DataLemur Easy Questions using Python

In [ ]:
```
# Connection Setup
import pandas as pd
import sqlalchemy as sal
import numpy as np

Engine = sal.create_engine('mssql://HP\SQLEXPRESS/DATALEMUR_DATABASE?driver=ODBC
Conn = Engine.connect()
```

In [ ]:
```
#2. Tesla is investigating production bottlenecks and they need your help to ext
#determine which parts have begun the assembly process but are not yet finished.
# Assumptions:
# parts_assembly table contains all parts currently in production, each at varyi
# An unfinished part is one that lacks a finish_date.

df_parts_assembly = pd.read_sql_query('select * from parts_assembly',Conn)
df_parts_assembly[df_parts_assembly['finish_date'].isnull()][['part','assembly_s
```

In [ ]:
```
#3. Assume you're given a table Twitter tweet data, write a query to obtain a hi
# Output the tweet count per user as the bucket and the number of Twitter users
# In other words, group the users by the number of tweets they posted in 2022 an
df_tweets = pd.read_sql_query('select * from tweets',Conn)
df1 = df_tweets[df_tweets['tweet_date'].dt.year == 2022].groupby(['user_id'])['t
df1.groupby('tweet_bucket')['user_id'].count().reset_index(name = 'users_num')
```

In [ ]:
```
#4. Assume you're given the table on user viewership categorised by device type
# Write a query that calculates the total viewership for laptops and mobile devi
#tablet and phone viewership. Output the total viewership for laptops as laptop_
#devices as mobile_views.
import numpy as np
df_viewership = pd.read_sql_query('select * from viewership',Conn)
df_viewership= df_viewerships[['user_id','device_type','view_time']]
df_viewership['category'] = np.where(df_viewership['device_type']== 'laptop', 'l
df_viewership.groupby('category')['user_id'].count().reset_index(name='total_vie
# laptop_viewership = np.where(df_viewership['device_type']== 'laptop', 1,0).sum
# mobile_viewership = np.where(df_viewership['device_type'] != 'laptop', 1,0).su
# print('laptop_viewership: {}'.format(laptop_viewership))
# print('mobile_viewership: {}'.format(mobile_viewership))
```

In [ ]:
```
#5. Given a table of candidates and their skills, you're tasked with finding the
#job. You want to find candidates who are proficient in Python, Tableau, and Pos
# Write a query to list the candidates who possess all of the required skills fo
#ascending order.

df_candidates = pd.read_sql_query('select * from candidates',Conn)
df_candidates = df_candidates[df_candidates['skill'].isin(['Python','Tableau','P
df_candidates[df_candidates['cnt']==3]['candidate_id'].sort_values()
```

In [ ]:
```
#6. Given a table of Facebook posts, for each user who posted at least twice in
#between each user’s first post of the year and last post of the year in the yea
#between each user's first and last post.

df_posts = pd.read_sql_query('select * from posts',Conn)
df_posts = df_posts[df_posts['post_date'].dt.year == 2021].groupby('user_id').fi
df_posts.groupby('user_id')['post_date'].apply(lambda x: (x.max() - x.min()).day
```

In [ ]:
```
#7. Write a query to identify the top 2 Power Users who sent the highest number
#Display the IDs of these 2 users along with the total number of messages they s
#based on the count of the messages.
df_messages = pd.read_sql_query('select * from messages',Conn)
df_messages = df_messages[(df_messages['sent_date'].dt.month == 8 )& (df_message
df_messages['rnk'] = df_messages['total_messages'].rank(method='dense',ascending
df_messages[df_messages['rnk'] <=2][['sender_id','total_messages']].sort_values(
```

In [ ]:
```
#8. Assume you're given a table containing job postings from various companies o
#retrieve the count of companies that have posted duplicate job listings.
#Definition:Duplicate job listings are defined as two job listings within the sa
#descriptions.

df_joblistings = pd.read_sql_query('select * from job_listings',Conn)
df_joblistings = df_joblistings.groupby(['company_id','title','description'])['j
df_joblistings[df_joblistings['cnt']>1]['company_id'].nunique()
```

