Python för Hantering av SQL Queries

Matthew H. Motallebipour

January 26, 2024

1 Teoretiska frågor

1.1 Relationsdatabas

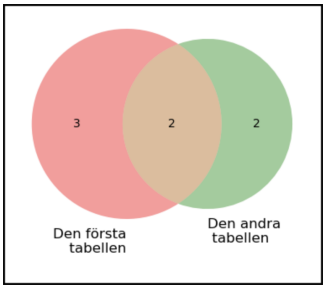
En relationsdatabas är en samling av tabeller som är relaterade till varandra med hjälp av ”nyckelegenskaper” som särskiljer raderna i varje tabell. Tanken med att ha relationsdatabaser är att undvika överflödiga rader som bara har en viss egenskap gemensamt.

1.2 CRUD

CRUD står för Create, Read, Update, och Delete, dvs skapa, läs, uppdatera och ta bort, vilka är de fundamentala funktionerna som kommer till användning just för att skapa och hantera en databas. Ett CRUD-flöde består därför av en blandning av desssa funktioner som kommer i en sekvens och möjliggör hanteringen.

1.3 LEFT JOIN och INNER JOIN

Man får se två databas tabeller som mängder, där join innebär att dessa kombineras för att få en ny tabell. LEFT JOIN innebär att alla rader från den första databasen tas med och motsvarande rader i den andra tabellen tillför information på samma rader som den första. Där den andra tabellen inte har någon ytterligare information kommer det att ersättas med NULL. INNER JOIN, å andra sidan, fyller den nya tabellen med de rader där både den första och den andra tabellen har information att tillföra. För de rader där den ena eller den andra inte innehåller någon information kommer ingen ny rad att skapas i den nya tabellen heller.



1.4 Indexering

Indexering är att skapa en metod att söka och hitta en viss rad i en tabell utan att nödvändigtvis behöva iterera igenom varenda rad i den tabellen.

1.5 Vy

En vy är samma sak som en tabell och kan användas och hanteras på samma sätt. Det finns dock en väsentlig skillnaden; att en vy skapas i ramminnet och är tillgängligt att använda så länge databasen är aktiv och servern/ datorn är påslagen.

1.6 Lagrad procedur

Lagrad procedur, som namnet föreslår, är en samling av SQL-kommandon som redan har kompilerats och är redo för körning. Detta för att spara tid och minne, undvika redundans i koden och att ge möjligheten att tillhandahålla information till ”extern” användare utan att denne ska ha direkt tillgång till databasen och källkoden.

2 Praktiska frågor

2.1 Beskrivning av databasen

Databasen AdventureWork är en fiktiv databas för att uppvisa Microsoft's möjligheter att hantera databaser. Den består av 5 scheman personal, kunder, tillverkning, försäljning, och återförsäljare, vilka i sig innehåller flera tabeller var. Originalen innehåller också redan flera vyer, vilket gör hopslagning (join) av tabellerna ännu enklare.

Det finns data om 296 anställda varav 290 jobbar kvar, 13 jobbkandidater, 181 försäljningsregioner, 1764 produkter, 6 cykelmodeller, 104 försäljningsställe med 156 kontaktpersoner, 19972 kunder och 17 säljare.

2.1.1 Installation av programvara

```
!pip install sqlalchemy
!pip install pyodbc
!pip install --upgrade pyodbc

from sqlalchemy import create_engine, MetaData, Table, inspect
import pandas as pd
# to show the whole width and length of the tables
pd.set_option('display.max_columns', None)
pd.reset_option('display.max_rows')
pd.reset_option('display.max_columns')
```

2.1.2 Databas uppkoppling och åtkomst

Detta har redan diskuterats och behöver inte redovisas. Dock är bra att påpeka att vår *engine* har vi använt parametervärdena 'mssql', 'DESKTOP-PJ0B80O', 'AdventureWorks2022', och integrated_security=True.

2.1.3 Förberedelse av scheman and tabeller

Detta är också redan diskuterat och behöver inte tas upp här.

