P2 Assignment: Analytical SQL

Query 1: Computing RANKs

Assign a RANK and DENSE_RANK to each year based on the federal funds rate (using the FRED_FEDFUNDS data) or a data set of your choice. How does the RANK differ from the DENSE_RANK assigned in the query?

Ans:

FRED_FEDFUNDS Data:

```sql
select * from FRED_FEDFUNDS;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.007 seconds

	FF_DATE	FF_RATE	FF_YEAR	CAL_DATE_KEY	CAL_MONTH_KEY
1	01-07-60	3.23	1960	19600701	196007
2	01-12-60	1.98	1960	19601201	196012
3	01-04-61	1.49	1961	19610401	196104
4	01-08-61	2	1961	19610801	196108
5	01-12-61	2.33	1961	19611201	196112
6	01-04-62	2.78	1962	19620401	196204
7	01-08-62	2.93	1962	19620801	196208
8	01-01-63	2.92	1963	19630101	196301
9	01-05-63	3	1963	19630501	196305
10	01-09-63	3.48	1963	19630901	196309
11	01-01-64	3.48	1964	19640101	196401
12	01-09-64	3.45	1964	19640901	196409
13	01-04-65	4.09	1965	19650401	196504
14	01-01-66	4.42	1966	19660101	196601
15	01-09-66	5.4	1966	19660901	196609
16	01-04-67	4.05	1967	19670401	196704

Rank and Dense Rank:

The below query will provide a yearly breakdown of the Federal Funds Rate, ranked by the average rate for each year, showcasing the differences between RANK and DENSE_RANK in handling ties.

```sql
SELECT
    FF_YEAR,
    AVG(FF_RATE) AS AvgFedFundsRate,
    RANK() OVER (ORDER BY AVG(FF_RATE) DESC) AS Rank,
    DENSE_RANK() OVER (ORDER BY AVG(FF_RATE) DESC) AS DenseRank
FROM
    FRED_FEDFUNDS
GROUP BY
    FF_YEAR
ORDER BY
    AVG(FF_RATE) DESC;
```

Script Output × | Query Result ×

SQL | Fetched 50 rows in 0.008 seconds

	FF_YEAR	AVGFEDFUNDSRATE	RANK	DENSERANK
1	1981	16.37833333333333333333333333333333333333	1	1
2	1980	13.35583333333333333333333333333333333333	2	2
3	1982	12.25833333333333333333333333333333333333	3	3
4	1979	11.19416666666666666666666666666666666667	4	4
5	1974	10.5025	5	5
6	1984	10.225	6	6
7	1989	9.21666666666666666666666666666666666667	7	7
8	1983	9.08666666666666666666666666666666666667	8	8
9	1973	8.7275	9	9
10	1969	8.20416666666666666666666666666666666667	10	10
11	1985	8.10083333333333333333333333333333333333	11	11

FF_YEAR: Groups the data by year.

AVG(FF_RATE) AS AvgFedFundsRate: Calculates the average Federal Funds Rate for each year.

RANK() OVER (ORDER BY AVG(FF_RATE) DESC) AS Rank: Assigns a rank to each year based on the average Federal Funds Rate, with gaps in the ranking for ties.

DENSE_RANK() OVER (ORDER BY AVG(FF_RATE) DESC) AS DenseRank: Similar to RANK(), but without gaps for ties, ensuring a continuous sequence of rank numbers.

Difference between Rank and Dense Rank:

The key difference between the RANK and DENSE_RANK functions in SQL is how they handle duplicate values. The RANK function may skip positions after equal rankings, resulting in gaps in the ranking, while the DENSE_RANK function does not skip any positions, ensuring there are no gaps in the ranking.

Query 2: Creating Bins with NTILE

It is often useful to assign bins or otherwise categorize data in a warehouse environment with functions like NTILE.

Use the NTILE function to bin indicator values, such as the federal funds rate (FRED_FEDFUNDS) or data set of your choice using quartiles or even deciles.

Group the data by year or decade and use NTILE within the partitions to assign values.

Ans: To use the NTILE function for binning the federal funds rate (FF_RATE) into quartiles within each year, we can modify the query to partition the data by FF_YEAR and then apply NTILE(4) to create four bins (quartiles) for each year. We can do create 10 bins and do the partition with deciles as well.

For Quartiles:

```sql
SELECT
    FF_YEAR,
    FF_RATE,
    NTILE(4) OVER (PARTITION BY FF_YEAR ORDER BY FF_RATE DESC) AS Quartile
FROM
    FRED_FEDFUNDS
ORDER BY
    FF_YEAR, FF_RATE DESC;
```

Query Result ×

SQL | Fetched 50 rows in 0.105 seconds

	FF_YEAR	FF_RATE	QUARTILE
1	1954	1.28	1
2	1954	1.22	1
3	1954	1.06	2
4	1954	0.85	2
5	1954	0.83	3
6	1954	0.8	4
7	1955	2.48	1
8	1955	2.35	1
9	1955	2.24	1
10	1955	2.18	2
11	1955	1.96	2

For Deciles:

```sql
SELECT
    FF_YEAR,
    FF_RATE,
    NTILE(10) OVER (PARTITION BY FF_YEAR ORDER BY FF_RATE DESC) AS Decile
FROM
    FRED_FEDFUNDS
ORDER BY
    FF_YEAR, FF_RATE DESC;
```

Query Result ×

SQL | Fetched 50 rows in 0.035 seconds

	FF_YEAR	FF_RATE	DECILE
1	1954	1.28	1
2	1954	1.22	2
3	1954	1.06	3
4	1954	0.85	4
5	1954	0.83	5
6	1954	0.8	6
7	1955	2.48	1
8	1955	2.35	1
9	1955	2.24	2
10	1955	2.18	2
11	1955	1.96	3

Query 3: Correlations (CORR)

The correlations between different economic indicators are often important. The whole idea of diversification is based on holding investments that are somewhat uncorrelated, so their values do not move in lock step. A diversified portfolio should do better under a variety of economic conditions, thereby reducing the risk of large swings in value.

Use analytic SQL to assess the correlation between some selected stocks within and across sectors.

Assess some correlations on other financial data.

Ans:

Aggregate Stock Performance by Sector and Year

First, aggregate the average closing prices of stocks within each GICS sector for each year. This will serve as a foundation for assessing correlations across sectors.

```sql
SELECT
    stk.GICS_SECTOR,
    EXTRACT(YEAR FROM eod.TRADE_DATE) AS YEAR,
    AVG(eod.CLOSE) AS AVG_CLOSE
FROM
    SP500_EOD_STOCK_FACTS eod
JOIN
    SP500_STOCKS stk ON eod.TICKER_SYMBOL = stk.TICKER_SYMBOL
GROUP BY
    stk.GICS_SECTOR,
    EXTRACT(YEAR FROM eod.TRADE_DATE)
ORDER BY
    stk.GICS_SECTOR,
    YEAR;
```

Query Result ×

SQL | All Rows Fetched: 22 in 0.095 seconds

	GICS_SECTOR	YEAR	AVG_CLOSE
1	Consumer Discretionary	2009	40.85990956687291765825797239409804854831
2	Consumer Discretionary	2010	45.03609092417454786792765886842541894807
3	Consumer Staples	2009	39.67461493065801121274712304514606078489
4	Consumer Staples	2010	41.81754678408554804214499135084132725271
5	Energy	2009	46.2827408637873754152823920265780730897
6	Energy	2010	46.75434048540274358072458670418571931059
7	Financials	2009	36.58870516445899687453490102693853251972

Comparative Analysis for Correlation

Incorporating Other Financial Data

To incorporate energy prices or political party data, we can join these tables on the year and compare trends. For example, comparing average stock prices in the Energy sector with crude oil prices from ENERGY_PRICE.