In [ ]:
```
#9. Assume you're given the tables containing completed trade orders and user de
# Write a query to retrieve the top three cities that have the highest number of
#order. Output the city name and the corresponding number of completed trade ord
df_trades = pd.read_sql_query('select * from trades',Conn)
df_users = pd.read_sql_query('select * from users', Conn)
df_merged = df_trades.merge(df_users, on='user_id')
df_merged = df_merged[df_merged['status']=='Completed'].groupby('city')['order_i
df_merged['rank'] = df_merged['trade_completed'].rank(method = 'dense',ascending
df_merged[df_merged['rank']<=3][['city','trade_completed']].sort_values('trade_c
```

In [ ]:
```
#10. Given the reviews table, write a query to retrieve the average star rating
#The output should display the month as a numerical value, product ID, and avera
#Sort the output first by month and then by product ID.
df_reviews = pd.read_sql_query('select * from reviews',Conn)

df_reviews['month'] = df_reviews['submit_date'].dt.month
df_reviews = df_reviews.groupby(['month','product_id'])['stars'].mean().reset_in
df_reviews
```

In [ ]:
```
#11. Given a table containing information about bank deposits and withdrawals ma
#final account balance for each account, taking into account all the transaction
#there are no missing transactions.account.
import numpy as np
df_transactions = pd.read_sql_query('select * from transactions',Conn)

df_transactions['amount'] = np.where(df_transactions['transaction_type']=='Depos
df_transactions.groupby('account_id')['amount'].sum().reset_index(name='final_ba
```

In [ ]:
```
#12. Write a query to determine the total number of tax filings made using Turbo
#once a year using only one product.
import numpy as np
df_filed_taxes = pd.read_sql_query('select * from filed_taxes',Conn)
df_filed_taxes['product_type'] = np.where(df_filed_taxes['product'].str.contains
df_filed_taxes.groupby('product_type')['filing_id'].count().reset_index(name = '
```

In [ ]:
```
#13. Assume you have an events table on Facebook app analytics. Write a query to
#the app in 2022 and round the results to 2 decimal places.
#Percentage of click-through rate (CTR) = 100.0 * Number of clicks / Number of i
```

```
#To avoid integer division, multiply the CTR by 100.0, not 100.

df_events = pd.read_sql_query('select * from events',Conn)
df_events = df_events[df_events['timestamp'].dt.year==2022]
pivot_df = df_events.pivot_table(index='app_id', columns='event_type', aggfunc='
pivot_df['CTR'] = (pivot_df['click'] / pivot_df['impression']) * 100.0
pivot_df['CTR'] = pivot_df['CTR'].round(2)
pivot_df
```

In [ ]:
```
#14. Assume you're given tables with information about TikTok user sign-ups and
#on TikTok sign up using their email addresses, and upon sign-up, each user rece
#their account. Write a query to display the user IDs of those who did not confi
#on the second day.
#Definition:
#action_date refers to the date when users activated their accounts and confirme

df_emails = pd.read_sql_query('select * from emails',Conn)
df_texts = pd.read_sql_query('select * from texts',Conn)
df_merged = df_emails.merge(df_texts,on='email_id')
df_merged = df_merged[((df_merged['action_date'] - df_merged['signup_date']).dt.
df_merged
```

In [ ]:
```
#15.Your team at JPMorgan Chase is preparing to launch a new credit card, and to
#credit cards were issued each month.Write a query that outputs the name of each
#of issued cards between the month with the highest issuance cards and the lowes
#Arrange the results based on the largest disparity.

df_monthly_cards_issued = pd.read_sql_query('select * from monthly_cards_issued'
df = df_monthly_cards_issued.groupby('card_name')['issued_amount'].agg(['max','m
df['amount_difference'] = df['max'] - df['min']
df[['card_name','amount_difference']].sort_values('amount_difference',ascending=
```

In [ ]:
```
#16. You're trying to find the mean number of items per order on Alibaba, rounde
#information on the count of items in each order (item_count table) and the corr
#(order_occurrences table).
df_itemsperorder = pd.read_sql_query('select * from items_per_order',Conn)
df_itemsperorder['total_orders'] = df_itemsperorder['order_occurrences']*df_item
(df_itemsperorder['total_orders'].sum() / df_itemsperorder['order_occurrences'].
```

