ASSIGNMENT 1

Part 1: Costco

1. **What is the product?**
Costco sells a wide variety of items including food, electronics, apparel, appliances, and furniture.

2. **How are they selling it?**
- Costco sells its products through a membership-based warehouse club model.
- They offer discounts by tracking customer purchase history and communicate those via email and direct mail campaigns.
- The members have access to their physical stores offering products in bulk for cost savings.
- They also leverage their online presence via SEO and by engaging their customers/members via sponsorship.

3. **How is it priced?**
- Costco focuses on delivering value to its members so it focuses on value-oriented pricing, and bulk buying power by providing high-quality member benefits.
- Costco emphasizes on minimal markups allowing members to recognize cost savings.
- Their private label brands exemplified Kirkland Signature also helps them in providing savings on their products.

4. **What promotions are they using?**
- They employ a wide range of marketing promotions to attract customers.
- They distribute coupon booklets featuring discounts on a range of products, holding seasonal sales campaigns like holiday promotions with limited-time offers.
- Separate promotions for online and in-store shoppers. They provide cashback rewards encouraging higher tier membership.
- Costco primary promotion aligns with its commitment to offering values and loyalty.

5. **what marketing promotions costco use to attract its customers?**

- Match Customer Offerings - Promotions and Advertisements:
    - Costco employs machine learning algorithms with specific customer segments. They target customers with relevant promotions ensuring that the members receive offers aligning with their preferences and buying patterns. They use this by analyzing member data and purchase history.

- Find products for a customer – Search and Recommendation

    - Costco uses recommendation algorithms to suggest products to its customers. They consider factors like browsing behavior and preferences of similar customers. This personalized recommendations to help discover new products and enables additional purchases.

- Determine offering properties – pricing and assortment.
    - Pricing is an integral part of Costco's strategy. They utilize dynamic pricing algorithms that will adjust based on demand, inventory and competitive pricing which ensures its members get competitive prices in real time.
    - Inventory mgt helps determine the assortment of products in stores. This helps Costco optimize product selection based on historical sales data and customer preferences ensuring stores are well-stocked with items that members are most likely to purchase.

6. **What datasets do you think you will need to build these algorithmic services?**
    a. How frequently will data change?
    b. How would you store these datasets?

- Customer Data: Costco's customers data, their demographics, purchase history, browsing behavior, location data, and contact details. Customer data may change infrequently for core demographics but more frequently for purchase and browsing behavior.This data would be stored in a database/data lake for analysis.

- Product Catalog: Costco's products, such as product id, prices, inventory quantity, and historical sales data.This data would change regularly due to price adjustments, new product lines, or discontinuation of some products.This product database will be integrated with the inventory management system so that stock requirements are updated regularly.

- Transaction Data: sales transactions, products purchased, date, and price. This data would change with each customer purchase and is dynamic.

- Website and App Analytics: user interactions on Costco's website and mobile apps ie. page views, click-through rates, and conversion metrics, revenue. Dynamic data that changes every minute and has to be updated and uploaded on the database instantly and is stored in a data lake for batch processing.

- Inventory Data: product availability, stock levels, and supply chain information. Changes as products are sold, restocked, or relocated within stores.

- Pricing and Promotions Data: pricing strategies, discounts, promotions, and competitors' pricing.Change frequently during holidays/sales/market conditions. Database integrated with price optimization algorithms.

- Customer Feedback and Reviews:Sentiment data from customer reviews and feedback on Costco's products and services.Generated continuously as reviews and can be accessed to use for sentiment analysis and customer sentiment tracking.A combination of databases/data warehouses/data lakes to store high volume and velocity data. Real-time data stored in databases for low-latency access, while historical data in warehouses/lakes for analysis and reporting.

7. Review the jobs/career site and search for Data/ Data science positions (https://www.stitchfix.com/careers/jobs#below-the-fold). After review of the site, what technologies and programmatic services is the company using?

Stitch Fix uses programmatic services like Python, Anaplan, Essbase as mentioned in their ML, DS roles on their career sites. Other technologies mentioned include Swift, iOS, GraphQL, SwiftUI, UIKit, Combine, Swift Concurrency, Unit & UI Testing.

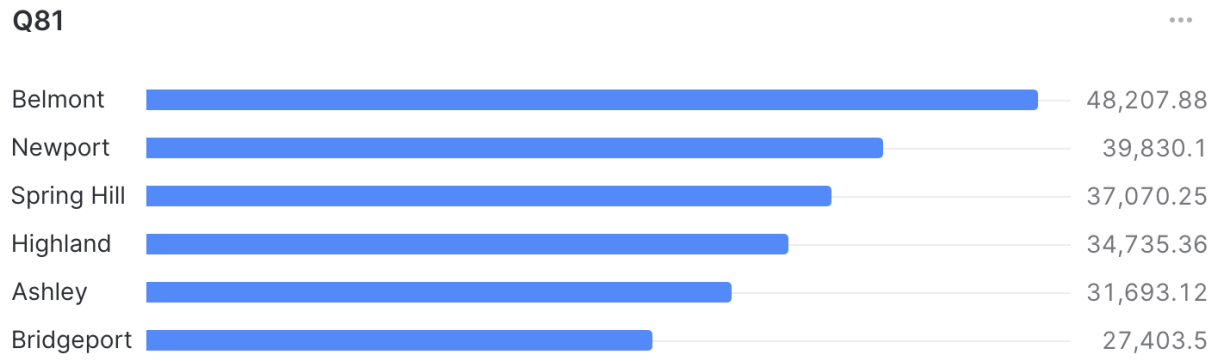
Other plausible data technologies that they make use of are as follows:

- Snowflake for its data warehousing needs due to its scalability. For relational databases, PostgreSQL, MySQL is utilized for various operational and transactional purposes. NoSQL databases like MongoDB or Cassandra.
- To facilitate ETL processes, Stitch Fix could use data integration tools such as Apache Nifi, Talend, or custom-built ETL pipelines.
- For data lake technologies, Amazon S3 / Azure Data Lake Storage can be used to store raw and semi-structured data before processing and analysis. Tools like Tableau, and Looker, for data visualization, reporting and business intelligence coupled with ML frameworks like TensorFlow, PyTorch are used.

Part 2: SQL Alchemy Dashboard

Query 81:

Find customers and their detailed customer data who have returned items bought from the catalog more than 20 percent the average customer returns for customers in a given state in a given time period. Order output by customer data.




```
-- TPC-DS_query81
with customer_total_return as
 (select cr_returning_customer_sk as ctr_customer_sk
        ,ca_state as ctr_state,
    sum(cr_return_amt_inc_tax) as ctr_total_return
 from catalog_returns
     ,date_dim
     ,customer_address
 where cr_returned_date_sk = d_date_sk
   and d_year =2000
   and cr_returning_addr_sk = ca_address_sk
 group by cr_returning_customer_sk
         ,ca_state )
  select  c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name
                   ,ca_street_type,ca_suite_number,ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset
                  ,ca_location_type,ctr_total_return
 from customer_total_return ctr1
     ,customer_address
     ,customer
 where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2
               from customer_total_return ctr2
                   where ctr1.ctr_state = ctr2.ctr_state)
       and ca_address_sk = c_current_addr_sk
       and ca_state = 'VA'
       and ctr1.ctr_customer_sk = c_customer_sk
 order by c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name
                   ,ca_street_type,ca_suite_number,ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset
                  ,ca_location_type,ctr_total_return
 limit 100;
```

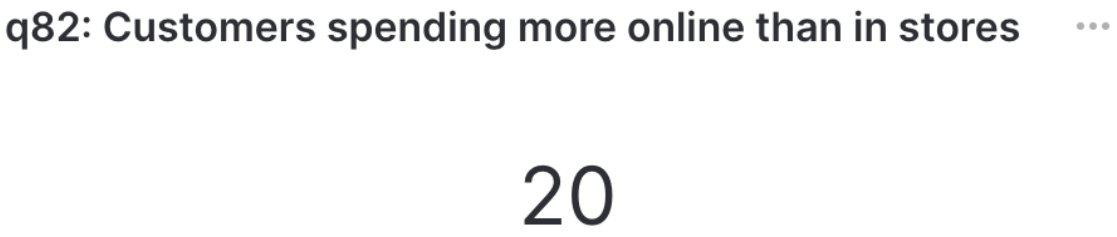
In the above SQL query,

1. A common table expression (CTE) named "customer_total_return" is created. This CTE calculates total return amount for customers who returned catalog items. It groups the data by returning customer, state, and calculates the sum of return amounts. The data is filtered based on the returned date being in the year 2000.
2. The main part of the query selects various customer-related columns along with the total return amount. It uses the CTE "customer_total_return" (ctr1) to filter customers who have returned more than 20% of the average return amount for customers in the same state.

