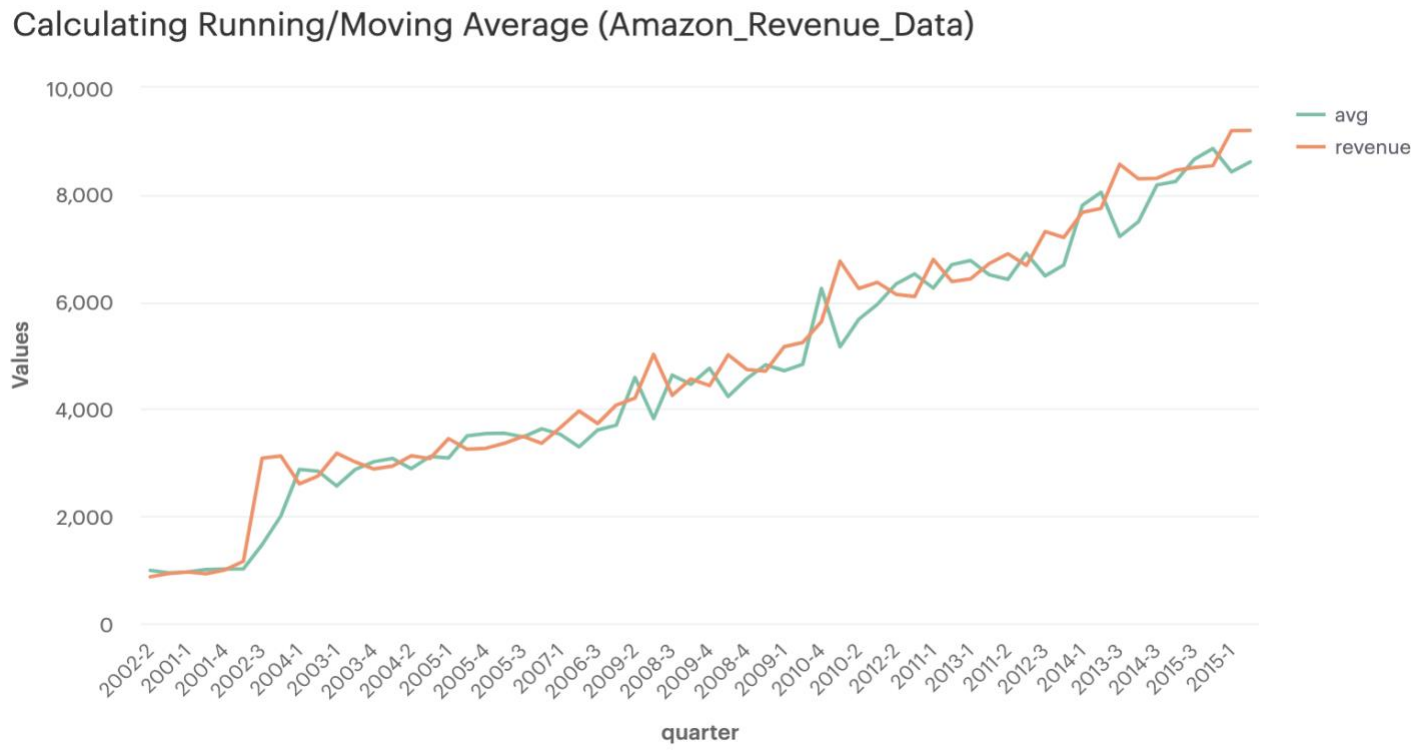
Okay, Let's dive into the queries!

Moving Averages: Essentially, moving averages are calculated to smoothen out the fluctuation and showcase the long-term trends or cycles (If existing).

Run | Limit 100 | Format SQL | View History...

```
1  --03 - Calculating Running/Moving Average in SQL
2  SELECT QUARTER,
3         REVENUE,
4         AVG(REVENUE) OVER (ORDER BY QUARTER ROWS BETWEEN 3 PRECEDING AND CURRENT ROW)
5  FROM sajith_gowthaman.sg_amazon_qtr
6
```

ROWS BETWEEN 3 PRECEDING AND CURRENT ROW is nothing but taking of the preceding 3 and the current to calculated the moving average for every quarter (n=4)



Weighted Moving Averages: Similar to the previous query, we add weights for each period. It is common that the most recent period gets a high weight compared to the rest.

Note: Make sure the weights add up to 1 (I have taken 0.4,0.3,0.2 and 0.1 as weights).

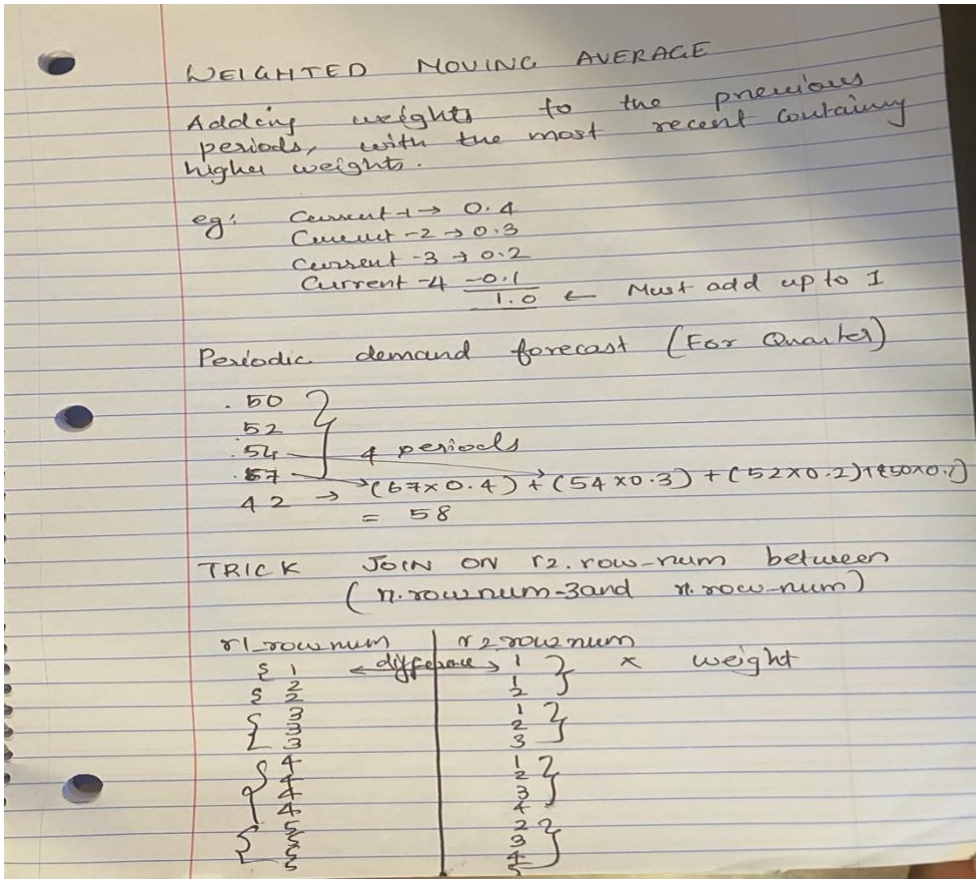
```sql
with r AS
    (SELECT quarter, revenue, ROW_NUMBER() OVER()  --Get row numbers
     FROM sajith_gowthaman.sg_amazon_qtr
     )
SELECT r.quarter, avg(r.revenue) AS revenue,
SUM(CASE
    WHEN r.row_number - r2.row_number = 0 THEN 0.4 * r2.revenue --current
    WHEN r.row_number - r2.row_number = 1 THEN 0.3 * r2.revenue --preceding by 1 quarter
    WHEN r.row_number - r2.row_number = 2 THEN 0.2 * r2.revenue --preceding by 2 quarters
    WHEN r.row_number - r2.row_number = 3 THEN 0.1 * r2.revenue --preceding by 3 quarters
END) AS Weighted_avgs
FROM r
JOIN r r2 on r2.row_number between r.row_number - 3 and r.row_number
GROUP BY 1
ORDER BY 1
```