In [ ]:
```
#17.CVS Health is trying to better understand its pharmacy sales, and how well d
#Each drug can only be produced by one manufacturer. Write a query to find the t
#how much profit they made. Assume that there are no ties in the profits. Displa
#lowest total profit
df_pharmacysales = pd.read_sql_query('select * from pharmacy_sales',Conn)

df_pharmacysales = df_pharmacysales.groupby('drug')[['total_sales','cogs']].sum(
df_pharmacysales['total_profit'] = df_pharmacysales['total_sales'] - df_pharmacy
df_pharmacysales['rank'] = df_pharmacysales['total_profit'].rank(method='dense',
df_pharmacysales[df_pharmacysales['rank']<=3][['drug','total_profit']].sort_valu
```

In [ ]:
```
#18. CVS Health is analyzing its pharmacy sales data, and how well different pro
#is exclusively manufactured by a single manufacturer.Write a query to identify
#that resulted in losses for CVS Health and calculate the total amount of losses
#the number of drugs associated with losses, and the total losses in absolute va
#descending order with the highest losses displayed at the top.

df_pharmacysales = pd.read_sql_query('select * from pharmacy_sales',Conn)
df_pharmacysales['total_loss'] = df_pharmacysales['cogs'] - df_pharmacysales['to
```

```
df_pharmacysales = df_pharmacysales[df_pharmacysales['total_loss'] >0].groupby('
df_pharmacysales
```

In [ ]:
```
#19. Write a query to calculate the total drug sales for each manufacturer. Roun
#your results in descending order of total sales. In case of any duplicates, sor
#Since this data will be displayed on a dashboard viewed by business stakeholder
#"$36 million".

df_pharmacysales = pd.read_sql_query('select * from pharmacy_sales',Conn)
df_pharmacysales = df_pharmacysales.groupby('manufacturer')['total_sales'].sum()
df_pharmacysales['drug_sales'] = "$" + (df_pharmacysales['total_drug_sales']/100
df_pharmacysales.sort_values('total_drug_sales', ascending=False)[['manufacturer
```

In [ ]:
```
#20. Amazon is trying to identify their high-end customers. To do so, they first
#the most expensive purchase made by each customer. Order the results by the mos

df_transactions = pd.read_sql_query('select * from mep_transactions',Conn)
df_transactions.groupby('customer_id')['purchase_amount'].max().reset_index(name
```

In [ ]:
```
#21. Visa is analysing its partnership with ApplyPay. Calculate the total transa
#transaction was performed via ApplePay.Output the merchant ID and the total tra
#transactions, output their total transaction volume as 0. Display the result in
import numpy as np
df_transactions = pd.read_sql_query('select * from apv_transactions',Conn)
df_transactions['transaction_amount'] = np.where(df_transactions['payment_method
    'transaction_amount'],0)
df_transactions.groupby('merchant_id')['transaction_amount'].sum().reset_index(n
```

In [ ]:
```
#22. You are tasked with identifying Subject Matter Experts (SMEs) at Accenture
#They have 8 or more years of work experience in a single domain.
#They have 12 or more years of work experience across two different domains.
#Write a query to return the employee IDs of all the subject matter experts at A
df_employeeexpertise = pd.read_sql_query('select * from employee_expertise',Conn
result = df_employeeexpertise.groupby('employee_id').filter(lambda x: (
    (x['domain'].nunique() == 2 and x['years_of_experience'].sum() >= 12) or
    (x['domain'].nunique() == 1 and x['years_of_experience'].sum() >= 8)
   )
)
result['employee_id'].drop_duplicates()
```