3. The subquery within the main query calculates the average return amount multiplied by 1.2 (20% more) for customers in the same state.
4. The data is filtered further to include only customers located in 'VA' (Virginia).
5. The query joins the customer address table (customer_address) and customer table using appropriate keys (current address and customer ID).
6. Results are ordered by various customer details, including customer ID, name, address, and total return amount.
7. The output is limited to the top 100 rows.
   In summary, this query identifies customers in Virginia who have returned catalog items significantly more than the state's average return rate (20% higher) in the year 2000. It retrieves detailed customer data and organizes the results by customer attributes.

Query 82:

Find customers who tend to spend more money (net-paid) on-line than in stores.




```
-- TPC-DS_query82
select  i_item_id
       ,i_item_desc
       ,i_current_price
 from item, inventory, date_dim, store_sales
 where i_current_price between 72 and 72+30
 and inv_item_sk = i_item_sk
 and d_date_sk=inv_date_sk
 and d_date between cast('1998-01-23' as date) and dateadd(day,60,to_date('1998-01-23'))
 and i_manufact_id in (412,343,781,156)
 and inv_quantity_on_hand between 100 and 500
 and ss_item_sk = i_item_sk
 group by i_item_id,i_item_desc,i_current_price
 order by i_item_id
 limit 100;
```

Here's an explanation of the above SQL code:

1. The query selects three columns: item ID (i_item_id), item description (i_item_desc), and current price (i_current_price).
2. Data is sourced from four tables: item, inventory, date_dim, and store_sales. These tables are joined using appropriate key relationships.
3. The query sets the following conditions for item selection:
    - The current price should fall within the range of $72 to $102 (72 + 30).
    - The inventory item key (inv_item_sk) should match the item key (i_item_sk).
    - The date key (d_date_sk) in inventory should match the date key (inv_date_sk).
    - Date should fall within a specified range (60 days starting from '1998-01-23').
    - Manufacturer ID (i_manufact_id) should belong to one of the specified values (412, 343, 781, 156).
    - The quantity on hand (inv_quantity_on_hand) in inventory should be between 100 and 500.
    - The item key in store_sales (ss_item_sk) should match the item key in item.
4. The data is grouped by item ID, item description, and current price.
5. The results are ordered by item ID in ascending order.
6. The output is limited to the top 100 rows.
   This query retrieves items that meet specific criteria based on price, manufacturer, quantity on hand, and date range. It allows for the identification of items that fit these criteria within a certain time frame and helps in analyzing item attributes.

Query 83:

Retrieve the items with the highest number of returns where the number of returns was approximately equivalent across all store, catalog and web channels (within a tolerance of +/- 10%), within the week ending a given date.

```sql
-- TPC-DS_query83
with sr_items as
 (select i_item_id item_id,
        sum(sr_return_quantity) sr_item_qty
 from store_returns,
      item,
      date_dim
 where sr_item_sk = i_item_sk
 and   d_date    in
    (select d_date
    from date_dim
    where d_week_seq in
        (select d_week_seq
        from date_dim
      where d_date in ('1998-02-20','1998-09-28','1998-11-14')))
 and   sr_returned_date_sk   = d_date_sk
 group by i_item_id),
 cr_items as
 (select i_item_id item_id,
        sum(cr_return_quantity) cr_item_qty
 from catalog_returns,
      item,
      date_dim
 where cr_item_sk = i_item_sk
 and   d_date    in
    (select d_date
    from date_dim
    where d_week_seq in
        (select d_week_seq
        from date_dim
      where d_date in ('1998-02-20','1998-09-28','1998-11-14')))
 and   cr_returned_date_sk   = d_date_sk
```

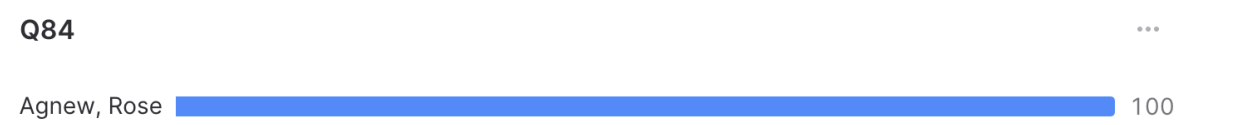
```
group by i_item_id),
wr_items as
(select i_item_id item_id,
       sum(wr_return_quantity) wr_item_qty
from web_returns,
     item,
     date_dim
where wr_item_sk = i_item_sk
and   d_date    in
   (select d_date
   from date_dim
   where d_week_seq in
       (select d_week_seq
       from date_dim
       where d_date in ('1998-02-20','1998-09-28','1998-11-14')))
and   wr_returned_date_sk   = d_date_sk
group by i_item_id)
 select  sr_items.item_id
      ,sr_item_qty
      ,sr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 sr_dev
      ,cr_item_qty
      ,cr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 cr_dev
      ,wr_item_qty
      ,wr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 wr_dev
      ,(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 average
from sr_items
    ,cr_items
    ,wr_items
where sr_items.item_id=cr_items.item_id
  and sr_items.item_id=wr_items.item_id
order by sr_items.item_id
        ,sr_item_qty
```

Here's breakdown of the code:

1. The first part of the query has 3 CTEs: sr_items, cr_items, and wr_items. Each CTE calculates the sum of return quantities for items from store, catalog, and web returns, respectively. It filters the data basis specific date range corresponding to the week ending on certain dates in 1998.
2. The main part of the query selects item_id, return quantities for each channel (sr_item_qty, cr_item_qty, wr_item_qty), and calculates the deviation percentage (sr_dev, cr_dev, wr_dev) for each channel's return quantity relative to the sum of return quantities for all three channels, normalized to a 100% scale. Additionally, it calculates the average return quantity across all channels.
3. The query joins all three CTEs on the item_id, so that they refer to the same items.
4. Output is ordered by item_id and return quantity (sr_item_qty) in ascending order.
5. Limit 100 will output top 100 rows.
   This query identified items with highest return quantities while ensuring that the returns are distributed relatively evenly across store, catalog, and web channels. It calculates the deviation percentage for each channel's return quantity and helps identify items with balanced return patterns within a specific week in 1998.

Query 84:

List all customers living in a specified city, with an income between 2 values




```sql
-- TPC-DS_query84
select  c_customer_id as customer_id
       , coalesce(c_last_name,'') || ', ' || coalesce(c_first_name,'') as customername
 from customer
     ,customer_address
     ,customer_demographics
     ,household_demographics
     ,income_band
     ,store_returns
 where ca_city          =  'Shady Grove'
   and c_current_addr_sk = ca_address_sk
   and ib_lower_bound   >=  29438
   and ib_upper_bound   <=  29438 + 50000
   and ib_income_band_sk = hd_income_band_sk
   and cd_demo_sk = c_current_cdemo_sk
   and hd_demo_sk = c_current_hdemo_sk
   and sr_cdemo_sk = cd_demo_sk
 order by c_customer_id
 limit 100;
```

Here's an explanation of the code:

1. The query selects the customer ID (c_customer_id) and concatenates the customer's last name and first name (customername) with a comma separator. This forms the result set containing customer information.
2. The data is sourced from several tables: customer, customer_address, customer_demographics, household_demographics, income_band, and store_returns.
3. It filters the data based on specific criteria:
    - Customers must be located in the city 'Shady Grove' (ca_city).