```
SELECT
    stk.GICS_SECTOR,
    EXTRACT(YEAR FROM eod.TRADE_DATE) AS YEAR,
    AVG(eod.CLOSE) AS AVG_STOCK_CLOSE,
    AVG(ep.CRUDE_OIL) AS AVG_CRUDE_OIL_PRICE
FROM
    SP500_EOD_STOCK_FACTS eod
JOIN
    SP500_STOCKS stk ON eod.TICKER_SYMBOL = stk.TICKER_SYMBOL
JOIN
    ENERGY_PRICES ep ON EXTRACT(YEAR FROM eod.TRADE_DATE) = EXTRACT(YEAR FROM ep.SQL_DATE)
WHERE
    stk.GICS_SECTOR = 'Energy'
GROUP BY
    stk.GICS_SECTOR,
    EXTRACT(YEAR FROM eod.TRADE_DATE)
ORDER BY
    YEAR;
```

Query Result

All Rows Fetched: 2 in 0.651 seconds

	GICS_SECTOR	YEAR	AVG_STOCK_CLOSE	AVG_CRUDE_OIL_PRICE
1	Energy	2009	46.282740863787375415282392026578073089 7	61.950436507936507936507936507936507936 51
2	Energy	2010	46.754340485402743580724586704185719310 59	79.475714285714285714285714285714285714 29

Query 4: Leading and Lagging Indicators

Economic indicators are often characterized as “leading or lagging” indicators. We can use analytic SQL to investigate the temporal relationships between indicators, especially using the LAG/LEAD functions.

For instance, the stock market is often considering a leading indicator of more general economic activity. You might look at a stock index such as the S&P 500 (SP500_EOD_FACTS) or the DJIA (DBERNDT.DJIA_WEEKLY_FACTS) and compare that with a general measure like Gross Domestic Product (FRED_GDP).

Investigate other leading and lagging indicator relationships.

Ans:

Quarterly Aggregation of S&P 500 Index

First, we aggregate the S&P 500 index data on a quarterly basis to match the reporting period of GDP data.SP500_EOD_FACTS contains daily closing values, we can aggregate these values to get an average or closing value for each quarter.

```sql
SELECT
    EXTRACT(YEAR FROM TRADE_DATE) AS YEAR,
    CEIL(EXTRACT(MONTH FROM TRADE_DATE) / 3) AS QUARTER,
    AVG(CLOSE) AS AVG_SP500_CLOSE
FROM
    DW169.SP500_EOD_FACTS
GROUP BY
    EXTRACT(YEAR FROM TRADE_DATE),
    CEIL(EXTRACT(MONTH FROM TRADE_DATE) / 3)
ORDER BY
    YEAR,
    QUARTER;
```

ript Output × | Query Result ×

SQL | Fetched 50 rows in 0.054 seconds

	YEAR	QUARTER	AVG_SP500_CLOSE
1	1950	1	17.142419354838709677419354838709677419354838709677419354838709677
2	1950	2	18.36238095238095238095238095238095238095
3	1950	3	18.3068253968253968253968253968253968254
4	1950	4	19.8021311475409836065573770491803278688
5	1951	1	21.5752459016393442622950819672131147541
6	1951	2	21.79203125
7	1951	3	22.7506349206349206349206349206349206349
8	1951	4	23.1816393442622950819672131147540983607

Analyzing GDP Data with S&P 500 Performance

Next, joining this aggregated S&P 500 data with the FRED_GDP table on the year and quarter to analyze their relationship. We can use the LAG or LEAD functions to compare the quarter-over-quarter changes in GDP with the S&P 500 performance.

Here's the below query:

```sql
WITH AnnualSP500 AS (
  SELECT
    EXTRACT(YEAR FROM TRADE_DATE) AS YEAR,
    AVG(CLOSE) AS AVG_CLOSE
  FROM
    DW169.SP500_EOD_FACTS
  GROUP BY
    EXTRACT(YEAR FROM TRADE_DATE)
),
AnnualGDP AS (
  SELECT
    EXTRACT(YEAR FROM GDP_DATE) AS YEAR,
    AVG(GDP_VALUE) AS AVG_GDP
  FROM
    DW169.FRED_GDP
  GROUP BY
    EXTRACT(YEAR FROM GDP_DATE)
)
SELECT
  sp.YEAR,
  sp.AVG_CLOSE AS SP500_AVG_CLOSE,
```