In [ ]:
```
#23. The LinkedIn Creator team is seeking out individuals who have a strong infl
#profiles as a company or influencer page. To identify such power creators, we c
#LinkedIn page with the number of followers on the company they work for. If a p
#than their company, we consider them to be a power creator.Write a query to ret
df_personalprofile = pd.read_sql_query('select * from personal_profiles',Conn)
df_companypages = pd.read_sql_query('select * from company_pages',Conn)
df_merged = df_personalprofile.merge(df_companypages, left_on = 'employer_id',ri
df_merged[df_merged['followers_x']> df_merged['followers_y']]['profile_id'].sort
```

In [ ]:
```
#24. Assume that you are given the table below containing information on various
#to obtain the user IDs and number of products purchased by the top 3 customers;
#$1,000 in total. Output the user id and number of products in descending order.
#bought 10 products), the user who spent more should take precedence.

df_usertransactions = pd.read_sql_query('select * from user_transactions', Conn)
df_usertransactions = df_usertransactions.groupby('user_id').agg({'product_id':'
df_usertransactions = df_usertransactions[df_usertransactions['spend']>=1000].sc
df_usertransactions['rank'] = df_usertransactions['product_id'].rank(method='fir
df_usertransactions = df_usertransactions[df_usertransactions['rank'] <=3][['use
df_usertransactions
```

In [ ]:
```
#25. Microsoft Azure's capacity planning team wants to understand how much data
#capacity is left in each of its data centers. You’re given three tables: custom
#a query to find each data centre’s total unused server capacity. Output the dat
#spare capacity.

df_datacenter = pd.read_sql_query('select * from datacenters',Conn)
df_forecasteddemand = pd.read_sql_query('select * from forecasted_demand',Conn)
df_merged = df_datacenter.merge(df_forecasteddemand,on='datacenter_id')
df_merged = df_merged.groupby(['datacenter_id','monthly_capacity'])['monthly_dem
df_merged['spare_data'] = df_merged['monthly_capacity'] - df_merged['monthly_dem
df_merged[['datacenter_id','spare_data']].sort_values(['datacenter_id','spare_da
```

In [ ]:
```
#26.Assume you are given the table below containing information on user reviews.
#of businesses that are top rated. A top-rated busines is defined as one whose r
#number of businesses and percentage of top rated businesses rounded to the near

df_reviews = pd.read_sql_query('select * from trb_reviews',Conn)
total_business = df_reviews['business_id'].count()
top_rated_business = df_reviews[df_reviews['review_stars'].isin([4,5])]['busines
percentage_of_top_rated = (100.0*top_rated_business/total_business).round()
result_df = pd.DataFrame({'top_rated_business':[top_rated_business],'percentage_
result_df
```

In [ ]:
```
#27. Google marketing managers are analyzing the performance of various advertis
#help to gather the relevant data. Write a query to calculate the return on ad s
#campaigns. Round your answer to 2 decimal places, and order your output by the
#Hint: ROAS = Ad Revenue / Ad Spend

df_adcampaigns = pd.read_sql_query('select * from ad_campaigns',Conn)
df_adcampaigns = df_adcampaigns.groupby('advertiser_id').agg(total_revenue=('rev
df_adcampaigns['ROAS'] = (df_adcampaigns['total_revenue'] / df_adcampaigns['tota
df_adcampaigns[['advertiser_id','ROAS']].sort_values('advertiser_id')
```

In [ ]:
```
#28. Trde In Payouts
df_trade_transaction = pd.read_sql_query('select * from tip_trade_in_transaction
df_trade_payouts = pd.read_sql_query('select * from tip_trade_in_payouts',Conn)
df_merged = df_trade_transaction.merge(df_trade_payouts,on='model_id')
df_merged.groupby('store_id')['payout_amount'].sum().reset_index(name = 'total_p
```

In [ ]:
```
#29. Webinar Popularity
#As a Data Analyst on Snowflake's Marketing Analytics team, you're analyzing the
df_marketing = pd.read_sql_query('select * from marketing_touches',Conn)
df_marketing = df_marketing[(df_marketing['event_date'].dt.year == 2022) & (df_m
perct = 100*df_marketing[df_marketing['event_type'].str.lower() == 'webinar']['e
perct
```