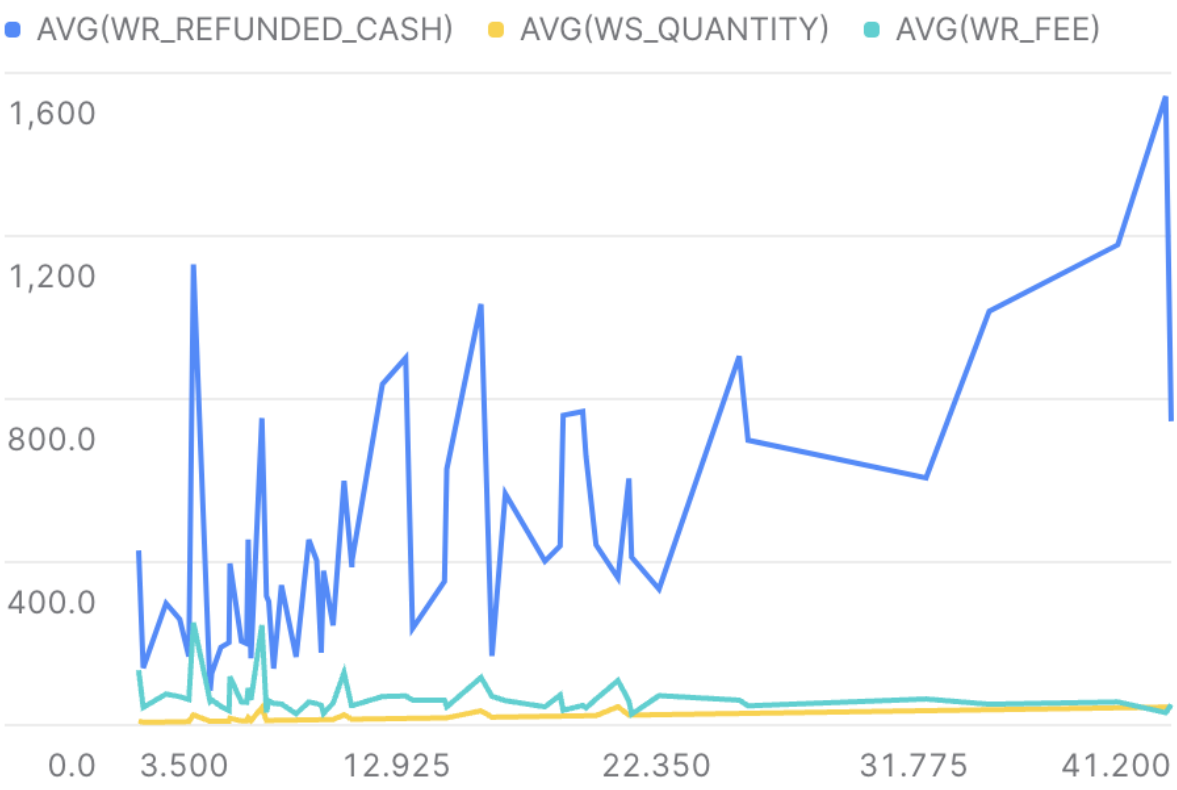
- The current address of the customer (c_current_addr_sk) must match the address in the customer_address table (ca_address_sk).
- The income band lower bound (ib_lower_bound) should be greater than or equal to 29438, and the upper bound (ib_upper_bound) should be less than or equal to 79438 (29438 + 50000).
- The income band (ib_income_band_sk) should match the income band associated with household demographics (hd_income_band_sk).
- The customer demographics (cd_demo_sk) should match the current customer demographics (c_current_cdemo_sk).
- The household demographics (hd_demo_sk) should match the current household demographics (c_current_hdemo_sk).
- Finally, there should be a match between store_returns' customer demographics (sr_cdemo_sk) and customer demographics (cd_demo_sk).

4. The results are ordered by customer ID (c_customer_id).
5. To limit the output, the query restricts the result set to the top 100 rows.
   In summary, this query retrieves a list of customers who live in Shady Grove, meet specific income criteria, and organizes the results by customer ID. It allows for the identification of customers within a particular city with a specified income range.

Query 85:

For all web return reason calculate the average sales, average refunded cash and average return fee by different combinations of customer and sales types (e.g., based on marital status, education status, state and sales profit).

Q85


AVG(WR_REFUNDED_CASH)
AVG(WS_QUANTITY)
AVG(WR_FEE)
1,600
1,200
800.0
400.0
0.0
3.500
12.925
22.350
31.775
41.200

```sql
-- TPC-DS_query85
select  substr(r_reason_desc,1,20)
       ,avg(ws_quantity)
       ,avg(wr_refunded_cash)
       ,avg(wr_fee)
 from web_sales, web_returns, web_page, customer_demographics cd1,
      customer_demographics cd2, customer_address, date_dim, reason
 where ws_web_page_sk = wp_web_page_sk
   and ws_item_sk = wr_item_sk
   and ws_order_number = wr_order_number
   and ws_sold_date_sk = d_date_sk and d_year = 1998
   and cd1.cd_demo_sk = wr_refunded_cdemo_sk
   and cd2.cd_demo_sk = wr_returning_cdemo_sk
   and ca_address_sk = wr_refunded_addr_sk
   and r_reason_sk = wr_reason_sk
   and
   (
    (
     cd1.cd_marital_status = 'D'
     and
     cd1.cd_marital_status = cd2.cd_marital_status
     and
     cd1.cd_education_status = 'Primary'
     and
     cd1.cd_education_status = cd2.cd_education_status
     and
     ws_sales_price between 100.00 and 150.00
    )
   or
    (
     cd1.cd_marital_status = 'U'
```

```
  and
  cd1.cd_marital_status = cd2.cd_marital_status
  and
  cd1.cd_education_status = '4 yr Degree'
  and
  cd1.cd_education_status = cd2.cd_education_status
  and
  ws_sales_price between 50.00 and 100.00
 )
or
 (
  cd1.cd_marital_status = 'W'
  and
  cd1.cd_marital_status = cd2.cd_marital_status
  and
  cd1.cd_education_status = 'Advanced Degree'
  and
  cd1.cd_education_status = cd2.cd_education_status
  and
  ws_sales_price between 150.00 and 200.00
 )
)
and
(
 (
  ca_country = 'United States'
  and
  ca_state in ('LA', 'CO', 'TX')
  and ws_net_profit between 100 and 200
 )
 or
 (
  ca_country = 'United States'
```

```
      and
      ca_state in ('OH', 'VA', 'MO')
      and ws_net_profit between 150 and 300
     )
     or
     (
      ca_country = 'United States'
      and
      ca_state in ('FL', 'OK', 'MS')
      and ws_net_profit between 50 and 250
     )
    )
group by r_reason_desc
order by substr(r_reason_desc,1,20)
        ,avg(ws_quantity)
        ,avg(wr_refunded_cash)
        ,avg(wr_fee)
limit 100;
```

Breakdown of the above code:

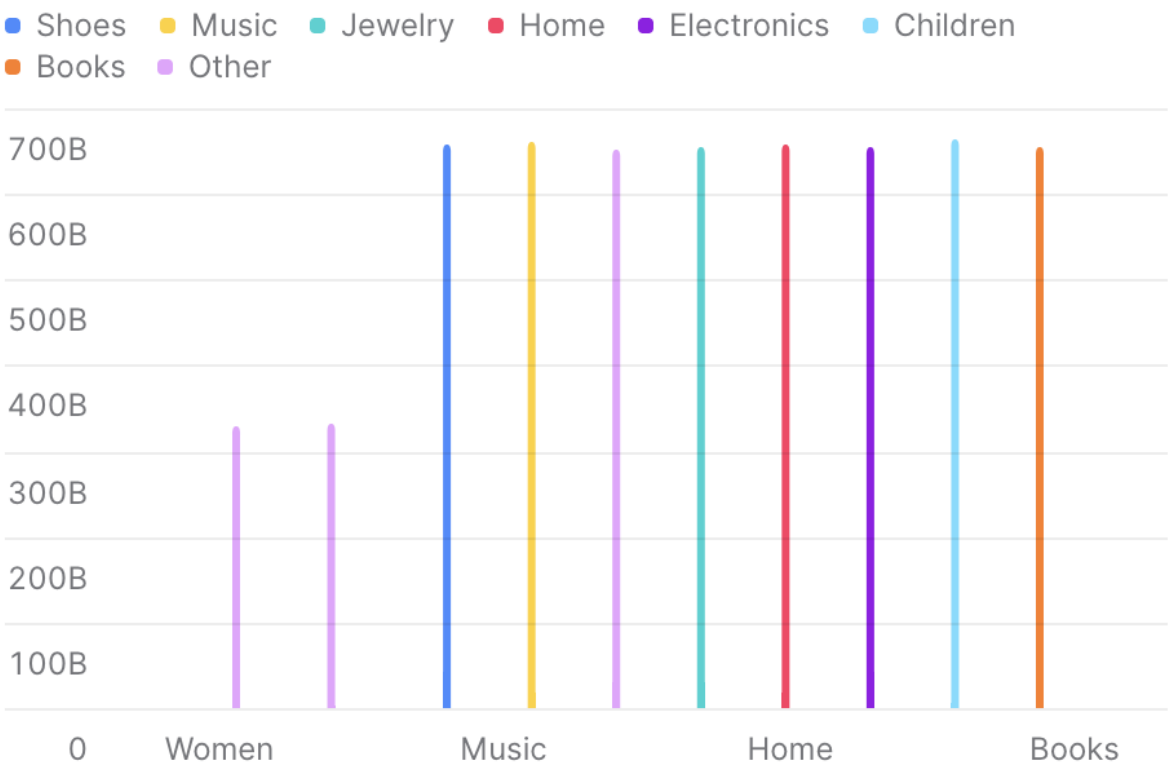
1. The query selects the first 20 characters of the return reason description (r_reason_desc), average web sales quantity (avg(ws_quantity)), average refunded cash (avg(wr_refunded_cash)), and average return fee (avg(wr_fee)).