```
 gd.AVG_GDP AS GDP_AVG,
 sp.AVG_CLOSE - LAG(sp.AVG_CLOSE) OVER (ORDER BY sp.YEAR) AS SP500_YOY_CHANGE,
 gd.AVG_GDP - LAG(gd.AVG_GDP) OVER (ORDER BY gd.YEAR) AS GDP_YOY_CHANGE
FROM
 AnnualSP500 sp
JOIN
 AnnualGDP gd ON sp.YEAR = gd.YEAR
ORDER BY
 sp.YEAR;
```

	YEAR	SP500_AVG_CLOSE	GDP_AVG	SP500_YOY_CHANGE	GDP_YOY_CHANGE
1	1950	18.397269076305220883534136546184738955582	293.725	(null)	(null)
2	1951	22.321887550200803212851405622489959839936	339.25	3.9246184738955823293172690763052208835 4	45.525
3	1952	24.49616	358.3	2.1742724497991967871485943775100401606 4	19.05
4	1953	24.722589641434262948207171314741035856 57	379.35	0.2264296414342629482071713147410358565 7	21.05
5	1954	29.724087301587301587301587301587301587 3	380.35	5.0014976601530386390944159868462657307 3	1
6	1955	40.498849206349206349206349206349206349 21	414.725	10.774761904761904761904761904761904761 91	34.375
7	1956	46.639521912350597609561752988047808764 94	437.45	6.1406727060013912603554037816986024157 3	22.725
8	1957	44.423373015873015873015873015873015873 02	461.075	-2.2161488964775817365458799721747928919 2	23.625
9	1958	46.203452380952380952380952380952380952 38	467.15	1.7800793650793650793650793650793650793 6	6.075
10	1959	57.418181818181818181818181818181818181 82	506.625	11.214729437229437229437229437229437229 44	39.475
11	1960	55.845753968253968253968253968253968253 97	526.475	-1.5724278499278499278499278499278499278 5	19.85
12	1961	66.26632	544.775	10.420566031746031746031746031746031746 03	18.3
13	1962	62.320753968253968253968253968253968253 97	585.65	-3.9455660317460317460317460317460317460 3	40.875
14	1963	69.859362549800796812749003984063745019 92	617.775	7.5386085815468285587807500158097767659 5	32.125

Annual Aggregation: This approach simplifies the analysis by focusing on the annual average closing price of the S&P 500 and the annual GDP, avoiding the complexities of quarterly data.

Year-over-Year Change: By using the LAG function, this query calculates the year-over-year change in both the S&P 500 average closing prices and GDP values. It allows for a simplified comparison of how changes in the stock market might relate to changes in the broader economy without detailed correlation analysis.

Additional Interesting Queries

Query 1: Sector Performance Comparison

Description: Compare the annual average performance of two distinct sectors within the S&P 500 to understand how different sectors react to economic conditions.

```sql
SELECT
    EXTRACT(YEAR FROM eod.TRADE_DATE) AS YEAR,
    stk.GICS_SECTOR,
    AVG(eod.CLOSE) AS AVG_CLOSE_PRICE
FROM
    DW169.SP500_EOD_STOCK_FACTS eod
JOIN
    DW169.SP500_STOCKS stk ON eod.TICKER_SYMBOL = stk.TICKER_SYMBOL
WHERE
    stk.GICS_SECTOR IN ('Technology', 'Health Care')
GROUP BY
    EXTRACT(YEAR FROM eod.TRADE_DATE),
    stk.GICS_SECTOR
ORDER BY
    YEAR, GICS_SECTOR;
```

ery Result × | Query Result 1 ×

SQL | All Rows Fetched: 2 in 0.058 seconds

	YEAR	GICS_SECTOR	AVG_CLOSE_PRICE
1	2009	Health Care	47.74326375245579567779960707269155206287
2	2010	Health Care	51.3959609375

Query 2: GDP Growth vs. Stock Market Volume

Description: Analyze the relationship between GDP growth and stock market trading volume on an annual basis to see if higher economic growth correlates with increased trading activity.

