SQL

Suppose you have a table named orders with columns delivery_date, restaurant_id, and order_id. Write a SQL query to output the result: For each day in Aug 2022, find the daily unique restaurant count, who have at least one order delivered that day.

(a) Among the restaurants from each day, how many of them have at least one order delivered on the previous day as well?

(b) Among the restaurants from each day, how many of them have at least one order delivered within the past 7 days as well?

Sample output:

delivery_date restaurant_count restaurant_count_ordered_1d restaurant_count_ordered_7d

2022-08-01 100 0 80

…

2022-08-30 125 10 40

2022-08-31 120 30 30

SQL

```
: !pip install pandasql
```

```
Requirement already satisfied: pandasql in d:\anaconda3\lib\site-packages (0.7.3)
Requirement already satisfied: sqlalchemy in d:\anaconda3\lib\site-packages (from pandasq
Requirement already satisfied: numpy in c:\users\user\appdata\roaming\python\python37\sit
Requirement already satisfied: pandas in d:\anaconda3\lib\site-packages (from pandasql) (
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\user\appdata\roaming\py
as->pandasql) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in c:\users\user\appdata\roaming\python\pytho
ql) (2022.2.1)
Requirement already satisfied: six>=1.5 in c:\users\user\appdata\roaming\python\python37\
2.7.3->pandas->pandasql) (1.16.0)
```

```
: from pandasql import sqldf
  import pandas as pd
```

```
: data = [["2022-8-1","ID_1", "1"],
          ["2022-8-1","ID_1", "2"],
          ["2022-8-1","ID_2", "1"],
          ["2022-8-1","ID_2", "2"],
          ["2022-8-1","ID_2", "3"],
          ["2022-8-2","ID_1", "11"],
          ["2022-8-2","ID_1", "12"],
          ["2022-8-3","ID_2", "21"],
          ["2022-8-3","ID_2", "12"],
          ["2022-8-4","ID_2", "23"]]

  data = pd.DataFrame(data=data, columns=['delivery_date', 'restaurant_id', 'order_id'])
  data['delivery_date'] = pd.to_datetime(data['delivery_date'])
  data
```

	delivery_date	restaurant_id	order_id
0	2022-08-01	ID_1	1
1	2022-08-01	ID_1	2
2	2022-08-01	ID_2	1
3	2022-08-01	ID_2	2
4	2022-08-01	ID_2	3
5	2022-08-02	ID_1	11
6	2022-08-02	ID_1	12
7	2022-08-03	ID_2	21
8	2022-08-03	ID_2	12
9	2022-08-04	ID_2	23

```
q = "select delivery_date, count(distinct  restaurant_id) as restaurant_count \
     from data where strftime('%m', delivery_date) = '08' group by delivery_date"
sqldf(q, globals())
```

	delivery_date	restaurant_count
0	2022-08-01 00:00:00.000000	2
1	2022-08-02 00:00:00.000000	1
2	2022-08-03 00:00:00.000000	1
3	2022-08-04 00:00:00.000000	1

```
q = "select delivery_date, count(distinct  restaurant_id) as restaurant_count, \
     (select count(order_id) from data where strftime('%d', delivery_date))-1 = strftime('%d', delivery_date) \
     as restaurant_count_ordered_1d \
     from data where strftime('%m', delivery_date) = '08' group by delivery_date"
sqldf(q, globals())
```

	delivery_date	restaurant_count	restaurant_count_ordered_1d
0	2022-08-01 00:00:00.000000	2	0
1	2022-08-02 00:00:00.000000	1	0
2	2022-08-03 00:00:00.000000	1	0
3	2022-08-04 00:00:00.000000	1	0

Selecting records of previous and find count of all orders where count(order_id) > 0

Python

Suppose we place PO (purchase order) to each vendor on a weekly set schedule specified by (order_days, receive_days and lead_time_days). For instance, a vendor may have the schedule as ([Monday, Wednesday], [Wednesday, Thursday], 8) - place PO on Monday or Wednesday, receive PO on Wednesday or Thursday, and the number of days between PO receipt and PO placement is at least 8 days. If today is Aug 31 (Wednesday), then the upcoming PO placement day is Aug 31 and its receipt day is Sep 8 (Thursday). Write a Python function to output the upcoming PO placement day and its receipt day, given the vendor schedule and today's date: upcoming_order_day(order_days, receive_days, lead_time_days, today_date) = [po_placement_day, po_receipt_day]. Assume order_days and receive_days are lists, lead_time_days is int, and today_date is date.