2. It retrieves data from multiple tables: web_sales, web_returns, web_page, customer_demographics (twice, as cd1 and cd2), customer_address, date_dim, and reason.
3. Several join conditions are used to connect these tables based on various keys and relationships, ensuring that web sales and web returns data are linked to customer demographics, addresses, and reasons for return.
4. The query filters the data based on specific criteria:
    - The year 1998 is selected.
    - Customer demographics criteria are grouped into three categories (marital status, education status, and sales price range) using OR conditions.
    - Geographic criteria are also grouped into three categories (country, state, and net profit range) using OR conditions.
5. The data is grouped by return reason description (r_reason_desc).
6. The results are ordered by the first 20 characters of the return reason description and the average values of sales quantity, refunded cash, and return fee.
7. Finally, the output is limited to the top 100 rows.
   In summary, this query provides insights into average sales, refunded cash, and return fees for different web return reasons, considering a variety of customer demographics and sales-related factors such as marital status, education status, geographic location, and sales profit. It allows for an analysis of how these factors relate to return reasons.

Query 86

Rollup the web sales for a given year by category and class, and rank the sales among peers within the parent, for each group compute sum of sales, location with the hierarchy and rank within the group.

Q86


Shoes
Music
Jewelry
Home
Electronics
Children
Books
Other
700B
600B
500B
400B
300B
200B
100B
0
Women
Music
Home
Books

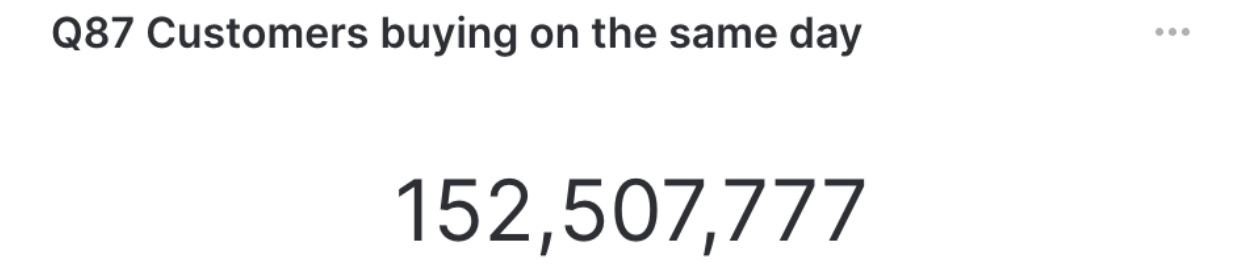
```sql
-- TPC-DS_query86
select
    sum(ws_net_paid) as total_sum
   ,i_category
   ,i_class
   ,grouping(i_category)+grouping(i_class) as lochierarchy
   ,rank() over (
    partition by grouping(i_category)+grouping(i_class),
    case when grouping(i_class) = 0 then i_category end
    order by sum(ws_net_paid) desc) as rank_within_parent
 from
    web_sales
   ,date_dim       d1
   ,item
 where
    d1.d_month_seq between 1205 and 1205+11
 and d1.d_date_sk = ws_sold_date_sk
 and i_item_sk  = ws_item_sk
 group by rollup(i_category,i_class)
 order by
   lochierarchy desc,
   case when lochierarchy = 0 then i_category end,
   rank_within_parent
 limit 100;
```

The breakdown of the query:

1. The query selects columns including the sum of web sales (total_sum), item category (i_category), item class (i_class), a calculated grouping identifier (lochierarchy), and a ranking within each group (rank_within_parent).
2. It retrieves data from the web_sales, date_dim, and item tables, using appropriate joins based on date and item keys.
3. The data is filtered to consider only records from a 12-month period starting in May 1205 (d_month_seq between 1205 and 1205+11).
4. The GROUP BY clause groups the data by item category and class, and it uses the ROLLUP modifier to generate subtotals for various levels of the hierarchy, including both category and class and just category.
5. Within each group, it calculates the rank of each category-class combination based on the sum of web sales, sorting them in descending order.
6. The results are ordered by the hierarchy level (lochierarchy), the category (if it's a parent category), and the rank_within_parent.
7. Finally, the query limits the output to the top 100 rows.
   In summary, this query provides a hierarchical view of web sales data by category and class, computing sales totals and ranking them within their respective parent categories. It allows for a comparison of sales performance among different categories and classes within the specified time frame.

Query 87

Count how many customers have ordered on the same day items on the web and the catalog and on the same day have bought items in a store.




```
-- TPC-DS_query87
select count(*)
from ((select distinct c_last_name, c_first_name, d_date
       from store_sales, date_dim, customer
       where store_sales.ss_sold_date_sk = date_dim.d_date_sk
         and store_sales.ss_customer_sk = customer.c_customer_sk
         and d_month_seq between 1208 and 1208+11)
       except
      (select distinct c_last_name, c_first_name, d_date
       from catalog_sales, date_dim, customer
       where catalog_sales.cs_sold_date_sk = date_dim.d_date_sk
         and catalog_sales.cs_bill_customer_sk = customer.c_customer_sk
         and d_month_seq between 1208 and 1208+11)
       except
      (select distinct c_last_name, c_first_name, d_date
       from web_sales, date_dim, customer
       where web_sales.ws_sold_date_sk = date_dim.d_date_sk
         and web_sales.ws_bill_customer_sk = customer.c_customer_sk
         and d_month_seq between 1208 and 1208+11)
) cool_cust
;
```

The Breakdown of the Query:

1. The query utilizes three subqueries, each responsible for selecting distinct customer names (c_last_name, c_first_name) and purchase dates (d_date) from different sales channels (store_sales, catalog_sales, and web_sales). These subqueries identify customers who made purchases within a 12-month period starting from August 2008 (d_month_seq between 1208 and 1208+11).
2. The "except" operator is used to create set differences between the results of these subqueries. The first subquery identifies customers from physical stores, the second from catalog sales, and the third from web sales.

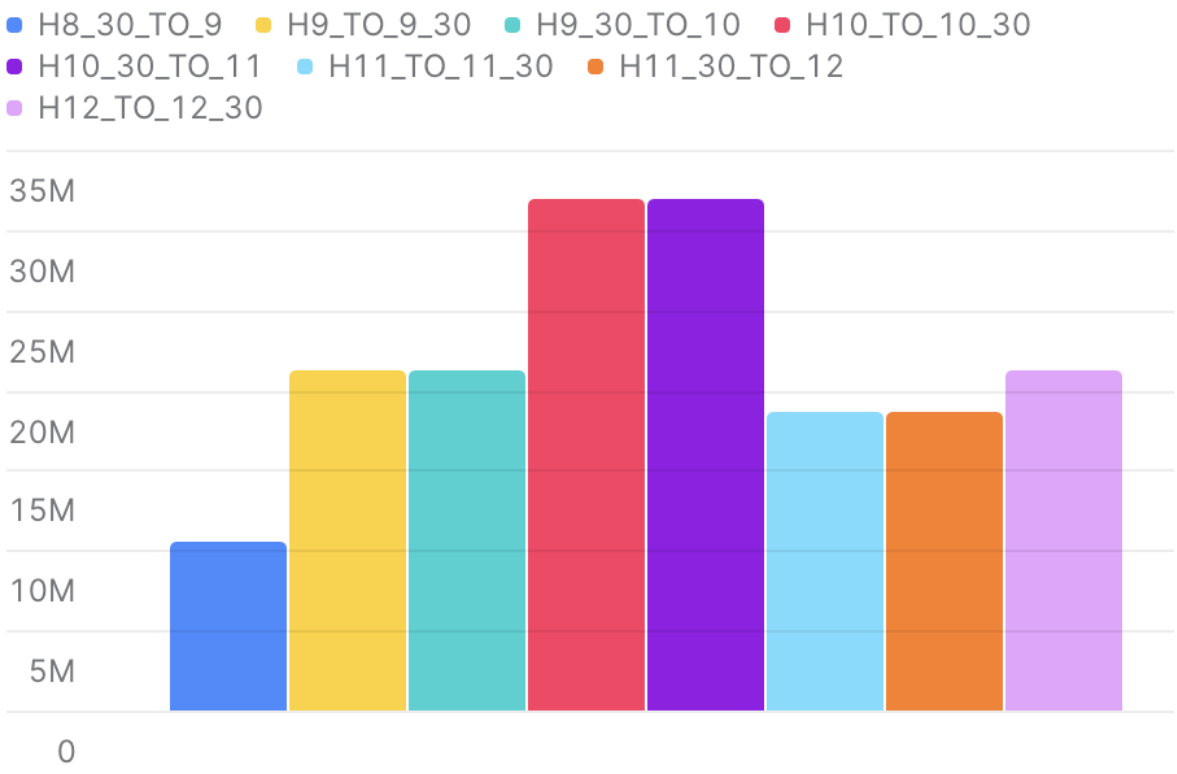
3. The "except" operation ensures that only unique customer names who purchased items from all three channels are retained in the "cool_cust" result set.
4. Finally, the outermost query counts the number of such unique customers, providing the total count of customers who made purchases both online and in physical stores on the same day during the specified time frame.

In summary, this query identifies and counts customers who made purchases from all three sales channels (store, catalog, and web) on the same day within a specific 12-month period, fulfilling the requirement of finding customers with cross-channel purchases.

