Data Analysis with SQL: NFT sales data

Understanding NFTs data

Over the past 18 months, an emerging technology has caught the attention of the world; the NFT. What is an NFT? They are digital assets stored on the blockchain. And over $22 billion was spent last year on purchasing NFTs. Why? People enjoyed the art, the speculated on what they might be worth in the future, and people didn't want to miss out.

The future of NFT's is unclear as much of the NFT's turned out to be scams of sorts since the field is wildly unregulated. They're also contested heavily for their impact on the environment.

Regardless of these controversies, it is clear that there is money to be made in NFT's. And one cool part about NFT's is that all of the data is recorded on the blockchain, meaning anytime something happens to an NFT, it is logged in this database.

In this project, real-world NFT data has been analyzed.

That data set is a sales data set of one of the most famous NFT projects, Cryptopunks. Meaning each row of the data set represents a sale of an NFT. The data includes sales from January 1st, 2018 to December 31st, 2021. The table has several columns including the buyer address, the ETH price, the price in U.S. dollars, the seller's address, the date, the time, the NFT ID, the transaction hash, and the NFT name. You might not understand all the jargon around the NFT space, but you should be able to infer enough to answer the following prompts.

1. How many sales occurred during this time period?

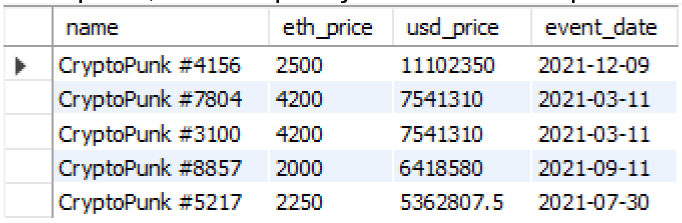
```
SELECT COUNT(*)FROM pricedata;
```

This query counts the number of rows in the pricedata table, effectively giving us the total number of sales. That is 19,920.

2. Return the top 5 most expensive transactions (by USD price) for this data set. Return the name, ETH price, and USD price, as well as the date.

```
SELECT name, eth_price, usd_price, event_date
FROM pricedata
ORDER BY usd_price DESC
LIMIT 5;
```

By ordering the transactions in descending order of their USD price and limiting the results to the top five, we can quickly see the most expensive NFT sales. That is



name	eth_price	usd_price	event_date
CryptoPunk #4156	2500	11102350	2021-12-09
CryptoPunk #7804	4200	7541310	2021-03-11
CryptoPunk #3100	4200	7541310	2021-03-11
CryptoPunk #8857	2000	6418580	2021-09-11
CryptoPunk #5217	2250	5362807.5	2021-07-30

3. **Return a table with a row for each transaction with an event column, a USD price column, and a moving average of USD price that averages the last 50 transactions.**

```
SELECT transaction_hash, usd_price,
AVG(usd_price) OVER(
ORDER BY event_date ROWS BETWEEN 49 PRECEDING AND CURRENT ROW)
as usd_mv_avg
FROM pricedata;
```

This query provides insights into the market trend by averaging the prices over a set number of preceding transactions, offering a clearer picture of the price dynamics in the NFT market.

4. **Return all the NFT names and their average sale price in USD. Sort descending. Name the average column as average_price.**

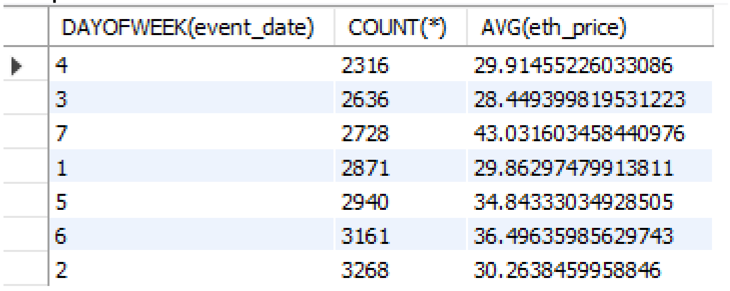
```
SELECT name, AVG(usd_price) as average_price
FROM pricedata
GROUP BY name
ORDER BY average_price DESC;
```

This query groups the sales data by the NFT name and calculates the average USD price for each group. Sorting the results by average_price in descending order highlights the most valuable NFTs based on their average sale price.