Sample output: upcoming_order_day([Monday, Wednesday], [Wednesday, Thursday], 8, datetime.date(2022, 8, 31)) = [datetime.date(2022, 8, 31), datetime.date(2022, 9, 8)].

```
from datetime import datetime, timedelta
import calendar

def upcoming_order_day(place_PO, recieve_PO, Days, Today_date):
    result = []

    days = [Today_date+timedelta(i) for i in [0, 1, 2, 3, 4, 5, 6, 7]]

    start_day = None

    for i in days:
        if calendar.day_name[i.weekday()] in place_PO:
            start_day = i
            break

    recieve_day = start_day + timedelta(Days)

    days = [recieve_day+timedelta(i) for i in [0, 1, 2, 3, 4, 5, 6, 7]]

    recieve_day = None

    for i in days:
        if calendar.day_name[i.weekday()] in recieve_PO:
            recieve_day = i
            break

    result = [start_day.date(), recieve_day.date()]

    return "upcoming_order_day("+str(place_PO)+", "+str(recieve_PO)+", "+str(Days)+",
"+str(Today_date.date())+" = "+str(result)
```

```
from datetime import datetime, timedelta
import calendar

def upcoming_order_day(place_PO, recieve_PO, Days, Today_date):
    result = []

    days = [Today_date+timedelta(i) for i in [0, 1, 2, 3, 4, 5, 6, 7]]

    start_day = None

    for i in days:
        if calendar.day_name[i.weekday()] in place_PO:
            start_day = i
            break

    recieve_day = start_day + timedelta(Days)

    days = [recieve_day+timedelta(i) for i in [0, 1, 2, 3, 4, 5, 6, 7]]

    recieve_day = None

    for i in days:
        if calendar.day_name[i.weekday()] in recieve_PO:
            recieve_day = i
            break

    result = [start_day.date(), recieve_day.date()]

    return "upcoming_order_day("+str(place_PO)+", "+str(recieve_PO)+", "+str(Days)+\
           ", "+str(Today_date.date())+" = "+str(result)
```

```
upcoming_order_day(["Monday", "Wednesday"], ["Wednesday", "Thursday"], 8, datetime.today())
```

```
"upcoming_order_day(['Monday', 'Wednesday'], ['Wednesday', 'Thursday'], 8, 2022-10-08 = [datetime.date(2022, 10, 10), datetime.
date(2022, 10, 19)]"
```

```
upcoming_order_day(["Sunday", "Wednesday"], ["Wednesday", "Thursday"], 8, datetime.today())
```

```
"upcoming_order_day(['Sunday', 'Wednesday'], ['Wednesday', 'Thursday'], 8, 2022-10-08 = [datetime.date(2022, 10, 9), datetime.d
ate(2022, 10, 19)]"
```

Optimization

Suppose we need to procure inventory for N items from M vendors. We need to procure at least $R\ n$ amount for each item $n \in \{1,..., N\}$. Each vendor m may charge the item n at different prices P where $nmn \in \{1,..., N\}$ and $m \in \{1,..., M\}$. In addition, there is a fixed cost, such as freight, associated with the vendor procurement C for each vendor . The fixed cost is incurred only when we procure at $m\ m \in \{1,..., M\}$ least one item from the given vendor.

(a) Formulate a MILP (Mixed Integer Linear Programming) problem to find the solution for x the nm amount of item n we should procure from vendor m to minimize the total cost.

(b) Now suppose there is a vendor contract stating that, if we procure from vendors A and B, we must procure from vendor C as well. Write the additional constraint(s) needed to the MILP problem.