Query 88

How many items do we sell between pacific times of a day in certain stores to customers with one dependent count and 2 or less vehicles registered or 2 dependents with 4 or fewer vehicles registered or 3 dependents and five or less vehicles registered? In one row break the counts into sells from 8:30 to 9, 9 to 9:30, 9:30 to 10 ... 12 to 12:30

q88 qty sold by timestamps




```
-- TPC-DS_query88
select  *
from
 (select count(*) h8_30_to_9
 from store_sales, household_demographics , time_dim, store
 where ss_sold_time_sk = time_dim.t_time_sk
     and ss_hdemo_sk = household_demographics.hd_demo_sk
     and ss_store_sk = s_store_sk
     and time_dim.t_hour = 8
     and time_dim.t_minute >= 30
     and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
          (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
     and store.s_store_name = 'ese') s1,
 (select count(*) h9_to_9_30
 from store_sales, household_demographics , time_dim, store
 where ss_sold_time_sk = time_dim.t_time_sk
     and ss_hdemo_sk = household_demographics.hd_demo_sk
     and ss_store_sk = s_store_sk
     and time_dim.t_hour = 9
     and time_dim.t_minute < 30
     and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
          (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
          (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
     and store.s_store_name = 'ese') s2,
 (select count(*) h9_30_to_10
 from store_sales, household_demographics , time_dim, store
 where ss_sold_time_sk = time_dim.t_time_sk
     and ss_hdemo_sk = household_demographics.hd_demo_sk
     and ss_store_sk = s_store_sk
     and time_dim.t_hour = 9
     and time_dim.t_minute >= 30
```

```
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s3,
(select count(*) h10_to_10_30
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 10
    and time_dim.t_minute < 30
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s4,
(select count(*) h10_30_to_11
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 10
    and time_dim.t_minute >= 30
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s5,
(select count(*) h11_to_11_30
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 11

    and time_dim.t_minute < 30
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s6,
(select count(*) h11_30_to_12
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 11
    and time_dim.t_minute >= 30
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s7,
(select count(*) h12_to_12_30
from store_sales, household_demographics , time_dim, store
where ss_sold_time_sk = time_dim.t_time_sk
    and ss_hdemo_sk = household_demographics.hd_demo_sk
    and ss_store_sk = s_store_sk
    and time_dim.t_hour = 12
    and time_dim.t_minute < 30
    and ((household_demographics.hd_dep_count = 0 and household_demographics.hd_vehicle_count<=0+2) or
         (household_demographics.hd_dep_count = 1 and household_demographics.hd_vehicle_count<=1+2) or
         (household_demographics.hd_dep_count = -1 and household_demographics.hd_vehicle_count<=-1+2))
    and store.s_store_name = 'ese') s8
;
```

The Breakdown of the Query:

1. The query consists of multiple subqueries, each focusing on a specific time interval, from 8:30 to 12:30.
2. Within each subquery, it counts the number of items sold by joining data from store_sales, household_demographics, time_dim, and store tables. It filters the

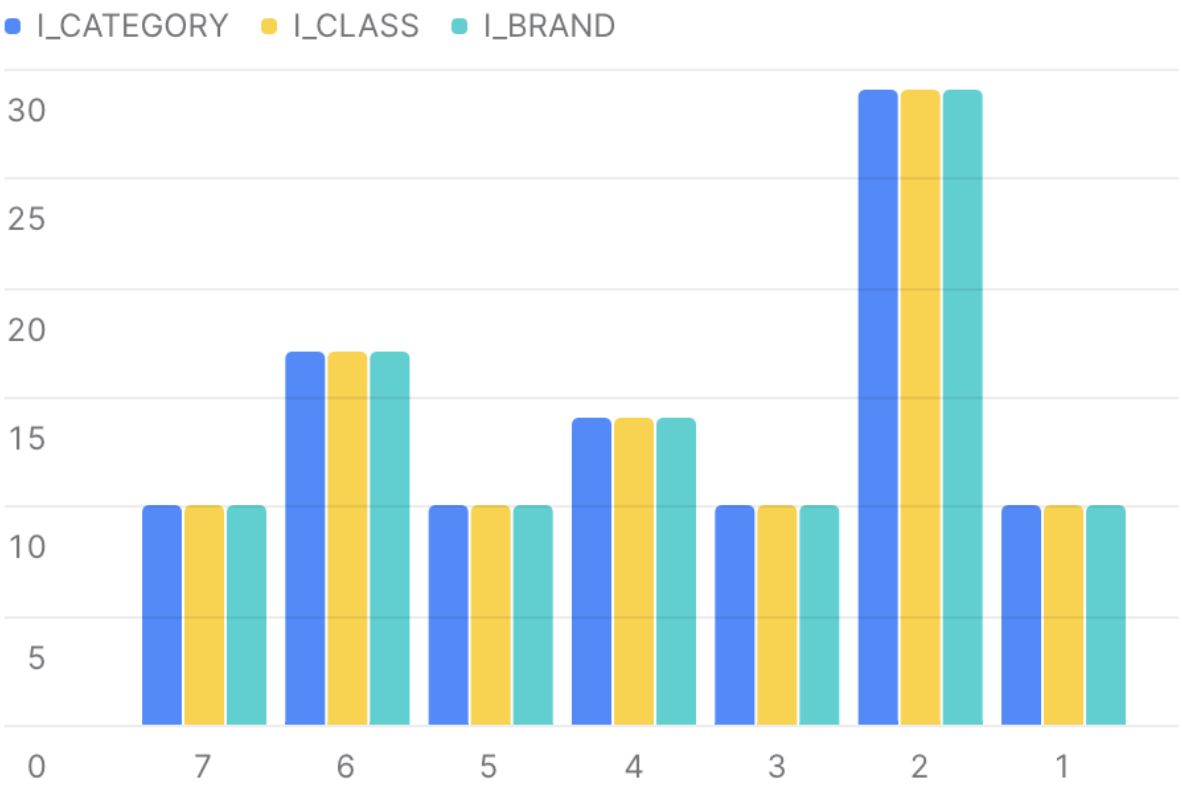
data based on sold time, customer demographics, and the specified store name ('ese').
3. The subqueries are named sequentially (s1, s2, s3, ... s8) to represent each time interval.
4. The final result is a row containing counts for items sold in each time interval, meeting the criteria specified in the question.

In summary, this query provides a breakdown of item sales within specific time windows throughout the morning, catering to customers with specific dependent and vehicle registration counts, all within certain stores. It helps analyze sales patterns during these time intervals for the specified customer demographics.

Query 89

Within a year list all month and combination of item categories, classes and brands that have had monthly sales larger than 0.1 percent of the total yearly sales.

Q89




```sql
-- TPC-DS_query89
select  *
from(
select i_category, i_class, i_brand,
       s_store_name, s_company_name,
       d_moy,
       sum(ss_sales_price) sum_sales,
       avg(sum(ss_sales_price)) over
         (partition by i_category, i_brand, s_store_name, s_company_name)
         avg_monthly_sales
from item, store_sales, date_dim, store
where ss_item_sk = i_item_sk and
      ss_sold_date_sk = d_date_sk and
      ss_store_sk = s_store_sk and
      d_year in (2000) and
        ((i_category in ('Electronics','Women','Home') and
          i_class in ('cameras','maternity','kids')
         )
      or (i_category in ('Sports','Jewelry','Shoes') and
          i_class in ('guns','semi-precious','athletic')
        ))
group by i_category, i_class, i_brand,
         s_store_name, s_company_name, d_moy) tmp1
where case when (avg_monthly_sales <> 0) then (abs(sum_sales - avg_monthly_sales) / avg_monthly_sales) else null end > 0.1
order by sum_sales - avg_monthly_sales, s_store_name
limit 100;
```

The breakdown of the query:

1. The query begins by selecting relevant columns: item category, item class, item brand, store name, company name, month (d_moy), and the sum of store sales (sum_sales).
2. It calculates the average monthly sales (avg_monthly_sales) for each combination of item category, item class, store name, and company name using the window function AVG().
3. The data is retrieved from the item, store_sales, date_dim, and store tables, with appropriate joins on item_sk, sold_date_sk, and store_sk, and filtering for the year 2000.
4. The results are grouped by item category, item class, item brand, store name, company name, and month.
5. A subquery (tmp1) filters the results based on whether the absolute difference between sum_sales and avg_monthly_sales is greater than 10 percent (0.1) of avg_monthly_sales.
6. The final output is ordered by the difference between sum_sales and avg_monthly_sales and store name, with a limit of 100 rows.

In summary, this query identifies and lists the combinations of item categories, classes, and brands that had significant monthly sales fluctuations compared to their average monthly sales within the specified year. It helps highlight instances where sales performance significantly deviates from the norm.