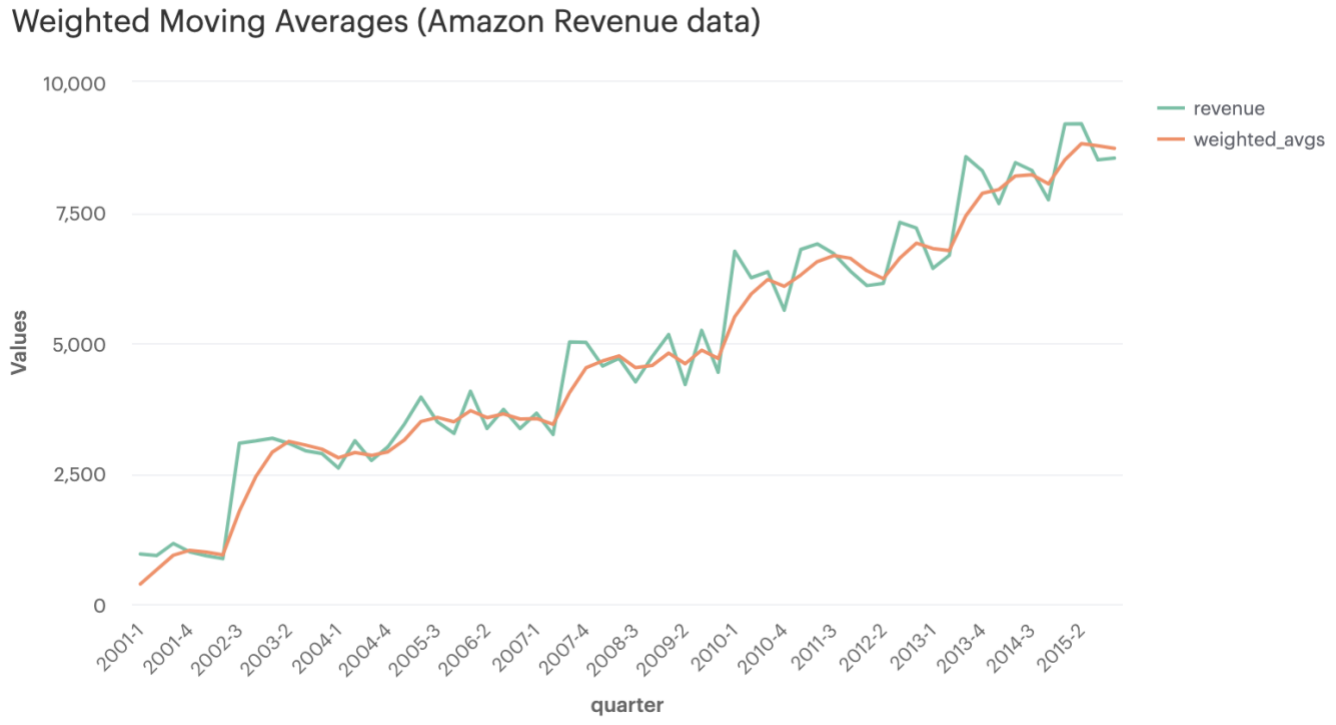
The inner join performed above returns a table that looks like this, from which the row-wise subtraction occurs:

	quarter	revenue	r_row_num	r2_row_num
1	2001-1	956	1	1
2	2001-2	929	2	1
3	2001-2	929	2	2
4	2001-3	1159	3	1
5	2001-3	1159	3	2
6	2001-3	1159	3	3
7	2001-4	996	4	1
8	2001-4	996	4	2
9	2001-4	996	4	3
10	2001-4	996	4	4
11	2002-1	923	5	2
12	2002-1	923	5	3
13	2002-1	923	5	4
14	2002-1	923	5	5
15	2002-2	867	6	3
16	2002-2	867	6	4

Now based on the row difference, the weight is multiplied to the sliding window that acquires new values.

How I interpreted comparing the two row number columns:





Top N with Grouped Data: We so often come across a point where we will need to choose the top few customers/products based on applicable conditions.

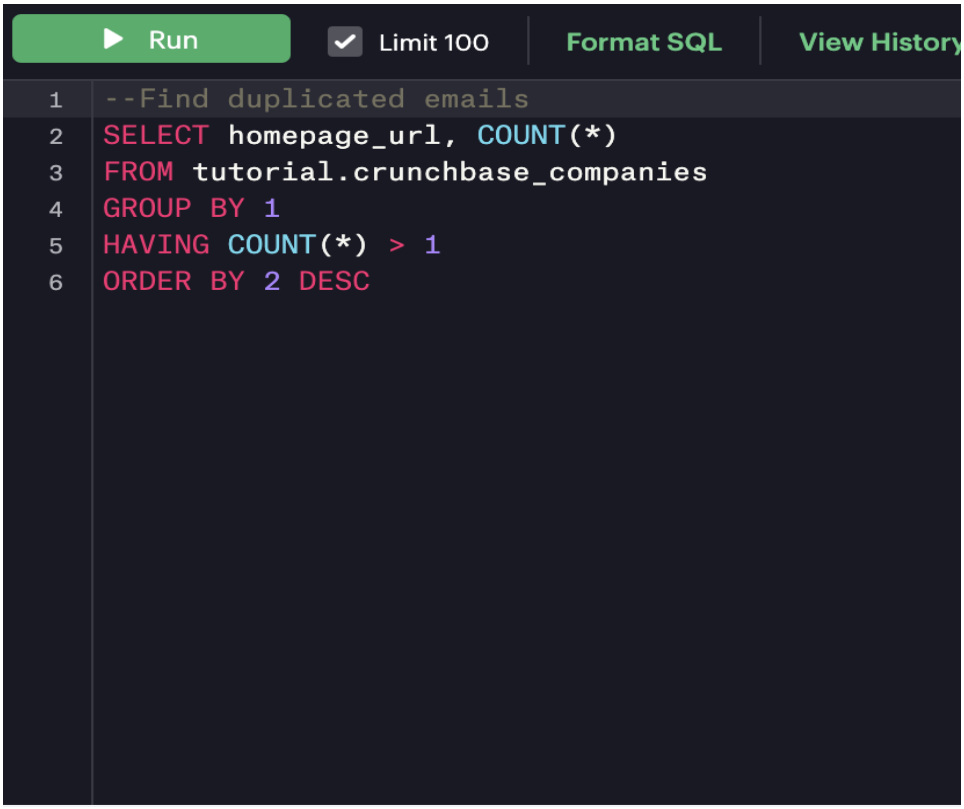
Note: Being a top customer does not necessarily have to be just the amount of goods that he/she bought. It can also be subscription duration / click through rates / number of customers referred / etc.

```sql
--Department wise top 3 highest salary. (Note: DENSE_RANK() is applicable as well)
WITH CTE AS
  (
    SELECT emp.sal, emp.ename, dept.dname,
    ROW_NUMBER () OVER (PARTITION BY dept.dname ORDER BY emp.sal DESC) as rank
    FROM sajith_gowthaman.sajithempstable emp
    JOIN sajith_gowthaman.sajithdept dept
    ON emp.deptno = dept.deptno
  )
SELECT * FROM CTE
WHERE cte.rank  <= 3

```

This is a simple query that creates a temporary table with row_numbers assigned to each row, and then calling out the top three row numbers. Dense_Rank works here too!

	dname	ename	sal
1	RESEARCH	FORD	3000
2	RESEARCH	JONES	2975

Duplicate Entries: Essentials – getting rid of duplicate entries.