There are a lot of python packages available for optimization but I will give you an example if scipy for optimization. The overall idea will remain same, the only change will be problem formulation. Following are the general steps to solve optimization problem:-

Lets Minimize following

$$
\begin{aligned}
&\text{minimize} && -z = -x - 2y \\
&\text{subject to:} && 2x + y \leq 20 \\
& && -4x + 5y \leq 10 \\
& && x - 2y \leq 2 \\
& && -x + 5y = 15 \\
& && x \geq 0 \\
& && y \geq 0
\end{aligned}
$$

Define all the input variables
Define boundary of each variable i.e. the upper and lower limit

Example

Optimization

```
from scipy.optimize import linprog
```

$$
\begin{aligned}
&\text{maximize} && z = x + 2y \\
&\text{subject to:} && 2x + y \leq 20 \\
& && -4x + 5y \leq 10 \\
& && -x + 2y \geq -2 \\
& && -x + 5y = 15 \\
& && x \geq 0 \\
& && y \geq 0
\end{aligned}
$$

```
obj = [-1, -2]
#      ┬  ┬
#      │  └┤ Coefficient for y
#      └──┤ Coefficient for x

lhs_ineq = [[ 2,  1],  # Red constraint left side
            [-4,  5],  # Blue constraint left side
            [ 1, -2]]  # Yellow constraint left side

rhs_ineq = [20,  # Red constraint right side
            10,  # Blue constraint right side
             2]  # Yellow constraint right side

lhs_eq = [[-1, 5]]  # Green constraint left side
rhs_eq = [15]       # Green constraint right side
```

```
obj holds the coefficients from the objective function.
lhs_ineq holds the left-side coefficients from the inequality (red, blue, and yellow) constraints.
rhs_ineq holds the right-side coefficients from the inequality (red, blue, and yellow) constraints.
lhs_eq holds the left-side coefficients from the equality (green) constraint.
rhs_eq holds the right-side coefficients from the equality (green) constraint.
```

```
bnd = [(0, float("inf")),  # Bounds of x
       (0, float("inf"))]  # Bounds of y
```

```
opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
              A_eq=lhs_eq, b_eq=rhs_eq, bounds=bnd,
              method="revised simplex")
opt
```

```
opt = linprog(c=obj, A_ub=lhs_ineq, b_ub=rhs_ineq,
              A_eq=lhs_eq, b_eq=rhs_eq, bounds=bnd,
              method="revised simplex")
opt
```

```
     con: array([0.])
     fun: -16.818181818181817
 message: 'Optimization terminated successfully.'
     nit: 3
   slack: array([ 0.        , 18.18181818,  3.36363636])
  status: 0
 success: True
       x: array([7.72727273, 4.54545455])
```

In our example we will define ITEM “n” as one variable and VENDER “m” as other variable instead of “x” and “y”.

Then we will define upper and lower bound using

1<= n <= N
And
1<=m<=N

Similarly we will define constraints then and solve as we have solved above problem.

Machine Learning

Suppose you're given a set of N number of scores normalized between 0 and 1 for each restaurant customer, describing their purchasing and engagement behavior.

(a) What dimension reduction algorithms can you use to derive a single score between 0 and 1 to represent each restaurant customer. In other words, how do you come up with the weight assigned to each score to calculate a weighted score at the end. Are there any advantages/disadvantages associated with each of the algorithms?

DIMENSIONALITY REDUCTION

Dimensionality reduction refers to the technique of reducing the dimension of a data feature set. Usually, machine learning datasets (feature set) contain hundreds of columns (i.e., features) or an array of points, creating a massive sphere in a three-dimensional space. By applying dimensionality reduction, you can decrease or bring down the number of columns to quantifiable counts, thereby transforming the three-dimensional sphere into a two-dimensional object.
There are many PCA techniques in literature, but some most frequently used are:-

Principal Component Analysis (PCA)

Principal Component Analysis is one of the leading linear techniques of dimensionality reduction. This method performs a direct mapping of the data to a lesser dimensional space in a way that maximizes the variance of the data in the low-dimensional representation. Essentially, it is a statistical procedure that orthogonally converts the 'n' coordinates of a dataset into a new set of n coordinates, known as the principal components. This conversion results in the creation of the first principal component having the maximum variance. Each succeeding principal component bears the highest possible variance, under the condition that it is orthogonal (not correlated) to the preceding components.