Query 90

What is the ratio between the number of items sold over the internet in the morning (8 to 9am) to the number of items sold in the evening (7 to 8pm) of customers with a specified number of dependents. Consider only websites with a high amount of content.

q90

0.60

```sql
-- TPC-DS_query90
select  cast(amc as decimal(15,4))/cast(pmc as decimal(15,4)) am_pm_ratio
 from ( select count(*) amc
       from web_sales, household_demographics , time_dim, web_page
       where ws_sold_time_sk = time_dim.t_time_sk
         and ws_ship_hdemo_sk = household_demographics.hd_demo_sk
         and ws_web_page_sk = web_page.wp_web_page_sk
         and time_dim.t_hour between 8 and 8+1
         and household_demographics.hd_dep_count = 9
         and web_page.wp_char_count between 5000 and 5200) at,
      ( select count(*) pmc
       from web_sales, household_demographics , time_dim, web_page
       where ws_sold_time_sk = time_dim.t_time_sk
         and ws_ship_hdemo_sk = household_demographics.hd_demo_sk
         and ws_web_page_sk = web_page.wp_web_page_sk
         and time_dim.t_hour between 18 and 18+1
         and household_demographics.hd_dep_count = 9
         and web_page.wp_char_count between 5000 and 5200) pt
 order by am_pm_ratio
 limit 100;
```

Breakdown of the query:

1. Two separate counts are performed, one for items sold in the morning (AM) and another for items sold in the evening (PM). These counts are based on data from web_sales, household_demographics, time_dim, and web_page tables. It filters data based on the time of sale, the number of dependents in households (specifically, 9 dependents), and the content richness of the web pages.
2. The counts for AM (amc) and PM (pmc) are converted to decimal values.
3. The query calculates the ratio (am_pm_ratio) by dividing the AM count by the PM count.
4. Finally, the results are sorted in ascending order by the am_pm_ratio, and only the top 100 rows are displayed.

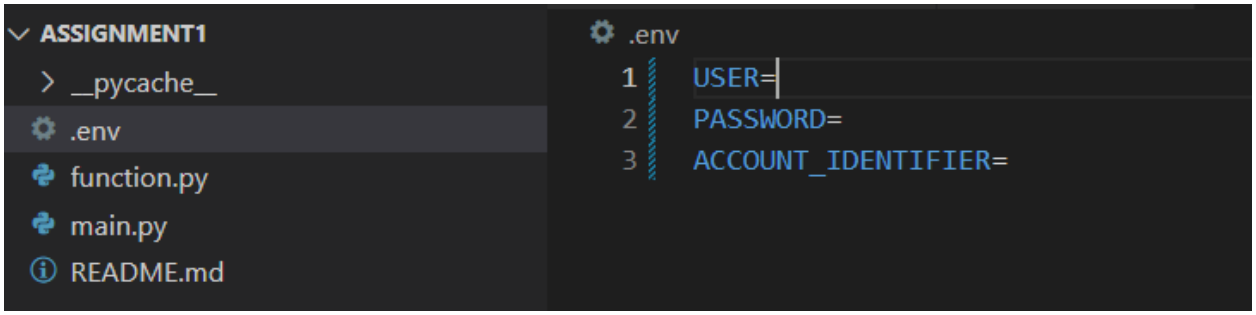
The query helps us understand whether people prefer shopping online in the morning or evening, especially when they have a certain number of dependents, and if the content of the websites they visit influences their choice. The ratio offers insights into their shopping behavior during specific timeframes.

Part 3: Streamlit Snowflake Integration

Imported the necessary libraries:

```
from snowflake.sqlalchemy import URL
from sqlalchemy import create_engine
from datetime import datetime

from dotenv import load_dotenv
from function import *
from urllib import parse
import streamlit as st
import pandas as pd
import numpy as np
import os
```

In the environment, entered the credentials of the account we are working on.



Loaded all environment variables in the main file:

```
#loads all the environment variables
load_dotenv()
```

Created an engine to connect with Snowflake:

```python
# creating the engine
engine = create_engine(URL(
    account=os.getenv('ACCOUNT_IDENTIFIER'),
    user =os.getenv('USER'),
    password=parse.quote(os.getenv('PASSWORD')),
    database = 'SNOWFLAKE_SAMPLE_DATA',
    schema = 'TPCDS_SF10TCL',
    warehouse = 'STREAMLIT',
    role='ACCOUNTADMIN',
    timezone = 'America/Los_Angeles',
))

#connecting with snowflake
connection = engine.connect()
```

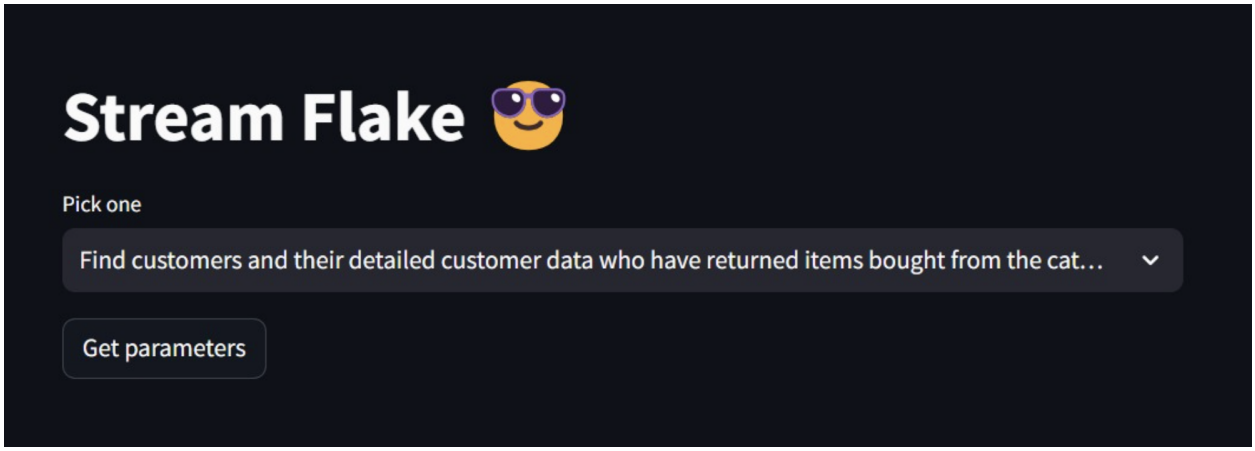
We created an array which stores all the questions of the queries

```python
# creating an aray of all the questions
question_queries = [
    "Find customers and their detailed customer data who have returned items bought from the catalog more than 20 percent the average customer returns for customers in a given state in a given time period. Order output by customer data.",
    "Find customers who tend to spend more money (net-paid) on-line than in stores.",
    "Retrieve the items with the highest number of returns where the number of returns was approximately equivalent across all store, catalog and web channels (within a tolerance of +/- 10%), within the week ending a given date",
    "List all customers living in a specified city, with an income between 2 values.",
    "For all web return reason calculate the average sales, average refunded cash and average return fee by different combinations of customer and sales types (e.g., based on marital status, education status, state and sales profit).",
    "Rollup the web sales for a given year by category and class, and rank the sales among peers within the parent, for each group compute sum of sales, location with the hierarchy and rank within the group.",
    "Count how many customers have ordered on the same day items on the web and the catalog and on the same day have bought items in a store",
    "How many items do we sell between pacific times of a day in certain stores to customers with one dependent count and 2 or less vehicles registered or 2 dependents with 4 or fewer vehicles registered or 3 dependents and five or less vehicles registered. In one row break the counts into sells from 8:30 to 9, 9 to 9:30, 9:30 to 10 ... 12 to 12:30",
    "Within a year list all month and combination of item categories, classes and brands that have had monthly sales larger than 0.1 percent of the total yearly sales.",
    "What is the ratio between the number of items sold over the internet in the morning (8 to 9am) to the number of items sold in the evening (7 to 8pm) of customers with a specified number of dependents"ßß
]
```

We pass this array inside a select box to created the initial UI of the webpage to display the select box which contains all the queries and a button to generate a form which provides the option to change the parameters inside a query.

```
# making the form
option_selected=st.selectbox("Pick one", question_queries)
get_parameters= st.button('Get parameters',on_click=dis,args=(1,))
```

The UI:



As soon as the user selects the option the show_parameters function is called which is explained below