5. **Return each day of the week and the number of sales that occurred on that day of the week, as well as the average price in ETH. Order by the count of transactions in ascending order.**

```
SELECT DAYOFWEEK(event_date), COUNT(*), AVG(eth_price)
FROM pricedata
GROUP BY DAYOFWEEK(event_date)
ORDER BY COUNT(*);
```

Output:



DAYOFWEEK(event_date)	COUNT(*)	AVG(eth_price)
4	2316	29.91455226033086
3	2636	28.449399819531223
7	2728	43.031603458440976
1	2871	29.86297479913811
5	2940	34.84333034928505
6	3161	36.49635985629743
2	3268	30.2638459958846

6. **Construct a column that describes each sale and is called summary. The sentence should include who sold the NFT name, who bought the NFT, who sold the NFT, the date, and what price it was sold for in USD rounded to the nearest thousandth.**

```
SELECT (CONCAT(name,' was sold for $',ROUND(usd_price, -3),' to ',
seller_address,' from ',buyer_address,' on ',event_date)) as summary
FROM pricedata;
```

- This query concatenates various fields (name, usd_price, seller_address, buyer_address, and event_date) into a single string.
- Rounds the usd_price to the nearest thousand for a cleaner look.
- Labels the result as summary for each transaction.

These summaries offer a straightforward, human-readable format, making it easier to quickly understand the specifics of each transaction. This can be particularly useful for presentations, reports, or simply for a more engaging way to explore the data.

Here’s an example summary:

```
“CryptoPunk #1139 was sold for $194000 to
0x91338ccfb8c0adb7756034a82008531d7713009d from
0x1593110441ab4c5f2c133f21b0743b2b43e297cb on 2022-01-14”
```

7. “*1919_purchases*” and contains any sales where “*0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685*” was the buyer.

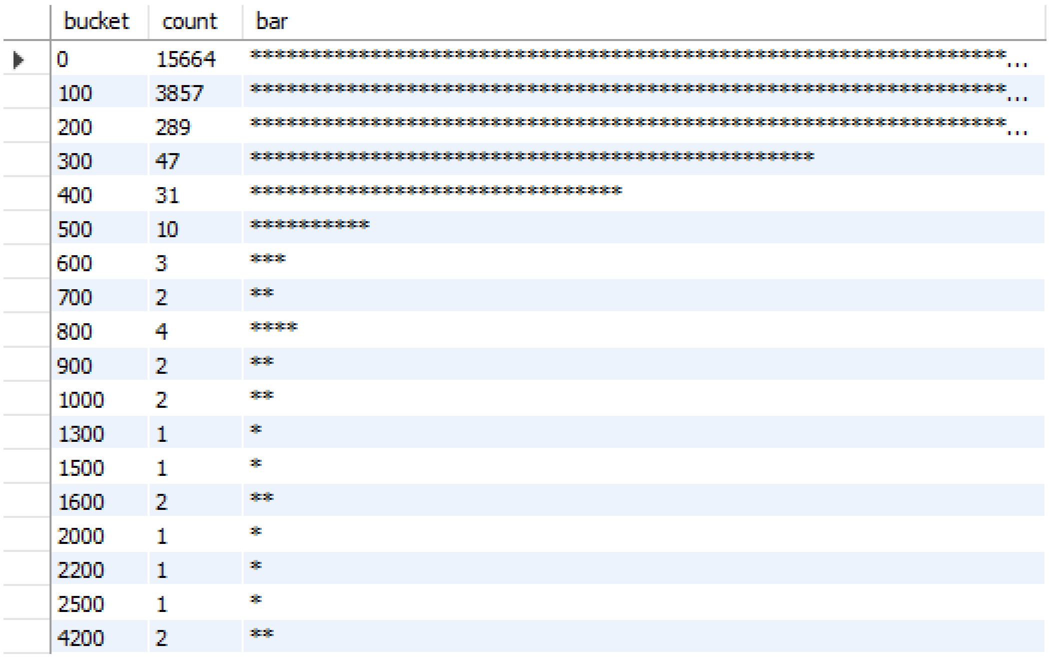
```
CREATE VIEW 1919_purchases AS
SELECT * FROM pricedata
WHERE buyer_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';

SELECT * FROM 1919_purchases;
```

- This query uses CREATE VIEW to establish a new view named 1919_purchases.
- Selects all columns from the price data table.
- Filters the data to include only those transactions where the specified address is the buyer.

Creating a view like 1919_purchases is an efficient way to repeatedly access a subset of data without the need to run complex queries each time. This can be particularly valuable for ongoing analysis or monitoring of specific entities in the NFT marketplace.

8. Create a histogram of ETH price ranges. Round to the nearest hundred value.

```
SELECT round(eth_price,-2) AS roundoff_eth_price,count(*) AS count,
RPAD('', COUNT(*), '*') AS bar FROM pricedata
GROUP BY roundoff_eth_price
ORDER BY roundoff_eth_price;
```



bucket	count	bar
0	15664	***...
100	3857	***...
200	289	***...
300	47	***
400	31	*******************************
500	10	**********
600	3	***
700	2	**
800	4	****
900	2	**
1000	2	**
1300	1	*
1500	1	*
1600	2	**
2000	1	*
2200	1	*
2500	1	*
4200	2	**

This histogram provides a quick, visual representation of the ETH price distribution in the NFT market, making it easier to understand which price ranges are most common.