```sql
WITH AnnualGDPGrowth AS (
  SELECT
    EXTRACT(YEAR FROM GDP_DATE) AS YEAR,
    AVG(GDP_VALUE) AS AVG_GDP,
    AVG(GDP_VALUE) - LAG(AVG(GDP_VALUE)) OVER (ORDER BY EXTRACT(YEAR FROM GDP_DATE)) AS GDP_GROWTH
  FROM
    DW169.FRED_GDP
  GROUP BY
    EXTRACT(YEAR FROM GDP_DATE)
) SELECT
    EXTRACT(YEAR FROM eod.TRADE_DATE) AS YEAR,
    AVG(eod.VOLUME) AS AVG_VOLUME,
    gdp.GDP_GROWTH FROM
    DW169.SP500_EOD_STOCK_FACTS eod
JOIN  AnnualGDPGrowth gdp ON EXTRACT(YEAR FROM eod.TRADE_DATE) = gdp.YEAR
GROUP BY EXTRACT(YEAR FROM eod.TRADE_DATE), gdp.GDP_GROWTH
ORDER BY YEAR;
```

ery Result × | Query Result 1 ×

SQL | All Rows Fetched: 2 in 0.087 seconds

	YEAR	AVG_VOLUME	GDP_GROWTH
1	2009	77206.60975326964103515443836378814581 21	-317.9
2	2010	83668.48638137193203272498426683448709 88	525.275

Query 3: Identifying Top Performing Stocks

Description: Identify the top 5 performing stocks in terms of yearly price increase within the technology sector.

```sql
SELECT
    pol.CONGREE_YEAR AS ELECTION_YEAR,
    AVG(eod.CLOSE) AS AVG_CLOSE_PRICE,
    'Year Before Election' AS TIME_FRAME FROM
    DW169.SP500_EOD_STOCK_FACTS eod JOIN
    DW169.SP500_DATES dt ON eod.JULIAN_DAY = dt.JULIAN_DAY
JOIN FIN.POLITICAL_PARTIES pol ON EXTRACT(YEAR FROM dt.ACTUAL_DATE) = pol.CONGREE_YEAR - 1
GROUP BY pol.CONGREE_YEAR
UNION ALL
SELECT
    pol.CONGREE_YEAR AS ELECTION_YEAR,
    AVG(eod.CLOSE) AS AVG_CLOSE_PRICE,
    'Year After Election' AS TIME_FRAME
FROM DW169.SP500_EOD_STOCK_FACTS eod
JOIN DW169.SP500_DATES dt ON eod.JULIAN_DAY = dt.JULIAN_DAY
JOIN FIN.POLITICAL_PARTIES pol ON EXTRACT(YEAR FROM dt.ACTUAL_DATE) = pol.CONGREE_YEAR + 1
GROUP BY pol.CONGREE_YEAR
ORDER BY ELECTION_YEAR, TIME_FRAME;
```

ery Result × | Query Result 1 ×

SQL | All Rows Fetched: 2 in 0.114 seconds

	ELECTION_YEAR	AVG_CLOSE_PRICE	TIME_FRAME
1	2009	43.80970522341095028319697923222152297042	Year After Election
2	2011	43.80970522341095028319697923222152297042	Year Before Election

Query 4: Analysing Stock Market Volatility by Month

Description: This query calculates the monthly volatility of the S&P 500 by measuring the standard deviation of daily closing prices. High volatility may indicate uncertain market conditions or investor sentiment.

```sql
SELECT
    EXTRACT(YEAR FROM TRADE_DATE) AS YEAR,
    EXTRACT(MONTH FROM TRADE_DATE) AS MONTH,
    STDDEV(CLOSE) AS MONTHLY_VOLATILITY
FROM
    DW169.SP500_EOD_FACTS
GROUP BY
    EXTRACT(YEAR FROM TRADE_DATE),
    EXTRACT(MONTH FROM TRADE_DATE)
ORDER BY
    YEAR, MONTH;
```

uery Result × | Query Result 1 ×

SQL | Fetched 50 rows in 0.049 seconds

	YEAR	MONTH	MONTHLY_VOLATILITY
1	1950	1	0.1323091617603039827915754376107866277515
2	1950	2	0.0951658339306308937293688923403735315203
3	1950	3	0.1555520578062117298877801925680151077951
4	1950	4	0.1500506737017950649474890775521951859999
5	1950	5	0.2276113494686203184596079247650829110812
6	1950	6	0.537670339464056673747992259518469932798
7	1950	7	0.3782007264319890363453470234098686777116
8	1950	8	0.2431317409929457761855282178700048161766
9	1950	9	0.3440084913272681963940715197844267614164
10	1950	10	0.1733630926834252826283077743248244114547