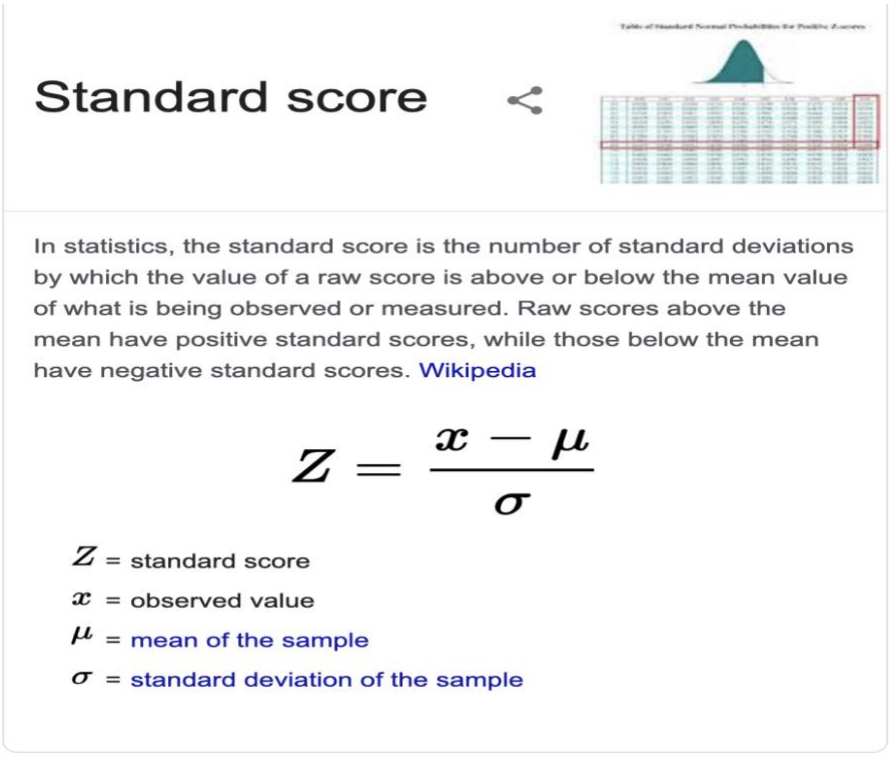
✓ **63 rows** | **2KB** returned in 1s

	homepage_url	count
1		1387
2	http://www.istorytime.com	2
3	http://www.cynvenio.com	2
4	http://www.dachisgroup.com	2
5	http://www.ikang.com	2
6	http://www.twist.com	2
7	http://www.natera.com	2
8	http://youbeauty.com	2
9	http://www.nitropdf.com	2

We can get rid of them by simply adding DELETE from table Where col in (pass in the above query).

Z-Scores:

In simple terms, Z-score is used to find how far a data point is from the mean.



Credits for above picture: Wikipedia.

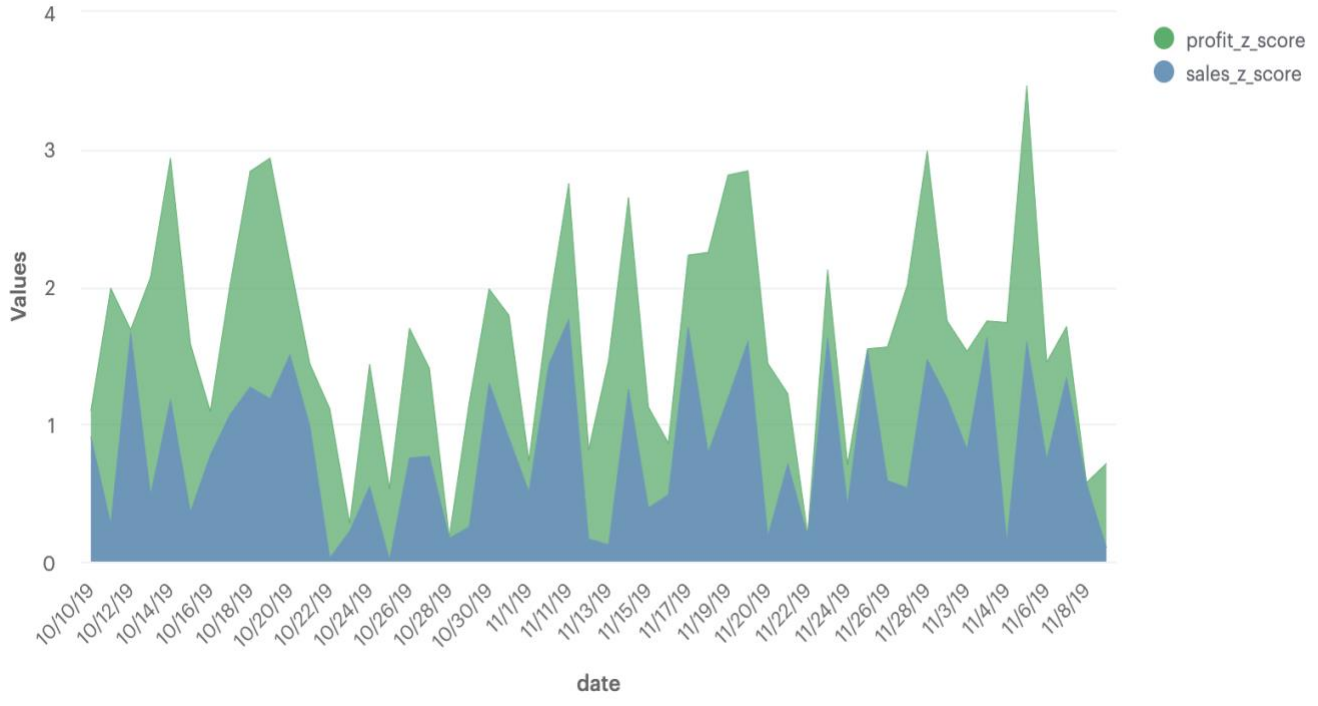
Run | Limit 100 | Format SQL | View History...

```
WITH sales_stats AS
    (SELECT AVG(sales) AS Mean,
            STDDEV(sales) AS Sd
    FROM sajith_gowthaman.sg_z_score),
    profit_stats AS
    (SELECT AVG(profit) AS Mean,
            STDDEV(profit) AS Sd
    from sajith_gowthaman.sg_z_score)
SELECT date,
    ABS(sales - sales_stats.mean) / sales_stats.sd AS Sales_z_score,
    ABS(profit - profit_stats.mean) / profit_stats.sd AS Profit_z_score
FROM sales_stats,
    profit_stats,
    sajith_gowthaman.sg_z_score;
```

✓ **52 rows** | **1KB** returned in 786ms

	date	sales_z_score	profit_z_score
1	10/10/19	0.9102650877646449	0.18620780376423798
2	10/11/19	0.2729910742419003	1.7159827400406056
3	10/12/19	1.6648545015414125	0.022377228280569008
4	10/13/19	0.4823897529492788	1.5874208963903471
5	10/14/19	1.1782749128687755	1.7578107496112743
6	10/15/19	0.3589292603009978	1.2253487652937012
7	10/16/19	0.7740023192949926	0.3188565820046343
8	10/17/19	1.0723651765283384	0.948395876833627
9	10/18/19	1.2697529928287572	1.570215406325693
10	10/19/19	1.1850252037564075	1.7514028398217991
11	10/20/19	1.504476900797322	0.6704338687216674

Sales vs Profit Z-Scores



List of Employees whose salary increased from XXXX-XXXX and decreased from XXXX-XXXX

I used a public data found Mode's public warehouse.

Run | Limit 100 | Format SQL | View History...

```
With results AS (SELECT *, EXTRACT('YEAR' FROM order_date) as year
FROM matt_zach_s.superstore),

data AS (
    SELECT customer_name,
        COALESCE(SUM(CASE WHEN year = 2015 THEN quantity ELSE 0 END), 0) AS total2015,
        COALESCE(SUM(CASE WHEN year = 2016 THEN quantity ELSE 0 END), 0) AS total2016,
        COALESCE(SUM(CASE WHEN year = 2017 THEN quantity ELSE 0 END), 0) AS total2017,
        COALESCE(SUM(CASE WHEN year = 2018 THEN quantity ELSE 0 END), 0) AS total2018,
        COALESCE(SUM(CASE WHEN year = 2019 THEN quantity ELSE 0 END), 0) AS total2019
    FROM results
    WHERE year BETWEEN 2015 AND 2019
    GROUP BY customer_name
)
SELECT *
FROM data
WHERE total2015 < total2016
    AND total2016 < total2017
    AND total2018 > total2019;
```

Basically, I am creating a column for each year, and then inputting the conditions mentioned above.