9. **Return a unioned query that contains the highest price each NFT was bought for and a new column called status saying “highest” with a query that has the lowest price each NFT was bought for and the status column saying “lowest”. The table should have a name column, a price column called price, and a status column. Order the result set by the name of the NFT, and the status, in ascending order.**

```
SELECT name, usd_price AS price, 'Heighest' AS status FROM pricedata
WHERE usd_price = (SELECT MAX(usd_price) FROM pricedata)
UNION
SELECT name, usd_price AS price, 'Lowest' AS status FROM pricedata
WHERE usd_price = (SELECT MIN(usd_price) FROM pricedata)
ORDER BY name ASC, status ASC;
```

This combined query provides a comprehensive view of the price range for each NFT, allowing for a comparative analysis of their highest and lowest sale values. Such insights are particularly useful for buyers and sellers in understanding the price volatility and market trends of specific NFTs.

10. **What NFT sold the most each month / year combination? Also, what was the name and the price in USD? Order in chronological format.**

```
SELECT name, sale_month, sale_year, sale_count, usd_price
FROM (SELECT name, MAX(usd_price) as usd_price,
YEAR(event_date) AS sale_year,
MONTH(event_date) AS sale_month,
COUNT(*) AS sale_count,
DENSE_RANK() OVER (
PARTITION BY YEAR(event_date), MONTH(event_date) ORDER BY COUNT(*) DESC)
as ranked_in_month
FROM pricedata
GROUP BY name, YEAR(event_date), MONTH(event_date)
) as dt
WHERE ranked_in_month = 1;
```

- First, groups the sales data by NFT name and each month-year combination.
- Calculates the maximum USD price, total sales count (COUNT(*)), and assigns a rank based on the sales count for each month-year (DENSE_RANK()).
- Filters to include only those NFTs that are ranked first in their respective month-year, indicating they are the top-selling for that period.

This approach provides a clear view of which NFTs dominated the market in terms of sales volume at different times, offering valuable insights into market preferences and trends.

11. **Return the total volume (sum of all sales), round to the nearest hundred on a monthly basis (month/year).**

```
SELECT YEAR(event_date) AS sale_year, MONTH(event_date) AS sale_month,
ROUND(SUM(usd_price), -2) AS sum_of_sales_volume
FROM pricedata
GROUP BY YEAR(event_date), MONTH(event_date)
ORDER BY YEAR(event_date), MONTH(event_date);
```

Understanding the total volume of sales in the NFT market on a monthly basis is crucial for grasping market dynamics and trends. The above SQL query calculates the total sales volume, rounded to the nearest hundred, for each month and year, providing a clear picture of how the market has evolved over time.

12. **Count how many transactions the wallet "0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685" had over this time period.**

```
SELECT COUNT(*) AS no_of_transactions FROM pricedata
WHERE seller_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685'
OR buyer_address = '0x1919db36ca2fa2e15f9000fd9cdc2edcf863e685';
```

Number of transactions the above mentioned wallet had over the time period is 491.

13. **Create an "estimated average value calculator" that has a representative price of the collection every day based off of these criteria:**
    Exclude all daily outlier sales where the purchase price is below 10% of the daily average price.
    Take the daily average of remaining transactions.

a) **First create a query that will be used as a sub-query. Select the event date, the USD price, and the average USD price for each day using a window function. Save it as a temporary table.**

b) **Use the table you created in Part A to filter out rows where the USD prices is below 10% of the daily average and return a new estimated value which is just the daily average of the filtered data.**

```
create temporary table temp1 as (select event_date, avg(usd_price)
over (partition by event_date) as avg_usd_price from pricedata);

select pricedata.*,temp1.avg_usd_price from pricedata left join
temp1 on temp1.event_date=pricedata.event_date
where pricedata.usd_price>temp1.avg_usd_price*0.1;
```

This approach involves creating an "estimated average value calculator" that excludes outlier sales and focuses on the average transaction value each day. This two-step process results in a more accurate and representative average value of NFT collections by excluding significant outliers.

14. **Create two temporary table and name 'buyer' and 'seller'. Return buying price and selling price along with the profit where buyer address and seller address are same.**

```
create temporary table buyer as
select buyer_address, sum(usd_price) as b_price from pricedata
group by buyer_address
order by b_price;

create temporary table seller as
select seller_address, sum(usd_price) as s_price from pricedata
group by seller_address
order by s_price;

select buyer.buyer_address,
buyer.b_price,
seller.seller_address,
seller.s_price,
(seller.s_price-buyer.b_price) as profit
from buyer right join seller
on buyer.buyer_address=seller.seller_address;
```

This profitability analysis is crucial for understanding who the key players are in the NFT market and how successful their trading strategies have been. It can reveal patterns in buying and selling behavior and highlight the most effective traders.