2.1.4 Databas förfrågan (Queries)

Vi försöker först med att ta en snabb titt på alla tabeller i databasen.

```
# Find the name of all the tables in the database within each schema
# Create a dictionary to store queries and results
all_queries_dict = {}
for sch in schemas:
    for table in inspector.get_table_names(schema=sch):
        if sch != 'dbo':
            # Get the names of all the tables in each schema and
            # generate a string with all the queries
            col_names = ', '.join(col['name'] for col in
                                  inspector.get_columns(table_name=table, schema=sch))
            full_table_name = f"{sch}_{table}"
            query = f"SELECT {col_names} FROM {sch}.{table}"
            all_queries_dict[full_table_name] = query
# Put the queries in action and print the results one by one
for full_table_name, query in all_queries_dict.items():
    print("QUERY\n------\n", query, "\n")
    try:
        # Read the table and print out the names of its columns
        # Store the DataFrame in a dictionary
        df_table = pd.read_sql(sql=query, con=connection)
        all_queries_dict[full_table_name] = df_table
        (schema_name, table_name) = full_table_name.split('_')
        print("*****************************************************************")
        print(f"Schema: {schema_name}\nTable: {table_name}\n")
        # Print 5 first rows of the table at hand and some of its basic statistics
        print(df_table.head(5))
        print("*****************************************************************")
```

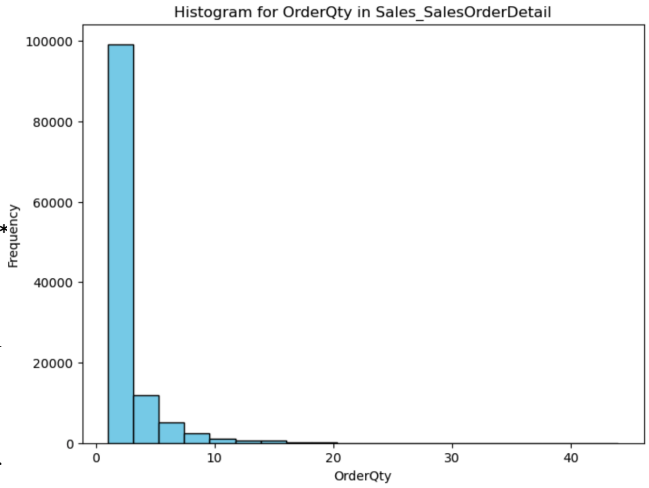
```
            print(df_table.describe())
            print("**********************************************************************")
        except Exception as e:
            print(f"Error executing query: {e}")
        print("\n"*3)
```

Resultatet blir en beskrivning av alla tabeller i databasen, vad vi har använt för *Query*, de första fem raderna i tabellen och lite statistik över just den tabellen.

```
QUERY
------
 SELECT DepartmentID, Name, GroupName, ModifiedDate FROM HumanResources.Department

**********************************************************************
Schema: HumanResources
Table: Department

   DepartmentID          Name                 GroupName ModifiedDate
0             1   Engineering  Research and Development   2008-04-30
1             2   Tool Design  Research and Development   2008-04-30
2             3         Sales        Sales and Marketing   2008-04-30
3             4     Marketing        Sales and Marketing   2008-04-30
4             5    Purchasing      Inventory Management   2008-04-30
**********************************************************************
       DepartmentID          ModifiedDate
count     16.000000                    16
mean       8.500000   2008-04-30 00:00:00
min        1.000000   2008-04-30 00:00:00
25%        4.750000   2008-04-30 00:00:00
50%        8.500000   2008-04-30 00:00:00
75%       12.250000   2008-04-30 00:00:00
max       16.000000   2008-04-30 00:00:00
std        4.760952                   NaN
**********************************************
```



Vi noterar att schemas Person och Sales innehåller mycket information som är relaterade till företagets kunder. Detta väljer vi som ämne för vår analys och framställer samma tabeller ännu en gång för att lättare undersöka sambanden mellan tabeller som innehåller all information om kunderna. Koden skiljer sig inte annat än att vi bara väljer de tabeller som tillhör schemas Persona och Sales.

```
if sch == 'Person' or sch == 'Sales':
```

Sedan att vi får en hel del tabeller som egentligen inte kommer till någon användning är något vi kan försumma när det kommer till automatisk utvinning of information, trots att det minskar överskådligheten. Detta eftersom vi ritar grafer för alla kolumner i alla tabeller, istället för att välja ut ”rätt” kolumn, vilket skulle kräva mer analytisk kodskrivning genom t ex regex-uttryck. Det var först här som vi noterade att det finns många egenskaper hos kunderna som skulle kunna undersökas närmare för att se ett mönster.