✓ **94 rows** | **5KB** returned in 722ms

	customer_name	total2015	total2016	total2017	total2018	total2019
1	Adam Bellavance	0	2	23	31	0
2	Adam Shillingsburg	13	16	26	26	0
3	Alan Haines	0	4	14	10	0
4	Alejandro Grove	4	6	46	2	0
5	Amy Cox	8	15	16	4	0
6	Anna Gayman	0	6	23	25	0
7	Art Ferguson	0	2	38	11	0
8	Arthur Wiediger	11	12	36	9	0
9	Bart Watters	0	20	35	19	0

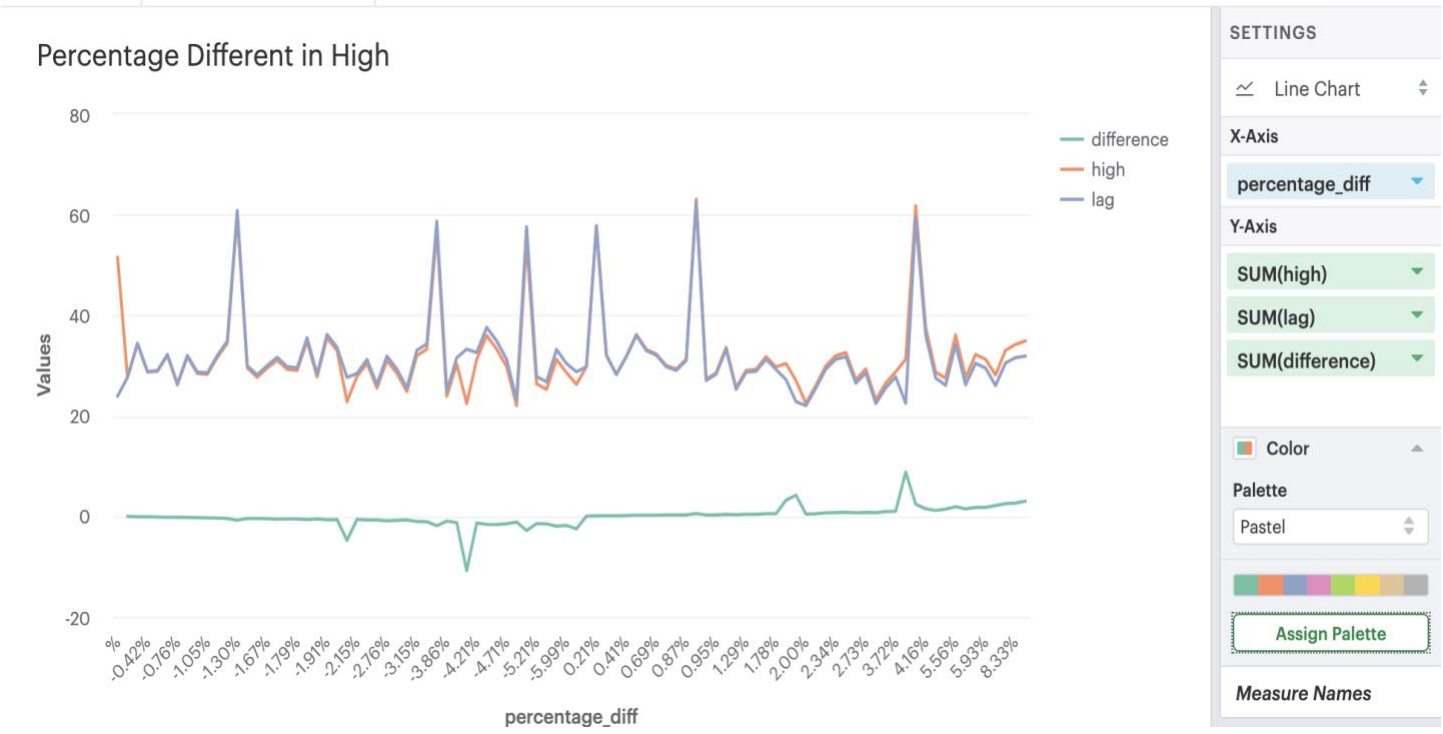
Percentage Difference:

```sql
--LAG and LEAD for apple stock highs
SELECT year,
        month,
        high,
        Lag,
        ROUND(CAST(difference as numeric), 2) AS difference,
        CONCAT(ROUND(CAST(difference *100/ Lag as numeric), 2),'%') AS  percentage_diff
FROM
(SELECT date, year, month, high,
      LAG(high, 1) OVER  (PARTITION BY year ORDER BY month) Lag,
      high -LAG(high, 1) OVER  (PARTITION BY year ORDER BY month) Difference
      FROM tutorial.aapl_historical_stock_price
      ) AS sub
```

Two steps to this:

1) Create a column that has a Lag of the “high” column
2) Next, subtract “high”- lag and partition by Month to get the difference (if required).

Getting the percentage is just concatenating, (difference * 100 / Lag) and ‘%’ together.



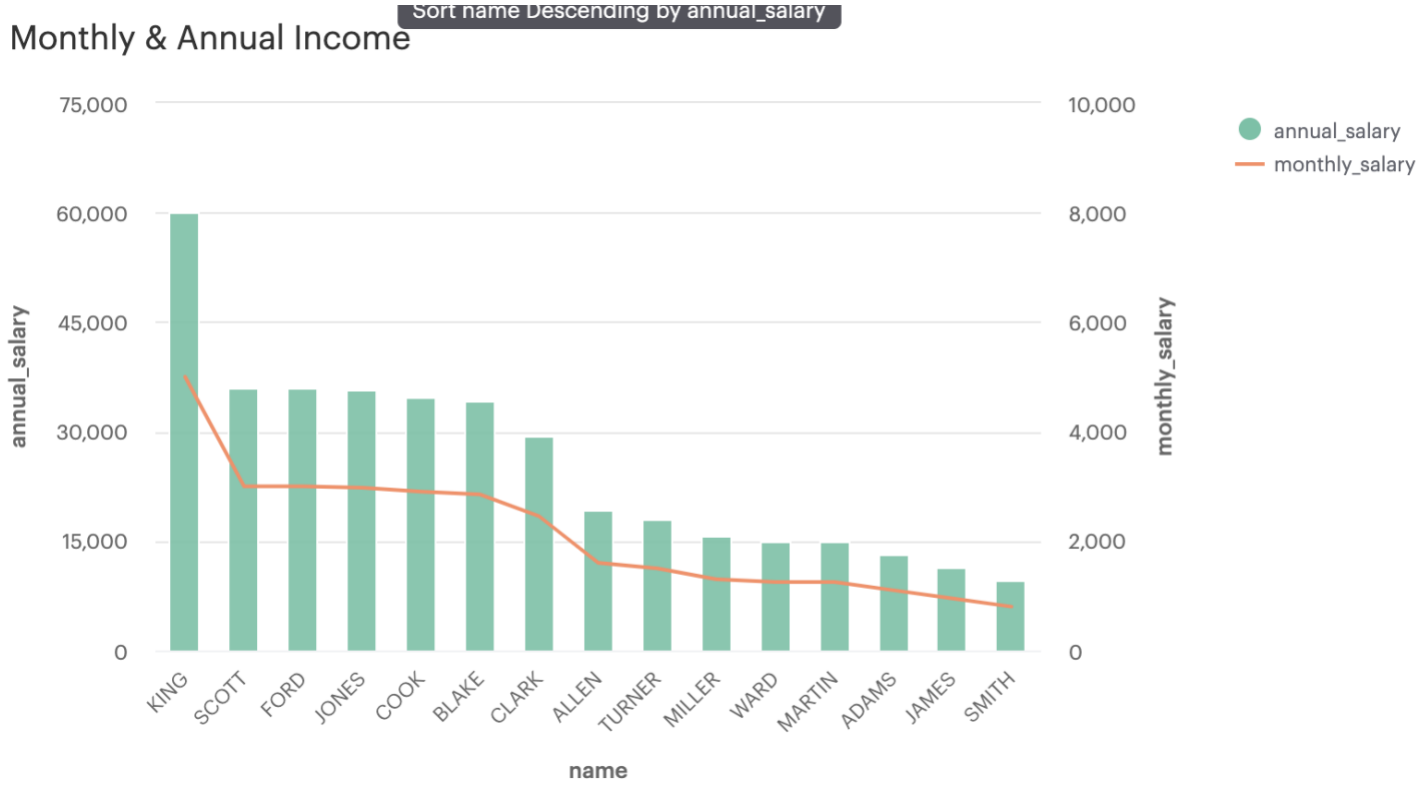
Next we will be working with Employment data that I received from my DS program at Thinkful.