In [ ]:
```
#30. Who made Quota?
#As a data analyst on the Oracle Sales Operations team, you are given a list of
#they need to hit. Write a query that outputs each employee id and whether they
import numpy as np
df_deals = pd.read_sql_query('select * from wmq_deals',Conn)
df_sales_quota = pd.read_sql_query('select * from wmq_sales_quotas',Conn)
df_merged = df_deals.merge(df_sales_quota,on='employee_id')
df_merged= df_merged.groupby(['employee_id','quota'])['deal_size'].sum().reset_i
```

```
df_merged['hit_or_miss'] = np.where(df_merged['deal_size']>=df_merged['quota'],'
df_merged[['employee_id','hit_or_miss']].sort_values('employee_id',ascending=Tru
```

DataLemur Medium Questions using Python

In [ ]:
```
#31. Uber's Third Transaction
df_transactions = pd.read_sql_query('select * from m_transactions',Conn)
df_transactions['rank'] = df_transactions.groupby('user_id')['transaction_date']
df_transactions[df_transactions['rank']==3][['user_id','spend','transaction_date
```

In [ ]:
```
#32.Sending Vs Opening Snapchat
import numpy as np
df_activities = pd.read_sql_query('select * from m_so_activities',Conn)
df_agebreakdown = pd.read_sql_query('select * from m_so_age_breakdown',Conn)
df_merged = df_activities.merge(df_agebreakdown,on='user_id')
df_merged_pivot = df_merged[df_merged['activity_type']!='chat'].pivot_table(valu
df_merged_pivot['send_prct'] = (100*df_merged_pivot['send']/(df_merged_pivot['op
df_merged_pivot['open_prct'] = (100*df_merged_pivot['open']/(df_merged_pivot['op
df_merged_pivot[['age_bucket','send_prct','open_prct']]
```

In [ ]:
```
#33.Tweets'Rolling Averages
df = pd.read_sql_query('select * from m_tra_tweets',Conn)
df['ld'] = df.groupby('user_id')['tweet_count'].rolling(window=3, min_periods=1)
df
```

In [ ]:
```
#34. Highest Grossing Items
#Assume you're given a table containing data on Amazon customers and their spend
#query to identify the top two highest-grossing products within each category in
#category, product, and total spend

df_product = pd.read_sql_query('select * from m_product_spend',Conn)
df_product = df_product[df_product['transaction_date'].dt.year==2022].groupby(['
df_product['rnk'] = df_product.groupby('category')['total_spend'].rank(method='d
df_product[df_product['rnk']<=2][['category','product','total_spend']]
```

In [ ]:
```
#35. Top 5 Artists
df_artist = pd.read_sql_query('select * from m_t_a_artists',Conn)
df_songs = pd.read_sql_query('select * from m_t_a_songs',Conn)
df_global_song_rank = pd.read_sql_query('select * from m_t_a_global_song_rank',C

df1 = df_artist.merge(df_songs,on='artist_id')
df_merged = df1.merge(df_global_song_rank,on='song_id')

df_merged = df_merged[df_merged['rank']<=10].groupby('artist_name')['song_id'].c
df_merged['rnk'] = df_merged['song_count'].rank(method='dense',ascending=False)
df_merged[df_merged['rnk']<=5][['artist_name','rnk']].sort_values('rnk',ascendin
```

In [ ]:
```
#36. Signup Activation Rate
df_emails = pd.read_sql_query('select * from m_sar_emails',Conn)
df_texts = pd.read_sql_query('select * from m_sar_texts',Conn)
df_merged = df_emails.merge(df_texts, how = 'left', on='email_id')
activation_rate = df_merged[df_merged['signup_action'] == 'Confirmed']['email_id
round(activation_rate,2)
```

In [ ]:
```
#38. Spotify Streaming History(Good Question)
df_history = pd.read_sql_query('select * from m_ssh_songs_history',Conn)
```

```
df_weekly = pd.read_sql_query('select * from m_ssh_songs_weekly',Conn)
df_weekly = df_weekly[df_weekly['listen_time'] <= '2022-08-04 23:59:59']
df_weekly = df_weekly.groupby(['user_id','song_id'])['listen_time'].count().rese
df_merged = df_weekly.merge(df_history,how='outer',on=['user_id','song_id'])
df_merged = df_merged.fillna(0)
df_merged['song_plays'] = df_merged['song_plays'] + df_merged['song_plays_weekly
df_merged[['user_id','song_id','song_plays']].sort_values('song_plays',ascending
```