Vi tittar också på views som redan finns presenterade i den originella databasen och speciellt där kunderna är uppradade. Den största skillnaden mot tidigare koder är

```
for sch in schemas:
    for view in inspector.get_view_names(schema=sch):
```

Vi bestämmer oss således för att jämföra köpvanan hos de båda könen och undersöka om vi kan hitta olika mönster i hur män och kvinnor investerar för sin hälsa/ ekonomi, alternativt för en miljövänligare transportmedel. Från de senaste utskrifterna kan vi konstatera att för en rättvis och noggrann jämförelse mellan könen behöver vi tabeller med många kunder i. Därför väljer vi *vIndividualCustomer* och *vPersonDemographics* från schema *Person*. Vi gör en *JOIN* mellan de två tabellerna och får

```
# SQL query for joining the two tables
joint_table = f'''
                SELECT *
                FROM Sales.vPersonDemographics AS VPD
```

```
            JOIN Sales.vIndividualCustomer AS VIC
            ON VIC.BusinessEntityID = VPD.BusinessEntityID;
        '''
# Save the result in a dataframe for further investigation
joint_df = pd.read_sql(sql=joint_table, con=connection)
print(joint_df.head(5))
```

Vi gör en klassificering av data i kolumnen för årsinkomst och får att:

```
# Count the number of incomes on each income level
income_counts = joint_df['YearlyIncome'].value_counts()
print(income_counts)
```

För att kunna använda denna kolumn i våra statistiska beräkningar ersätter vi varje intervall med dess max-gräns, dvs:

0-25000 with **25,000**
25001-50000 with **50,000**
50001-75000 with **75,000**
75001-100000 with **100,000**
greater than 100000 with **500,000**

för att göra talen i kolumnerna konvertibla till heltal inför nästa steg, som är att skapa en korrelationsmatris. Vi kommer inte att använda själva talen i våra beräkningar så det finns ingen risk att våra beräkningar genererar felaktiga resultat. Med andra ord använder vi årsinkomsten endast som en ordinal, kategorisk variabel.

```
# Make a regex to differentiate between the 'nnnnn-nnnnn' and 'greater then 100000'
pattern = r'-(\d+)| (\d+)'
# Apply the regex pattern on all the cells, remove NaN and convert the result to int
matches = joint_df['YearlyIncome'].str.extract(pattern).fillna(0).astype(int)
# Use the apply function to find the maximum value for each row and save it in a new column
joint_df['YearlyIncomex'] = matches.apply(lambda row: max(row[0], row[1]*5), axis=1)
print(joint_df)
```

2.1.5 Övergång från kategorisk data till ordinal data

Bortsett från födelsedatum för kunderna är för andra kolumnerna varken omvandlingen eller kategoriseringen lika komplicerade att utföra

```
from sklearn.preprocessing import LabelEncoder
# Initialize the LabelEncoder
label_encoder = LabelEncoder()
```

Här använder vi *LabelEncoder* till att beräkna ålders för företagets kunder

```
encoded_dates = label_encoder.fit_transform(joint_df['BirthDate'])
# Convert the encoded dates to strings
date_strings = joint_df['BirthDate'].astype(str)
# Extract the first four characters (digits) from each date
year = date_strings.str[:4]
year_from_date = pd.to_numeric(year)
# Calculate the interval between the year from date and the current year
# Choosing 1998 as the current year gives a time span of 18 through 88,
# which sounds more reasonable than any other end date
current_year = 1998
joint_df['Agex'] = current_year - year_from_date
joint_df['Agex']
```