Periodic Salary:

Daily, Monthly and Annual salary for each employee.

```sql
--Employee Salary/month
SELECT ename as Name, sal as Monthly_Salary, sal/30 as Daily_Salary, sal*12 as Annual_Salary
FROM sajith_gowthaman.sajithempstable
ORDER BY 4 DESC;
```

Daily Salary = Sal/3
Monthly Salary = Sal
Annual Salary = Sal*12



Employment Classification (Levels):

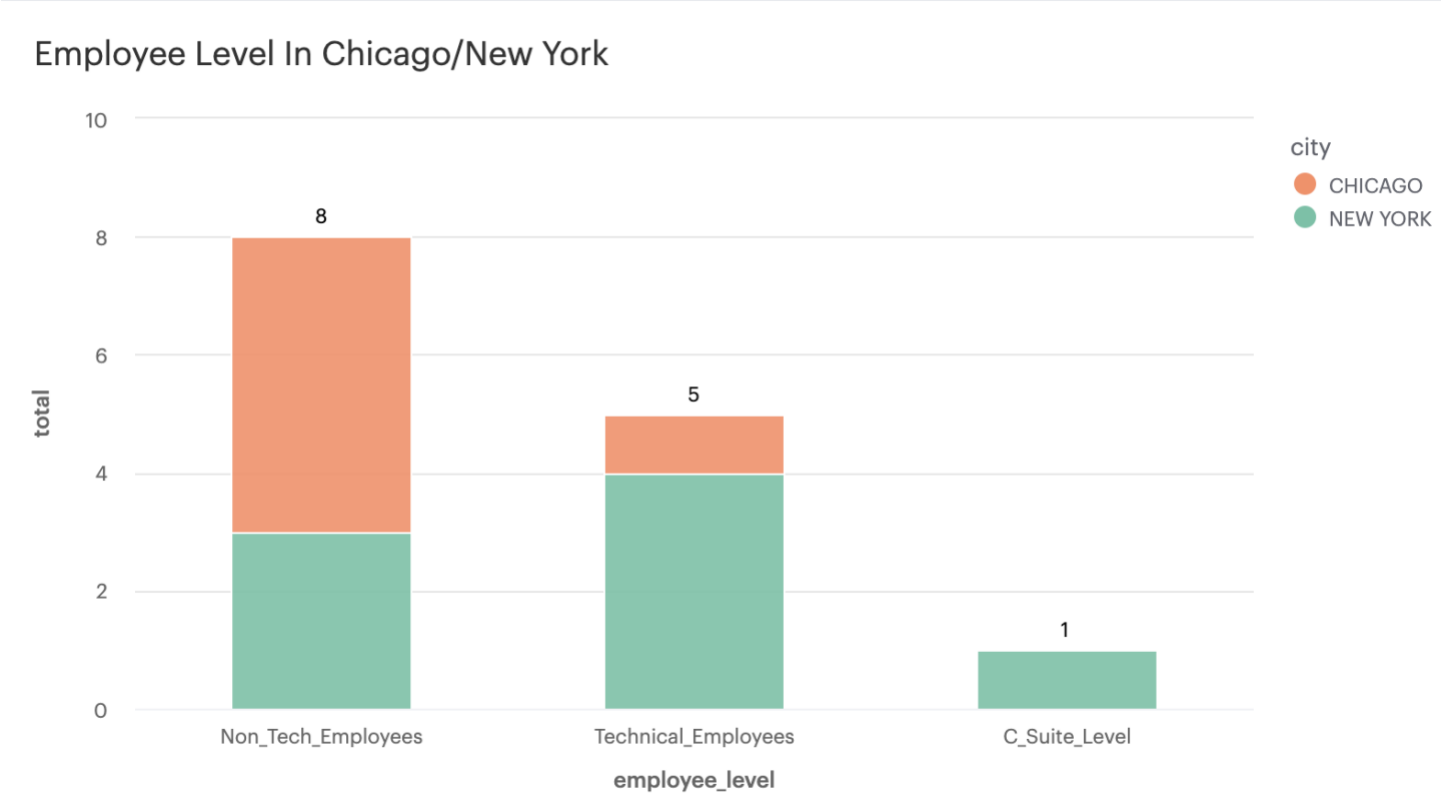
Let's Classify employees as technical, non-technical and C-level based on their job for employees working in New York.

Tables:
sajithsemptable (contains employee information)
sajithdept (contains department information)

common key column: deptno

```sql
SELECT CASE WHEN emp.job IN ('CLERK','SALESMAN') THEN 'Non_Tech_Employees'
            WHEN emp.job IN ('MANAGER', 'ANALYST') THEN 'Technical_Employees'
            WHEN emp.job = 'PRESIDENT' THEN 'C_Suite_Level'
            ELSE 'Others' END AS Employee_Level,
       loc.cty as City,
       COUNT (1) as Total
FROM sajith_gowthaman.sajithempstable as emp
JOIN sajith_gowthaman.sajithdept as dept ON emp.deptno = dept.deptno
JOIN sajith_gowthaman.sajithloc as loc ON dept.locno = loc.locno
WHERE loc.cty = 'NEW YORK' or loc.cty = 'CHICAGO'
GROUP BY 1,2
```

We are joining two tables on one common column, i.e deptno.