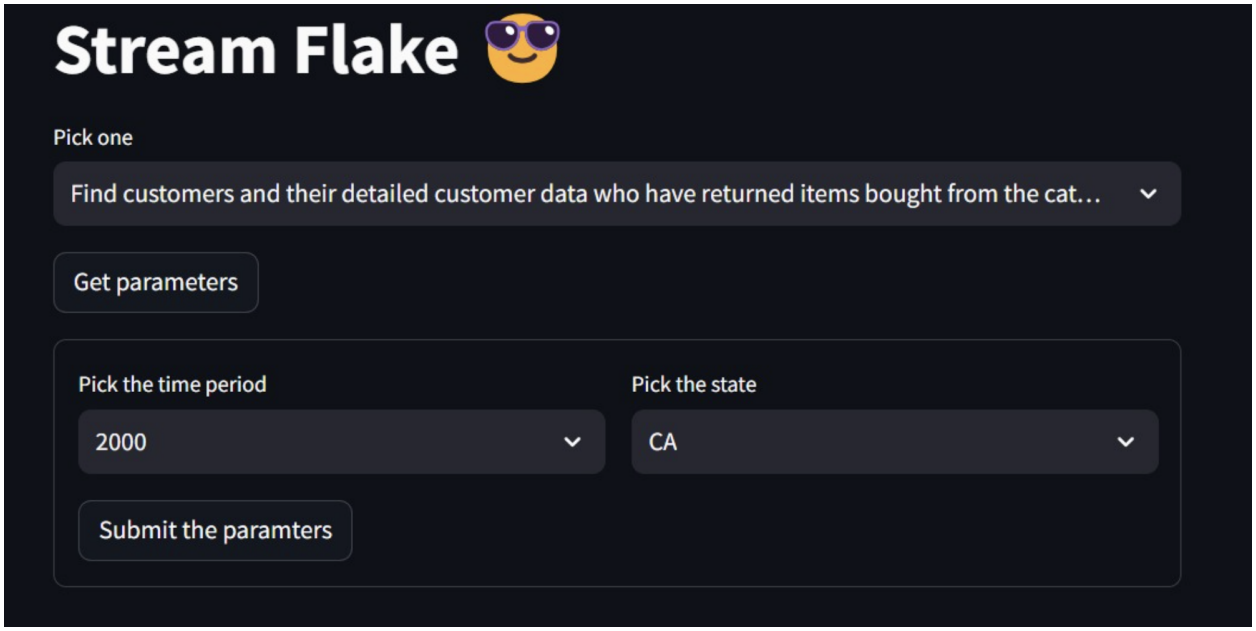
```
if(option_selected==question_queries[0]):
    time_period_year_select,ca_state_select=show_parameters_1()
```

For each query, we created a function to take user input value and return it. In that function we also created different widgets which help the user to provide its value. Inorder to display the different widgets, in some functions we have followed the Column layout of streamlit.

In below example we created a function named show_paramters_1. We added the column structure by specifying the number of columns to be created.In this example we created 2 columns col1 and col2 In col1 we made a select box to select ‘Year’ and in col2 we made a select box to select ‘State’. In the select box we need to give the list of options which can be selected by the user, we do this by selecting the unique values from the data set and provided that to the user so he/she can chose from. Lastly there is also a submit button which submits all the parameters and the param function is called when clicked

```
def show_parameters_1():
        with st.form(key='parameters_1'):
            col1,col2=st.columns(2)

            time_period_year=pd.read_sql_query("Select distinct(d_year) from date_dim order by d_year;",engine)
            ca_state=pd.read_sql_query(" Select distinct(ca_state) from customer_address order by ca_state",engine)

            with col1:
             time_period_year_select=st.selectbox("Pick the time period",time_period_year)




            with col2:
             ca_state_select=st.selectbox("Pick the state",ca_state)

            submit_query_1=st.form_submit_button('Submit the paramters',on_click=param,args=(1,))

        return time_period_year_select,ca_state_select
```

The UI of above code



Initially as the page loads we had stored some session variables and intialized it to 0. As soon as the user clicks on submit button the param function gets activated which changes the state to 1 and the entire file gets re run This re-run of the file is due to the default behaviour of streamlit

```
##initally storing the session variables
if 'stage' not in st.session_state:
    st.session_state.stage = 0

if 'runquery' not in st.session_state:
    st.session_state.runquery =0


#function to update the session variables
def param(runquery):
    st.session_state.runquery = runquery
def dis(stage):
    st.session_state.stage = stage
```

As soon as the session state gets updated the query corresponding to the option selected gets run and whatever the result (a dataframe) is then displayed on the website

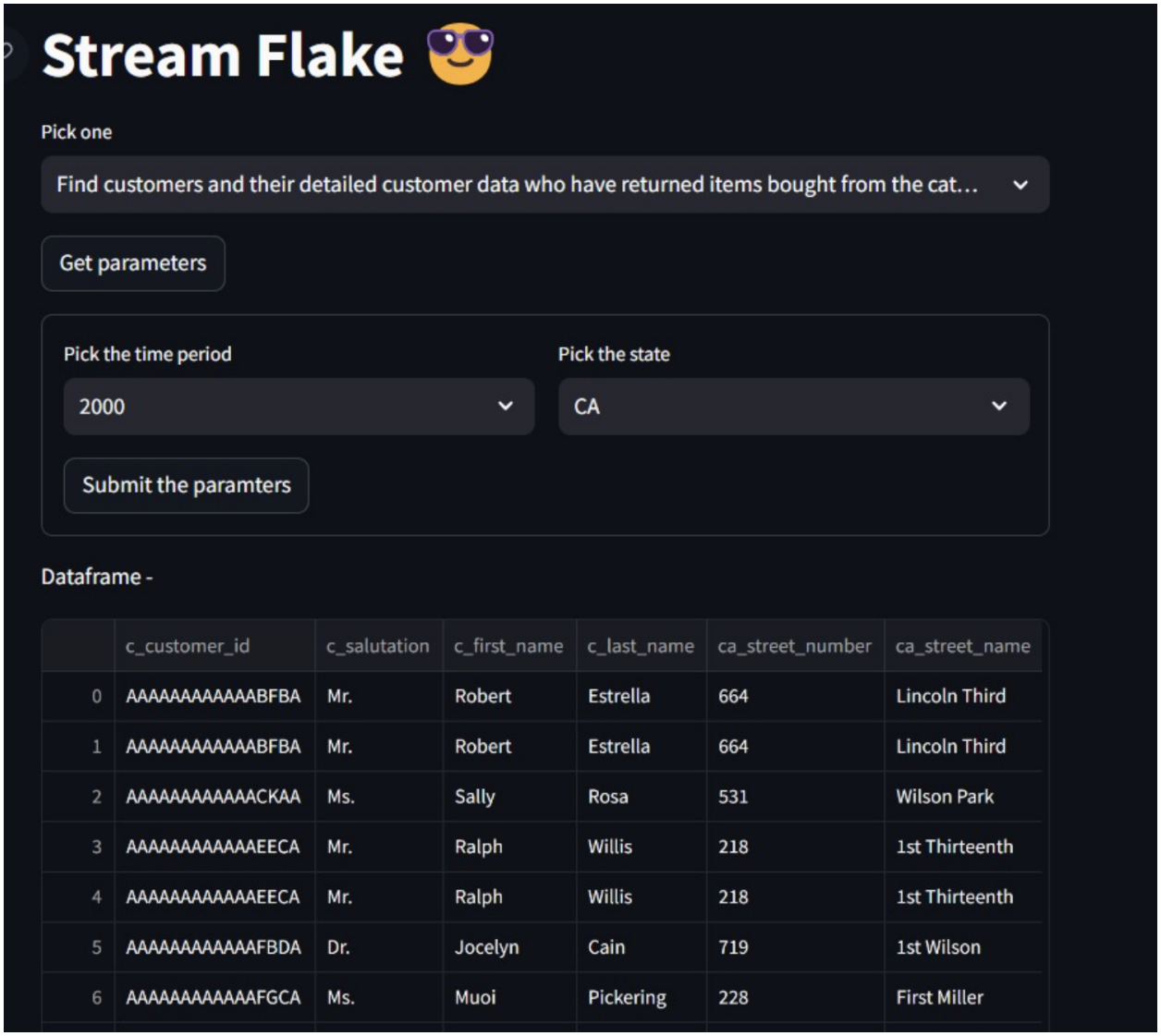
In the below code as soon as the user selects the first option and submit the parameters the read_sql_query of pandas library gets called which takes an sql query as the input (Instead of a direct input we have provided a function for each query which gets called) and displays the dataframe
In case if the dataframe is empty “No results found” gets displayed

```
if st.session_state.runquery==1:
    if(option_selected==question_queries[0]):
                    try:
                        df=pd.read_sql_query(run_query_1(time_period_year_select,ca_state_select),engine)

                        if df.empty:
                           st.write("No results found")
                        else:
                           st.write(df)

                    finally:

                        connection.close()
                        engine.dispose()
```