Pros and Cons of PCA

The PCA conversion is sensitive to the relative scaling of the original variables. Thus, the data column ranges must first be normalized before implementing the PCA method. Another thing to remember is that using the PCA approach will make your dataset lose its interpretability. So, if interpretability is crucial to your analysis, PCA is not the right dimensionality reduction method for your project.

Non-negative matrix factorization (NMF)

NMF breaks down a non-negative matrix into the product of two non-negative ones. This is what makes the NMF method a valuable tool in areas that are primarily concerned with non-negative signals (for instance, astronomy). The multiplicative update rule by Lee & Seung improved the NMF technique by – including uncertainties, considering missing data and parallel computation, and sequential construction.

Pros and Cons of NMF

These inclusions contributed to making the NMF approach stable and linear. Unlike PCA, NMF does not eliminate the mean of the matrices, thereby creating unphysical non-negative fluxes. Thus, NMF can preserve more information than the PCA method. Sequential NMF is characterized by a stable component base during construction and a linear modeling process. This makes it the perfect tool in astronomy. Sequential NMF can preserve the flux in the direct imaging of circumstellar structures in astronomy, such as detecting exoplanets and direct imaging of circumstellar disks.

Linear discriminant analysis (LDA)

The linear discriminant analysis is a generalization of Fisher's linear discriminant method that is widely applied in statistics, pattern recognition, and machine learning. The LDA technique aims to find a linear combination of features that can characterize or differentiate between two or more classes of objects. LDA represents data in a way that maximizes class separability. While objects belonging to the same class are juxtaposed via projection, objects from different classes are arranged far apart.

Pros and Cons of LDA

Simple prototype classifier: Distance to the class mean is used, it's simple to interpret. The decision boundary is linear: It's simple to implement and the classification is robust. Dimension reduction: It provides an informative low-dimensional view on the data, which is both useful for visualization and feature engineering.

Linear decision boundaries may not adequately separate the classes. Support for more general boundaries is desired. In a high-dimensional setting, LDA uses too many parameters. A regularized version of LDA is desired. Support for more complex prototype classification is desired.

Generalized discriminant analysis (GDA)

The generalized discriminant analysis is a nonlinear discriminant analysis that leverages the kernel function operator. Its underlying theory matches very closely to that of support vector machines (SVM), such that the GDA technique helps to map the input vectors into high-dimensional feature space. Just like the LDA approach, GDA also seeks to find a projection for variables in a lower-dimensional space by maximizing the ratio of between-class scatters to within-class scatter.

Pros and Cons of GDA

Well-designed for multiclass problems, Closed form solution, meaning computationally efficient, Outputs class probabilities for each label and Does not require much hyperparameter tuning

Assumes multivariate normality among features, though it is relatively robust to this assumption and Not as well suited for mixed data types as decision tree based methods

Missing Values Ratio (MVR)

When you explore a given dataset, you might find that there are some missing values in the dataset. The first step in dealing with missing values is to identify the reason behind them. Accordingly, you can then impute the missing values or drop them altogether by using the befitting methods. This approach is perfect for situations when there are a few missing values.

Pros and Cons of MVR

However, what to do when there are too many missing values, say, over 50%? In such situations, you can set a threshold value and use the missing values ratio method. The higher the threshold value, the more aggressive will be the dimensionality reduction. If the percentage of missing values in a variable exceeds the threshold, you can drop the variable. Generally, data columns having numerous missing values hardly contain useful information. So, you can remove all the data columns having missing values higher than the set threshold.

High Correlation Filter

If a dataset consists of data columns having a lot of similar patterns/trends, these data columns are highly likely to contain identical information. Also, dimensions that depict a higher correlation can

adversely impact the model's performance. In such an instance, one of those variables is enough to feed the ML model.