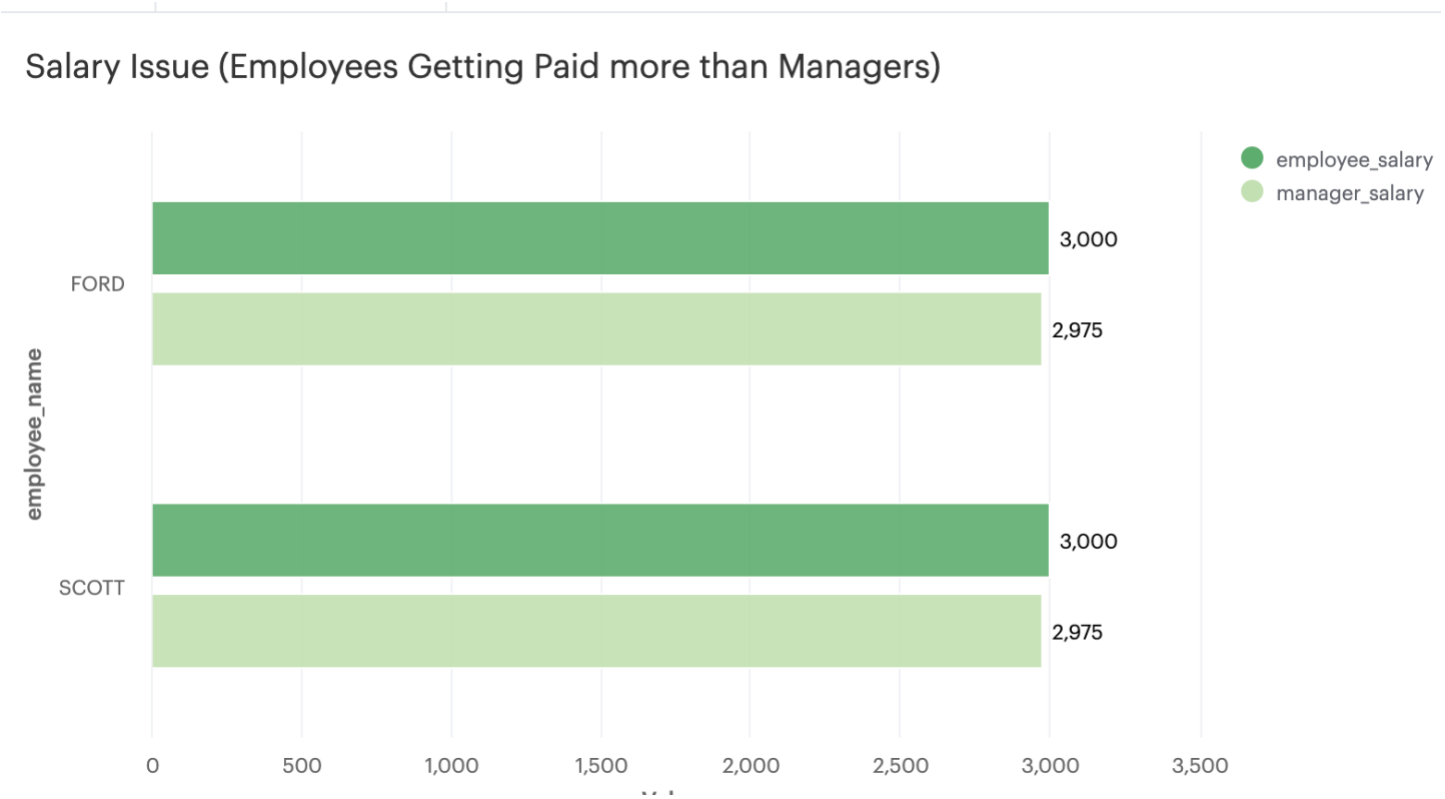


Days/Years of Experience:

```sql
--Experience of Employees
Select ename,
DATE_TRUNC('DAY', NOW()) - DATE_TRUNC('DAY'   , hiredate) AS days_of_experience,
(DATE_TRUNC('DAY', NOW()) - DATE_TRUNC('DAY'   , hiredate))/365 AS yrs_of_experience

from sajith_gowthaman.sajithempstable

```

Employees Earning More Than Their Managers:

```sql
select employees.ename as employee_name, employees.sal as employee_salary,
manager.ename as manager_name, manager.sal as manager_salary

From sajith_gowthaman.sajithempstable as employees
INNER Join sajith_gowthaman.sajithempstable as manager

ON employees.mgr = manager.empno
And employees.sal > manager.sal

```

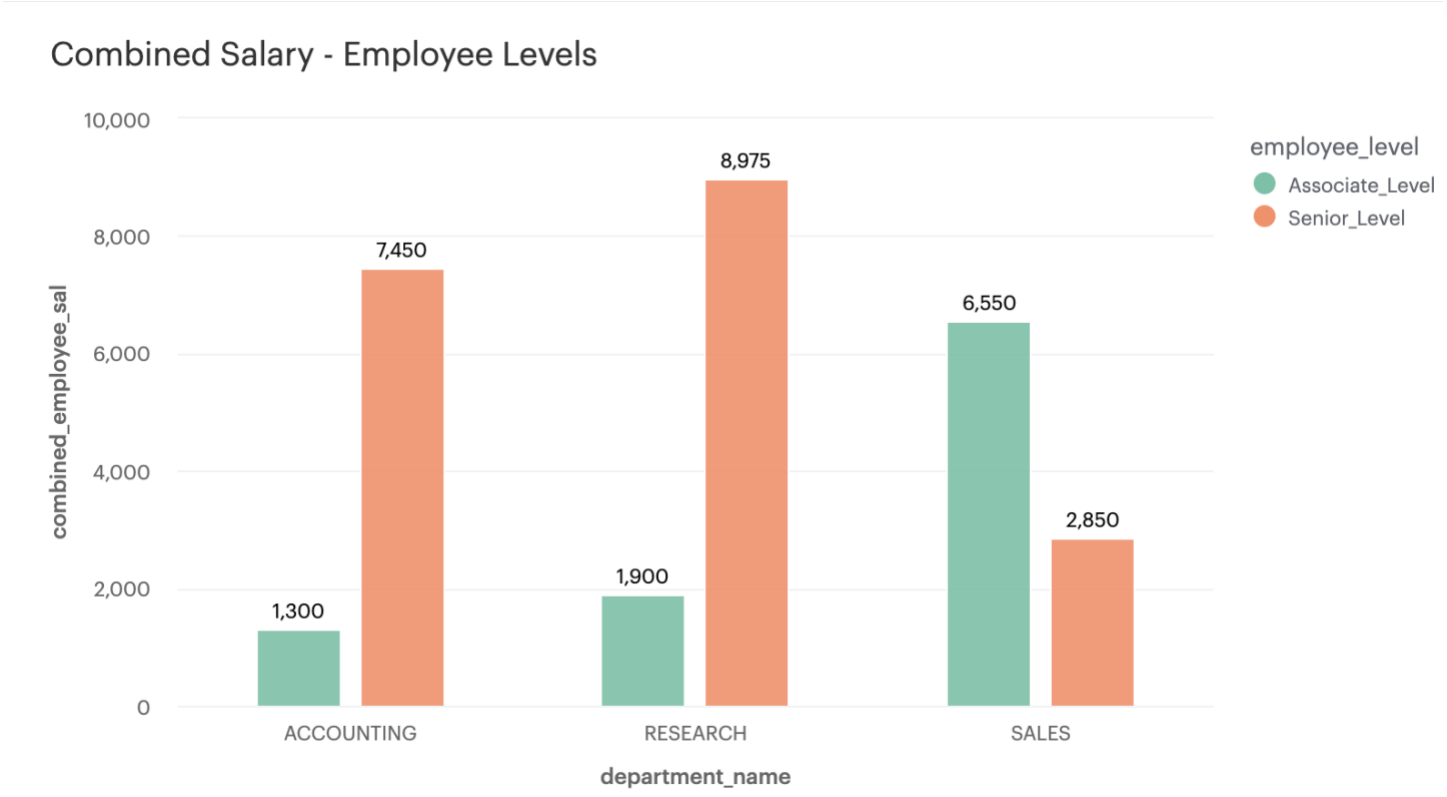


Combined Salary:

Run | Limit 100 | Format SQL | View History...

```sql
--Combined Salary

SELECT CASE WHEN emp.job IN ('SALESMAN', 'CLERK') THEN 'Associate_Level'
            WHEN emp.job IN ('ANALYST', 'MANAGER', 'PRESIDENT') THEN 'Senior_Level'
            ELSE NULL END AS Employee_Level,
       dept.dname as Department_Name,
       SUM(emp.sal) AS combined_Employee_Sal
FROM sajith_gowthaman.sajithempstable as emp
JOIN sajith_gowthaman.sajithdept as dept ON emp.deptno = dept.deptno
GROUP BY 1,2
ORDER BY 3 DESC
```

Selecting a column with employment levels and grouping it by sum of salary.



Now, let's play around with crunchbase data.