In [ ]:
```
#40. Pharmacy Analytics(Part-4)
df_pharmacy = pd.read_sql_query('select * from pharmacy_sales',Conn)
df_pharmacy = df_pharmacy.groupby(['manufacturer','drug'])['units_sold'].sum().r
df_pharmacy['rank'] = df_pharmacy.groupby('manufacturer')['total_units_sold'].ra
df_pharmacy[df_pharmacy['rank']<=2][['manufacturer','drug']]
```

In [ ]:
```
#41. Frequently Purchased Pairs
df_transaction = pd.read_sql_query('select * from product_transactions',Conn)
result_df = df_transaction.groupby('transaction_date')['product_id'].agg(lambda
result_df = result_df[result_df['product_id'].str.count(',') > 0]['product_id'].
result_df
```

In [ ]:
```
#42. Supercloud  Customer
df_customercontract = pd.read_sql_query('select * from m_sc_customer_contracts',
df_products = pd.read_sql_query('select * from m_sc_products',Conn)

df_merged = df_customercontract.merge(df_products,on='product_id')
df_merged = df_merged.groupby('customer_id')['product_category'].nunique().reset
df_merged = df_merged[df_merged['total_products'] == (df_products['product_categ
df_merged
```

In [ ]:
```
#43. Odd and Even Measurements
df_measurements = pd.read_sql_query('select * from m_oem_measurements',Conn)
df_measurements['measurement_day'] = df_measurements['measurement_time'].dt.date
df_measurements['rank']= df_measurements.groupby('measurement_day')['measurement
df_measurements['odd_value'] = np.where(df_measurements['rank']%2!=0, df_measure
df_measurements['even_value'] = np.where(df_measurements['rank']%2==0, df_measur
df_measurements.groupby('measurement_day').agg(odd_sum = ('odd_value','sum'),eve
```

In [ ]:
```
#44. Booking Referral Source
df_bookings = pd.read_sql_query('select * from m_brs_bookings',Conn)
df_bookingattr = pd.read_sql_query('select * from m_brs_booking_attribution',Con
df_merged = df_bookingattr.merge(df_bookings,on='booking_id')
df_merged['rnk'] = df_merged.groupby('user_id')['booking_date'].rank(method='fir
df_merged['channel'].fillna('None',inplace=True)
df_merged = df_merged[df_merged['rnk'] == 1].groupby(['channel','rnk'])['booking
df_merged['prct'] = (100*df_merged['cnt']/df_merged['cnt'].sum()).round(2)
df_merged[(df_merged['cnt'] == df_merged['cnt'].max()) &(df_merged['channel']!='
```

In [ ]:
```
#45. Shopping Spree
df_transactions = pd.read_sql_query('select * from m_uss_transactions',Conn)
df_transactions = df_transactions.sort_values(by=['user_id', 'transaction_date']
df_transactions['diff1'] = (df_transactions.groupby('user_id')['transaction_date
df_transactions['diff2'] = (df_transactions.groupby('user_id')['transaction_date
df_transactions[(df_transactions['diff1']==1)&(df_transactions['diff2'] ==1)]['u
```

In [ ]:
```
#46.2nd Ride Delay
df_users = pd.read_sql_query('select * from m_rd_users',Conn)
df_rides = pd.read_sql_query('select * from m_rd_rides',Conn)
df_merged = df_users.merge(df_rides,on='user_id')
```

```
df_merged['rnk'] = df_merged.groupby('user_id')['ride_date'].rank(method='first'
user_id_in_the_moment = df_merged[(df_merged['rnk']==1) & (df_merged['registrati
df_merged = df_merged[df_merged['user_id'].isin(user_id_in_the_moment)]
df_merged = df_merged[df_merged['rnk'] == 2]
ride_delay = round(((df_merged['ride_date'].dt.day - df_merged['registration_dat
ride_delay
```