For such situations, it's best to use the Pearson correlation matrix to identify the variables showing a high correlation. Once they are identified, you can select one of them using VIF (Variance Inflation Factor). You can remove all the variables having a higher value (VIF > 5). In this approach, you have to calculate the correlation coefficient between numerical columns (Pearson's Product Moment Coefficient) and between nominal columns (Pearson's chi-square value). Here, all the pairs of columns having a correlation coefficient higher than the set threshold will be reduced to 1.

Pros and Cons of Correlation

Since correlation is scale-sensitive, you must perform column normalization.

Backward Feature Elimination

In the backward feature elimination technique, you have to begin with all 'n' dimensions. Thus, at a given iteration, you can train a specific classification algorithm is trained on n input features. Now, you have to remove one input feature at a time and train the same model on n-1 input variables n times. Then you remove the input variable whose elimination generates the smallest increase in the error rate, which leaves behind n-1 input features. Further, you repeat the classification using n-2 features, and this continues till no other variable can be removed. Each iteration (k) creates a model trained on n-k features having an error rate of e(k). Following this, you must select the maximum bearable error rate to define the smallest number of features needed to reach that classification performance with the given ML algorithm.

Forward Feature Construction

The forward feature construction is the opposite of the backward feature elimination method. In the forward feature construction method, you begin with one feature and continue to progress by adding one feature at a time (this is the variable that results in the greatest boost in performance).

Pros and Cons of Backward and forward feature

Both forward feature construction and backward feature elimination are time and computation-intensive. These methods are best suited for datasets that already have a low number of input columns.

We can assign weight to each score as another dataframe. This can be simply used for calculating weighted score.

(b) Suppose you have a dataframe df with columns restaurant_id, score_1, score_2, …, score_N. Write a Python function to output the customer scores using one of the dimension reduction techniques mentioned above. The output result should be a dataframe with columns restaurant_id and score. (**I have used PCA column name instead of score**)

```
def findComponents(dataframe=None):
    result = dataframe[['restaurant_id']]
    from sklearn.decomposition import PCA
    pca = PCA(n_components=1)
    principalComponents = pca.fit_transform(df.drop(columns=['restaurant_id']))
    columns = []
    principalDf = pd.DataFrame(data = principalComponents
             , columns = ['Score'])
    result[principalDf.columns] = principalDf.values
    return result
```

```
findComponents(df)
```

	restaurant_id	Score
0	ID_1	-0.199955
1	ID_2	0.502128
2	ID_3	-0.302173

Example DataFrame

```
import pandas as pd
import warnings
warnings.filterwarnings('ignore')

data = [["ID_1", 0.4, 0.2, 0.3],
        ["ID_2", 0.9, 0.5, 0.7],
        ["ID_3", 0.3, 0.3, 0.2]]

df = pd.DataFrame(data=data, columns=['restaurant_id', "Score1", "Score2", "Score3"])

df
```

	restaurant_id	Score1	Score2	Score3
0	ID_1	0.4	0.2	0.3
1	ID_2	0.9	0.5	0.7
2	ID_3	0.3	0.3	0.2

```
principalDf
```

	principal component 1	principal component 2
0	-0.199955	0.074657
1	0.502128	-0.009488
2	-0.302173	-0.065169

Generic Function for many components

Function

```
def findComponents(dataframe=None, component_you_want=1):
    result = dataframe[['restaurant_id']]
    from sklearn.decomposition import PCA
    pca = PCA(n_components=component_you_want)
    principalComponents = pca.fit_transform(df.drop(columns=['restaurant_id']))
    columns = []
    for i in range(0, component_you_want):
        columns.append("Score_"+str(i))
    principalDf = pd.DataFrame(data = principalComponents
             , columns = columns)
    result[principalDf.columns] = principalDf.values
    return result
```

```
findComponents(df)
```

	restaurant_id	Score_0
0	ID_1	-0.199955
1	ID_2	0.502128
2	ID_3	-0.302173

```
print("If you want more components")
```

```
If you want more components
```

```
findComponents(df, component_you_want=2)
```

	restaurant_id	Score_0	Score_1
0	ID_1	-0.199955	0.074657
1	ID_2	0.502128	-0.009488
2	ID_3	-0.302173	-0.065169