Query that lists investors based on the number of companies in which they are invested:

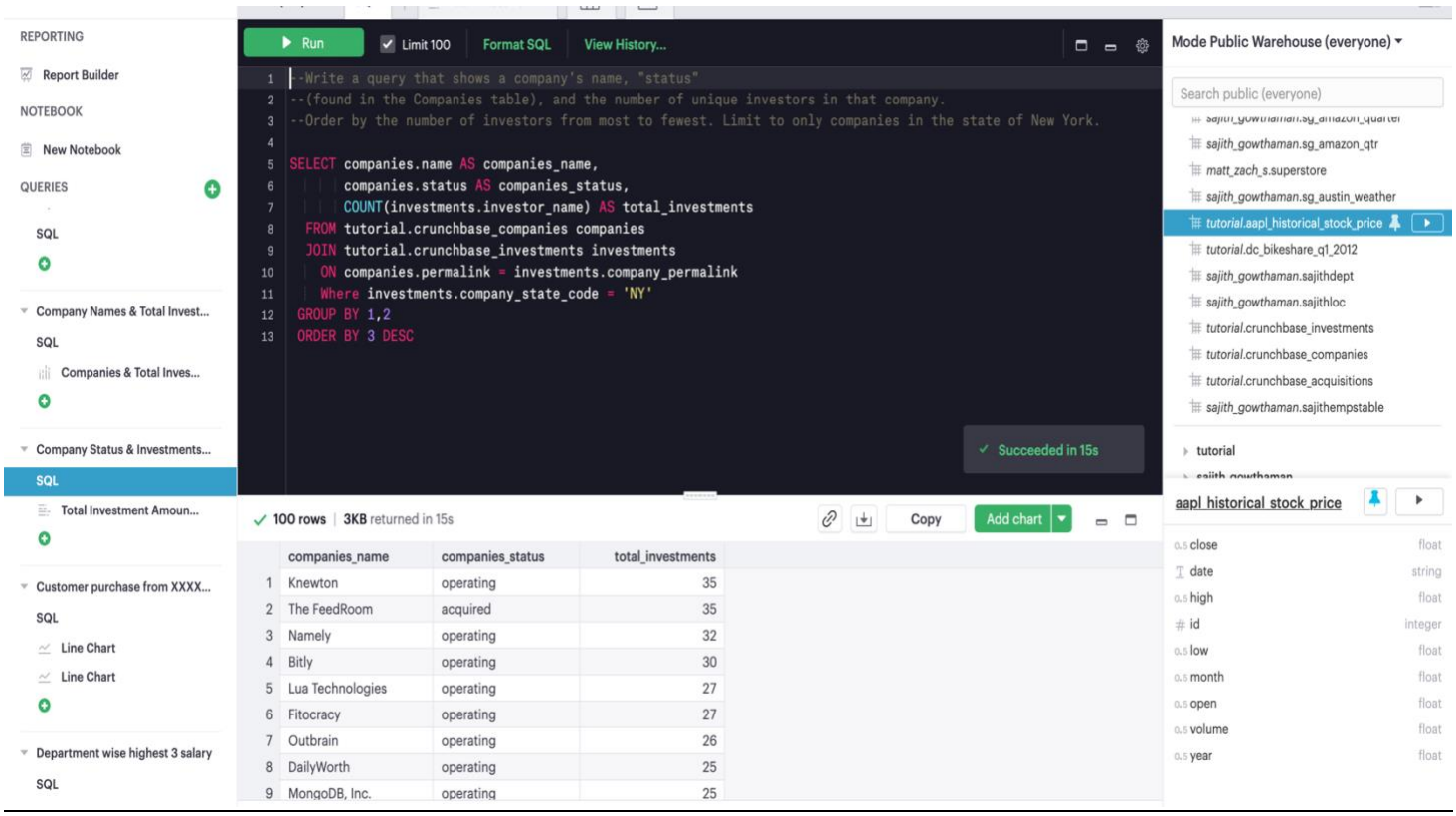
```
--Write a query that lists investors based on the number of companies in which they are invested.
--Include a row for companies with no investor, and order from most companies to least.

SELECT CASE WHEN investments.investor_name IS NULL THEN 'No Investors'
            ELSE investments.investor_name END AS investors,
       COUNT(investments.company_permalink) AS Companies_Invested_In

  FROM tutorial.crunchbase_companies companies
  JOIN tutorial.crunchbase_investments investments
    ON companies.permalink = investments.company_permalink
  GROUP BY investments.investor_name
  ORDER BY 2 DESC
```

Succeed

100 rows | 3KB returned in 3m 6s Copy Add chart

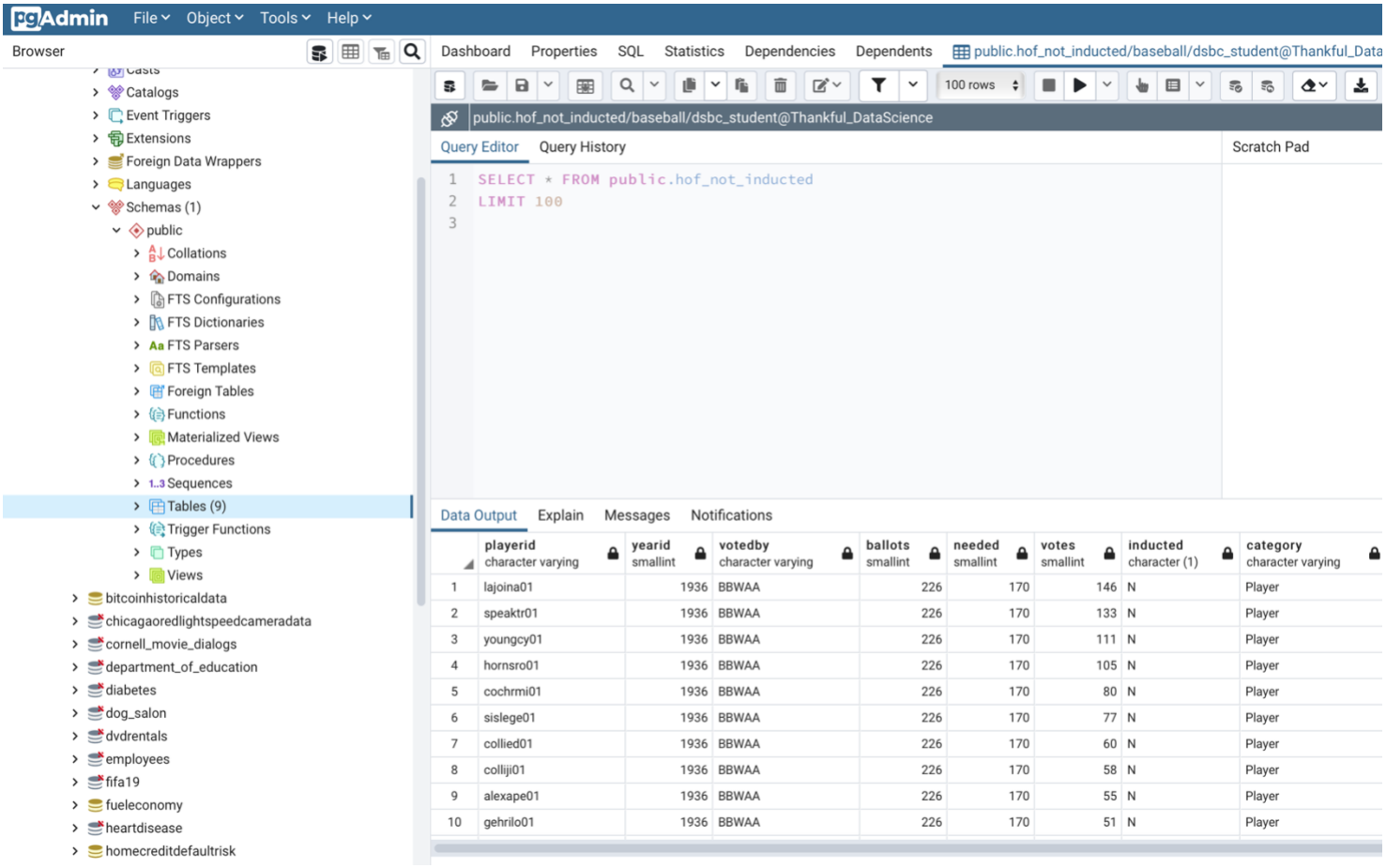
	investors	companies_invested_in
1	Sequoia Capital	462
2	Intel Capital	445
3	New Enterprise Associates	415
4	Accel Partners	407
5	Draper Fisher Jurvetson (DFJ)	390
6	SV Angel	387

Query to show company status and total investments:



As a part of my Data Science program at Thinkful, I had three level SQL classes and an examination. I will be posting the codes I used for it.