In [ ]:
```
#47. Histogram of Users and Purchases
df_transactions = pd.read_sql_query('select * from m_hup_user_transactions',Conn
df_transactions['rnk'] = df_transactions.groupby('user_id')['transaction_date'].
df_transactions = df_transactions[df_transactions['rnk'] == 1].groupby(['user_id
df_transactions.sort_values('transaction_date')
```

In [ ]:
```
#48. Google Maps Flagged UGC
df_placeinfo = pd.read_sql_query('select * from m_gmf_place_info',Conn)
df_mapsreview = pd.read_sql_query('select * from m_gmf_maps_ugc_review',Conn)
df_merged = df_placeinfo.merge(df_mapsreview,on='place_id')
df_merged = df_merged[df_merged['content_tag'].str.lower() == 'off-topic'].group
df_merged['rnk'] = df_merged['total_tags'].rank(method= 'dense',ascending=False)
df_merged[df_merged['rnk']==1]['place_category'].sort_values()
```

In [ ]:
```
#49. Compressed Mode
df_items = pd.read_sql_query('select * from  items_per_order',Conn)
df_items = df_items.groupby('item_count')['order_occurrences'].sum().reset_index
df_items['occurence_count'] = df_items['order_occurrences'].rank(method = 'dense
df_items[df_items['occurence_count'] == 1]['item_count']
```

In [ ]:
```
#50. Card Launch
df_cardsissued = pd.read_sql_query('select * from monthly_cards_issued',Conn)
df_cardsissued = df_cardsissued.groupby(['card_name','issue_year','issue_month']
df_cardsissued['issue_date'] = df_cardsissued['issue_year'].astype(str)+'-'+df_c
df_cardsissued['rnk'] = df_cardsissued.groupby('card_name')['issue_date'].rank(m
df_cardsissued[df_cardsissued['rnk']==1][['card_name','total_amount']].sort_valu
```

In [ ]:
```
#51. International Call Percentage
df_phonecalls = pd.read_sql_query('select * from phone_calls',Conn)
df_phoneinfo = pd.read_sql_query('select * from m_icp_phone_info',Conn)
df_merged_caller = df_phonecalls.merge(df_phoneinfo,on='caller_id')
df_merged = df_merged_caller.merge(df_phoneinfo, left_on='receiver_id_x', right_
call_prct = (100*df_merged[df_merged['country_id_x']!=df_merged['country_id_y']]
call_prct
```

In [ ]:
```
#52. LinkedIn Power Creators(Part 2)
df_personalprofile = pd.read_sql_query('select * from m_lpc_personal_profiles',C
df_employee = pd.read_sql_query('select * from m_lpc_employee_company',Conn)
df_companypages = pd.read_sql_query('select * from m_lpc_company_pages',Conn)
df_merged = df_personalprofile.merge(df_employee,left_on = 'profile_id',right_on
df_merged = df_merged.merge(df_companypages, on ='company_id')
df_merged['rnk'] = df_merged.groupby('profile_id')['followers_y'].rank(method='d
df_merged[(df_merged['rnk']==1)&(df_merged['followers_x']>df_merged['followers_y
```

In [ ]:
```
#53. Unique Money Transfer
df_payments = pd.read_sql_query('select * from m_umtp_payments',Conn)
df_merged = df_payments.merge(df_payments, left_on = ['payer_id','recipient_id']
df_merged = df_merged[['payer_id_x','recipient_id_x']].drop_duplicates()
df_merged['payer_id_x'].count() / 2
```

In [ ]:
```
#54. User Session Activity
df_session = pd.read_sql_query('select * from m_usa_sessions',Conn)
df_session = df_session[(df_session['start_date'] >='2022-01-01')&(df_session['s
df_session['rnk'] = df_session.groupby('session_type')['total_duration'].rank(me
df_session[['user_id','session_type','rnk']].sort_values(['session_type','rnk'])
```