This is the run_query_1 function which takes the 2 parameters (year and state) and returns the sql code with the updated parameters

```python
def run_query_1(time_period_year,customer_state):
    sql_query_1=f"with customer_total_return as (select cr_returning_customer_sk as ctr_customer_sk ,ca_state as ctr_state, sum
    (cr_return_amt_inc_tax) as ctr_total_return from catalog_returns ,date_dim ,customer_address where cr_returned_date_sk =
    d_date_sk and d_year ={time_period_year} and cr_returning_addr_sk = ca_address_sk group by cr_returning_customer_sk ,ca_state )
    select  c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name ,ca_street_type,ca_suite_number,
    ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset ,ca_location_type,ctr_total_return from customer_total_return ctr1 ,
    customer_address ,customer where ctr1.ctr_total_return > (select avg(ctr_total_return)*1.2 from customer_total_return ctr2 where
    ctr1.ctr_state = ctr2.ctr_state) and ca_address_sk = c_current_addr_sk and ca_state = '{customer_state}' and ctr1.ctr_customer_sk
    = c_customer_sk order by c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name ,ca_street_type,
    ca_suite_number,ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset ,ca_location_type,ctr_total_return limit 100;"

    return sql_query_1
```

Final UI



	c_customer_id	c_salutation	c_first_name	c_last_name	ca_street_number	ca_street_name
0	AAAAAAAAAAAABFBA	Mr.	Robert	Estrella	664	Lincoln Third
1	AAAAAAAAAAAABFBA	Mr.	Robert	Estrella	664	Lincoln Third
2	AAAAAAAAAAAACKAA	Ms.	Sally	Rosa	531	Wilson Park
3	AAAAAAAAAAAAEECA	Mr.	Ralph	Willis	218	1st Thirteenth
4	AAAAAAAAAAAAEECA	Mr.	Ralph	Willis	218	1st Thirteenth
5	AAAAAAAAAAAAFBDA	Dr.	Jocelyn	Cain	719	1st Wilson
6	AAAAAAAAAAAAFGCA	Ms.	Muoi	Pickering	228	First Miller

Similarly Each query has a run_query_function and show_parameter function. Below are some of the snippets of the entier file

function.py

```python
def run_query_1(time_period_year,customer_state):
    sql_query_1=f"with customer_total_return as (select cr_returning_customer_sk as ctr_customer_sk ,ca_state as ctr_state, sum(cr_return_amt_inc_tax) as ctr_total_return from catalog_returns ,date_dim ,customer_address where cr_returned_date_sk = d_date_sk and d_year ={time_period_year} and cr_returning_addr_sk = ca_address_sk group by cr_returning_customer_sk ,ca_state ) select  c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name ,ca_street_type,ca_suite_number, ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset ,ca_location_type,ctr_total_return from customer_total_return ctr1 ,customer_address ,customer where ctr1. ctr_total_return > (select avg(ctr_total_return)*1.2 from customer_total_return ctr2 where ctr1.ctr_state = ctr2.ctr_state) and ca_address_sk = c_current_addr_sk and ca_state = '{customer_state}' and ctr1.ctr_customer_sk = c_customer_sk order by c_customer_id,c_salutation,c_first_name,c_last_name,ca_street_number,ca_street_name , ca_street_type,ca_suite_number,ca_city,ca_county,ca_state,ca_zip,ca_country,ca_gmt_offset ,ca_location_type,ctr_total_return limit 100;"

    return sql_query_1

def run_query_2(mid1,mid2,mid3,mid4,date,price,add_price):
  sql_query_2=f"select  i_item_id ,i_item_desc ,i_current_price from item, inventory, date_dim, store_sales where i_current_price between {price} and {add_price} and inv_item_sk = i_item_sk and d_date_sk=inv_date_sk and d_date between cast('{date}' as date) and dateadd(day,60,to_date('{date}')) and i_manufact_id in ({mid1},{mid2}, {mid3},{mid4}) and inv_quantity_on_hand between 100 and 500 and ss_item_sk = i_item_sk group by i_item_id,i_item_desc,i_current_price order by i_item_id limit 100;"
  return sql_query_2


def run_query_3(return_date_1, return_date_2, return_date_3):

  sql_query_3=f"with sr_items as (select i_item_id item_id, sum(sr_return_quantity) sr_item_qty from store_returns, item, date_dim where sr_item_sk = i_item_sk and d_date    in (select d_date from date_dim where d_week_seq in (select d_week_seq from date_dim where d_date in ('{return_date_1}', '{return_date_2}', '{return_date_3}'))) and   sr_returned_date_sk   = d_date_sk group by i_item_id), cr_items as (select i_item_id item_id, sum(cr_return_quantity) cr_item_qty from catalog_returns, item, date_dim where cr_item_sk = i_item_sk and   d_date    in (select d_date from date_dim where d_week_seq in (select d_week_seq from date_dim where d_date in ('{return_date_1}', '{return_date_2}','{return_date_3}'))) and   cr_returned_date_sk   = d_date_sk group by i_item_id), wr_items as (select i_item_id item_id, sum(wr_return_quantity) wr_item_qty from web_returns, item, date_dim where wr_item_sk = i_item_sk and   d_date    in (select d_date from date_dim where d_week_seq in (select d_week_seq from date_dim where d_date in ('1998-02-20','1998-09-28','1998-11-14'))) and   wr_returned_date_sk   = d_date_sk group by i_item_id) select  sr_items.item_id ,sr_item_qty , sr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 sr_dev ,cr_item_qty ,cr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 cr_dev ,wr_item_qty , wr_item_qty/(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 * 100 wr_dev ,(sr_item_qty+cr_item_qty+wr_item_qty)/3.0 average from sr_items ,cr_items ,wr_items where sr_items. item_id=cr_items.item_id and sr_items.item_id=wr_items.item_id order by sr_items.item_id ,sr_item_qty limit 100;"
  return sql_query_3

def run_query_4(city, income,add_income):
  sql_query_4=f"select  c_customer_id as customer_id , coalesce(c_last_name,'') || ', ' || coalesce(c_first_name,'') as customername from customer ,customer_address , customer_demographics ,household_demographics ,income_band ,store_returns where ca_city         =  '{city}'and c_current_addr_sk = ca_address_sk and ib_lower_bound   >= {income} and ib_upper_bound   <= {add_income}   and ib_income_band_sk = hd_income_band_sk and cd_demo_sk = c_current_cdemo_sk and hd_demo_sk = c_current_hdemo_sk and sr_cdemo_sk = cd_demo_sk order by c_customer_id limit 100;"

  return sql_query_4

def run_query_5(maratial_status,education_status,year,state_1,state_2,state_3):
   sql_query_5 = f" select  substr(r_reason_desc,1,20) ,avg(ws_quantity) ,avg(wr_refunded_cash) ,avg(wr_fee) from web_sales, web_returns, web_page, customer_demographics cd1, customer_demographics cd2, customer_address, date_dim, reason where ws_web_page_sk = wp_web_page_sk and ws_item_sk = wr_item_sk and ws_order_number = wr_order_number and ws_sold_date_sk = d_date_sk and d_year = {year} and cd1.cd_demo_sk = wr_refunded_cdemo_sk and cd2.cd_demo_sk = wr_returning_cdemo_sk and ca_address_sk = wr_refunded_addr_sk and r_reason_sk = wr_reason_sk and ( cd1.cd_marital_status = '{maratial_status}' and cd1.cd_marital_status = cd2.cd_marital_status and cd1.
```

```
        return time_period_year_select,ca_state_select

def show_parameters_2():
        with st.form(key='paramers_2'):
            manx_id=pd.read_sql_query("select Distinct(i_manufact_id) from item order by i_manufact_id;",engine)
            manx_id.dropna(subset=['i_manufact_id'], inplace=True)



            manx_id['i_manufact_id'] = manx_id['i_manufact_id'].astype(int)

            print(manx_id)
            # st.write(man_id)
            # print(man_id)
            i_price=pd.read_sql_query(" select distinct(i_current_price) from item order by i_current_price;",engine)
            col1,col2=st.columns(2)
            max_date = datetime(2100, 1, 1)
            min_date=datetime(1900,1,2)
            with col1:
                 m_id_1=st.selectbox("Pick the first manufacture",manx_id,key="1")
                 m_id_2=st.selectbox("Pick the second manufacture id",manx_id,key="2")
                 first_date = st.date_input("Select first date ")

            with col2:
                m_id_3=st.selectbox("Pick the third manufacture id",manx_id)
                m_id_4=st.selectbox("Pick the fouth manufacture id",manx_id)
                price=st.selectbox("Pick the fouth price id",i_price)

            submit_query_2=st.form_submit_button('Submit the paramters',on_click=param,args=(1,))

        return m_id_1,m_id_2,m_id_3,m_id_4,price,first_date

def show_parameters_3():
        with st.form(key='parameters_3'):
            col1,col2=st.columns(2)
            max_date = datetime(2100, 1, 1)
            min_date=datetime(1900,1,2)
            with col1:
                 first_date = st.date_input("Select first date ",max_value=max_date,min_value=min_date)
                 second_date=st.date_input("Select second date ",max_value=max_date,min_value=min_date)

            with col2:
                 third_date=st.date_input("Select third date",min_value=min_date,max_value=max_date)

            submit_query_2=st.form_submit_button('Submit the paramters',on_click=param,args=(1,))

        return first_date,second_date,third_date
```