PgAdmin Interface: (Thinkful Data Science Database Server)



	playerid character varying	yearid smallint	votedby character varying	ballots smallint	needed smallint	votes smallint	inducted character (1)	category character varying
1	lajoina01	1936	BBWAA	226	170	146	N	Player
2	speaktr01	1936	BBWAA	226	170	133	N	Player
3	youngcy01	1936	BBWAA	226	170	111	N	Player
4	hornsro01	1936	BBWAA	226	170	105	N	Player
5	cochrmi01	1936	BBWAA	226	170	80	N	Player
6	sislege01	1936	BBWAA	226	170	77	N	Player
7	collied01	1936	BBWAA	226	170	60	N	Player
8	colliji01	1936	BBWAA	226	170	58	N	Player
9	alexape01	1936	BBWAA	226	170	55	N	Player
10	gehrilo01	1936	BBWAA	226	170	51	N	Player

SQL – Baseball dataset to carry out operations in data bases and basic querying.

```
--2. all namefirst and namelast from people
--along with inducted field from hof_inducted

SELECT namefirst, namelast, inducted
FROM people LEFT OUTER JOIN hof_inducted
ON people.playerid = hof_inducted.playerid;


--3. 2006 Negro League HOF Inductions

SELECT birthyear, deathyear, birthcountry, namefirst, namelast
FROM people LEFT OUTER JOIN hof_inducted
ON people.playerid = hof_inducted.playerid
WHERE yearid = 2006 AND votedby = 'Negro League';


--4. hof_inducted and salaries INNER JOIN
```

```
SELECT salaries.yearid, salaries.playerid, teamid, salary, category
FROM salaries INNER JOIN hof_inducted
ON salaries.playerid = hof_inducted.playerid;


--5. salaries and hof_inducted FULL OUTER JOIN


SELECT salaries.playerid, salaries.yearid, teamid, lgid, salary, inducted
FROM hof_inducted FULL OUTER JOIN salaries
ON hof_inducted.playerid = salaries.playerid;


--6. hof_inducted and hof_inducted UNION

SELECT * FROM hof_inducted
UNION ALL
SELECT * FROM hof_not_inducted;

SELECT playerid FROM hof_inducted
UNION
SELECT playerid FROM hof_not_inducted;

--7. SUM of salaries by name

SELECT
       namelast,
       namefirst,
       SUM(salary) AS total_salary
FROM salaries AS s
INNER JOIN people AS p
ON s.playerid = p.playerid
GROUP BY namelast, namefirst, playerid


--8. namefirst and last with all hof records

SELECT hof_inducted.playerid, yearid, namefirst, namelast
FROM hof_inducted LEFT OUTER JOIN people
ON hof_inducted.playerid = people.playerid

UNION ALL

SELECT hof_not_inducted.playerid, yearid, namefirst, namelast
FROM hof_not_inducted LEFT OUTER JOIN people
ON hof_not_inducted.playerid = people.playerid;


--9. Like 8. but Filtered since 1980 and
--sorted by year and a field "lastname, firstname"


SELECT concat(namelast,', ', namefirst) AS namefull, yearid, inducted
FROM hof_inducted LEFT OUTER JOIN people
ON hof_inducted.playerid = people.playerid
WHERE yearid >= 1980
```

```
UNION ALL

SELECT concat(namelast,', ', namefirst) AS namefull, yearid, inducted
FROM hof_not_inducted LEFT OUTER JOIN people
ON hof_not_inducted.playerid = people.playerid
WHERE yearid >= 1980

ORDER BY yearid,  inducted DESC, namefull;

--10. Return a table containing the highest annual salary
-- for each teamid, ranked high to low along with the
-- matching playerid.
-- BONUS! In addition to playerid, return namelast
-- and namefirst in this table (These are in the people table.).

WITH max AS
(SELECT MAX(salary) as max_salary, teamid, yearid
FROM salaries
GROUP BY teamid, yearid)
SELECT salaries.yearid, salaries.teamid, playerid, max.max_salary
FROM max LEFT OUTER JOIN salaries
ON salaries.teamid = max.teamid AND salaries.yearid = max.yearid AND salaries.salary
= max.max_salary
ORDER BY max.max_salary DESC;
```

SQL Exam on Dept. Of Education Data:

```
1) Write a query that allows you to inspect the schema of the naep table.

SELECT data_type
FROM information_schema.columns
WHERE table_name = 'naep';

2) Write a query that returns the first 50 records of the naep table.

SELECT *
FROM naep LIMIT 50;

3) Write a query that returns summary statistics for avg_math_4_score by state. Make
sure to sort the results alphabetically by state name.

SELECT AVG(avg_math_4_score), COUNT(avg_math_4_score), MIN(avg_math_4_score),
MAX(avg_math_4_score)
FROM naep
GROUP BY naep.state
ORDER BY naep.state ASC;

4) Write a query that alters the previous query so that it returns only the summary
statistics for avg_math_4_score by state with differences in max and min values that
are greater than 30.
```

```
SELECT AVG(avg_math_4_score), COUNT(avg_math_4_score), MIN(avg_math_4_score),
MAX(avg_math_4_score)
FROM naep
GROUP BY naep.state
HAVING MAX(avg_math_4_score) - MIN(avg_math_4_score) > 30
ORDER BY naep.state ASC;
```

5)Write a query that returns a field called bottom_10_states that lists the states in the bottom 10 for avg_math_4_score in the year 2000.

```
SELECT state AS bottom_10_states
FROM naep
WHERE YEAR = 2000
AND avg_math_4_score IS NOT NULL
ORDER BY avg_math_4_score DESC
LIMIT 10;
```

7) Write a query that returns a field called below_average_states_y2000 that lists all states with an avg_math_4_score less than the average over all states in the year 2000.

```
WITH average_2000 AS
  (
   SELECT
   naep.state AS states,
   AVG(avg_math_4_score) AS averages
   FROM naep
   WHERE naep.year = 2000
   GROUP BY naep.state
)

SELECT states AS below_average_states_y2000
FROM average_2000
WHERE averages < (SELECT AVG(avg_math_4_score)
FROM naep
WHERE naep.year = 2000);
```

9) Write a query that returns for the year 2000 the state, avg_math_4_score, and total_expenditure from the naep table left outer joined with the finance table, using id as the key and ordered by total_expenditure greatest to least.
Be sure to round avg_math_4_score to the nearest 2 decimal places, and then filter out NULL avg_math_4_scores in order to see any correlation more clearly.

```
SELECT naep.state, (ROUND(avg_math_4_score),2) AS AVG_MATH_4_SCORE,
finance.total_expenditure
FROM naep LEFT OUTER JOIN finance
ON naep.id=finance.id
WHERE naep.year = 2000
AND avg_math_4_score IS NOT NULL
ORDER BY finance.total_expenditure DESC;
```

Linear Regression Using SQL:

I came across an article from Silota (BI tool) about fitting a linear regression model into SQL and checking for the linearity or the model (if the data points lie close to the linear plane).

The formula for linear regression is Y = mx + b

B is the intercept, m is the slope, y is the target variable (dependent variable) and x is the Independent variable from which the predictions are usuall

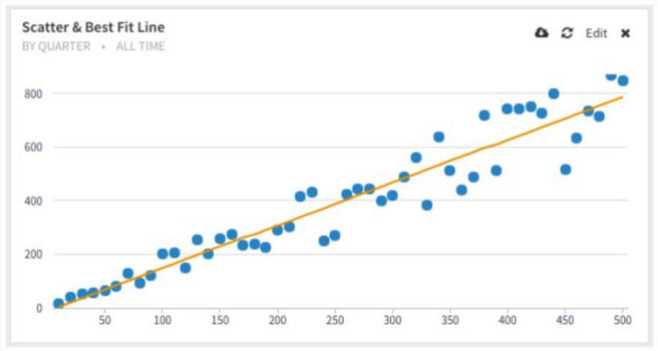
$$m = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{\sum(x_i - \bar{x})^2}$$

$$b = \bar{y} - m\bar{x}$$

where $\bar{y}$ and $\bar{x}$ are the averages for x and y.

```
select slope,
       y_bar_max - x_bar_max * slope as intercept
from (
    select sum((x - x_bar) * (y - y_bar)) / sum((x - x_bar) * (x - x_bar)) as slope,
           max(x_bar) as x_bar_max,
           max(y_bar) as y_bar_max
    from (
        select x, avg(x) over () as x_bar,
                  y, avg(y) over () as y_bar
        from ols) s;
```

Visualizing the Regression line



THANK YOU!