In [ ]:
```
#55. First Transaction
df_transactions = pd.read_sql_query('select * from m_ft_user_transactions',Conn)
df_transactions = df_transactions.groupby(['user_id','transaction_date'])['spend
df_transactions['rnk'] = df_transactions.groupby('user_id')['transaction_date'].
df_transactions[(df_transactions['rnk']==1) & (df_transactions['total_spend']>=5
```

In [ ]:
```
#56. Email Table Transaction
#Each Facebook user can designate a personal email address, a business email add
#the table is currently in the wrong format, so you need to transform its struct
#output): user id, personal email, business email, and recovery email. Sort your
df_users = pd.read_sql_query('select * from m_ett_users',Conn)
pivoted_df = df_users.pivot(index='user_id', columns='email_type', values='email
pivoted_df.reset_index()
```

In [ ]:
```
#57. Photoshop Revenue Analysis
#For every customer that bought Photoshop, return a list of the customers, and t
#Photoshop products.Sort your answer by customer ids in ascending order.
df_transactions = pd.read_sql_query('select * from m_pra_adobe_transactions',Con
customer_ids = df_transactions[df_transactions['product'].str.lower()=='photoshc
df_transactions[(df_transactions['customer_id'].isin(customer_ids))&(df_transact
```

In [ ]:
```
#58. Cumulative Purchase by Product Type
df_transactions = pd.read_sql_query('select * from m_cppt_total_trans',Conn)
df_transactions = df_transactions.sort_values('order_date')
df_transactions['total_sum'] = df_transactions.groupby('product_type')['quantity
df_transactions[['order_date','product_type','total_sum']]
```

In [ ]:
```
#59.Invalid Search Results
df_category = pd.read_sql_query('select * from m_isr_search_category',Conn)
df_category['invalid_search'] = df_category['num_search']*df_category['invalid_r
df_category = df_category[df_category['invalid_result_pct'].notnull()].groupby('
df_category['overall_invalid_prct'] = 100.0*df_category['invalid_search']/df_cat
df_category[['country','num_search','overall_invalid_prct']].round(2).sort_value
```

In [ ]:
```
#60. Repeat Purchases on Multiple Days
df_purchases = pd.read_sql_query('select * from m_rpmd_purchases',Conn)
df_purchases['purchase_date'] = df_purchases['purchase_date'].dt.date
df_purchases = df_purchases.groupby(['user_id','product_id'])['purchase_date'].n
df_purchases = df_purchases[df_purchases['total_orders']>1]
df_purchases['user_id'].nunique()
```

In [ ]:
```
#61. Compensation Outliers
df_employee = pd.read_sql_query('select * from m_co_employee_pay',Conn)
df_employee['average'] = df_employee.groupby(['title'])['salary'].transform('mea
df_employee = df_employee[(df_employee['salary'] > df_employee['average']*2) | (
df_employee['decision'] = np.where(df_employee['salary'] > 2*df_employee['averag
df_employee
```

In [ ]:
```
#62. Y-on-Y Growth Rate
df_transactions = pd.read_sql_query('select * from h_ygr_user_transactions',Conn
df_transactions['transaction_date'] = df_transactions['transaction_date'].dt.yea
```

```
df_transactions = df_transactions.groupby(['product_id','transaction_date'])['sp
df_transactions.sort_values(['product_id','transaction_date'],inplace=True)
df_transactions['previous_year_spend'] = df_transactions.groupby('product_id')['
df_transactions['yoy'] = (100*(df_transactions['current_year_spend'] - df_transa
df_transactions
```

```
In [ ]: #63. Consecutive Filing Years
df_taxes = pd.read_sql_query('select * from filed_taxes',Conn)
df_taxes = df_taxes[df_taxes['product'].str.contains('turbotax', case=False)]
df_taxes['filing_date'] = df_taxes['filing_date'].dt.year
df_taxes.sort_values(['user_id','filing_date'],inplace=True)
df_taxes['ld1'] = df_taxes.groupby('user_id')['product'].shift(-1)
df_taxes['ld2'] = df_taxes.groupby('user_id')['product'].shift(-2)
df_taxes[(df_taxes['ld1'].isna()==False)&(df_taxes['ld2'].isna()==False)]['user_
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```

```
In [ ]:
```