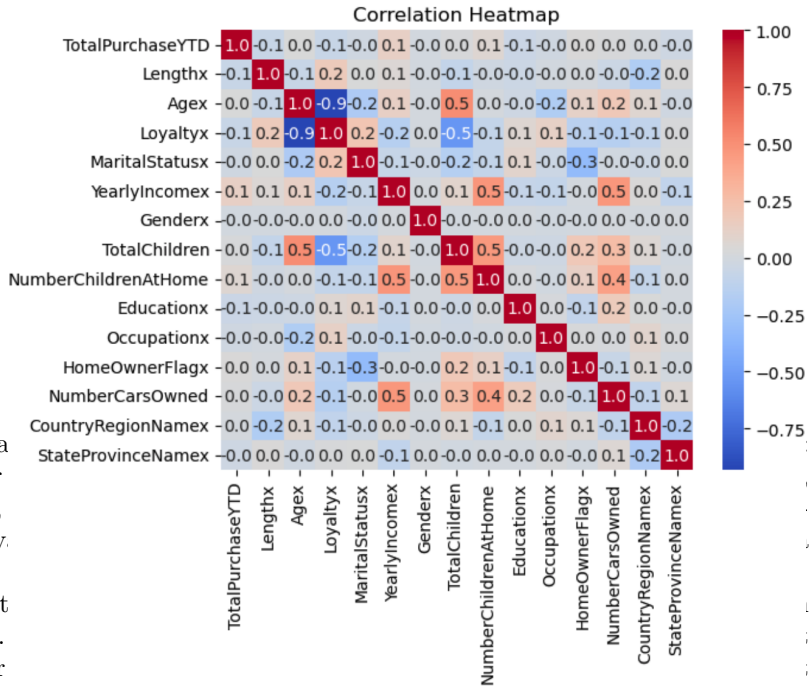
```
YearlyIncome
25001-50000           5713
50001-75000           5483
0-25000               2922
75001-100000          2762
greater than 100000   1628
Name: count, dtype: int64
```

För de andra kolumnerna är det som sagt inte like dramatiskt, utan alla följer samma mönster som nedan

```
joint_df['MaritalStatusx'] = label_encoder.fit_transform(joint_df['MaritalStatus'])
# Display the mapping of original labels to encoded integers
label_mapping = dict(zip(label_encoder.classes_, label_encoder.transform(label_encoder.classes_)))
print("Label Mapping:", label_mapping)
```

3 Analys och slutsatser

3.1 Försök: hitta samband mellan Total Purchase YTD och andra faktorer



Innan vi går vidare med att jämföra könen, gör vi en analys för att se om det eventuellt finns något samband mellan det belopp man spenderat under det nuvarande året och något av de relevanta egenskaperna hos företagets kunder. Vi kan använda de nya kolumnerna till att beräkna korrelationen mellan alla dessa variabler, med hopp om att hitta variabler som är mer korrelerade med andra, speciellt de som är kopplade till kundernas kön.

Från tabellen kan vi endast utläsa Agex-Loyaltyx (0.9; detta är framför NumberChildren-TotalChildren (0.5), 0.5), Inga av dessa är relevanta för v någon annanstans.

Den enda finansiellt relevanta slut som i sin tur är korrelerad med ålder. kan förstås i och med att de äldre har spendera på egna intressen.

I nästa steg tittar vi på inkomster och försöker hitta ett samband mellan könen och hur mycket de tjänar och spenderar.

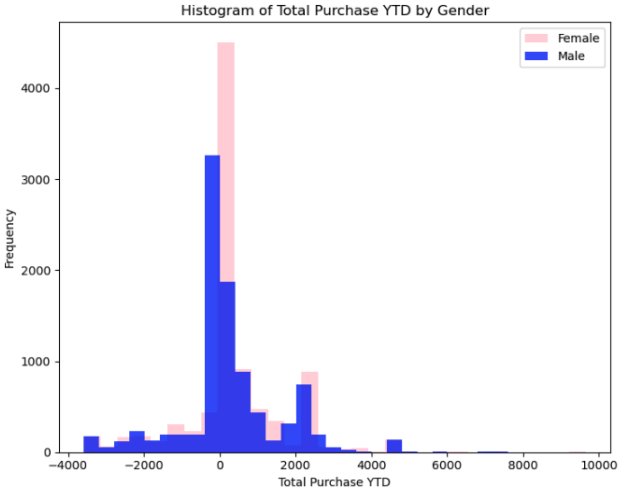
```
import seaborn as sns
# Create a correlation matrix to present the relation between all
correlation_matrix = joint_df[['TotalPurchaseYTD', 'Lengthx', 'Agex', ...]].corr()
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt=".1f")
plt.title('Correlation Heatmap')
plt.show()
```

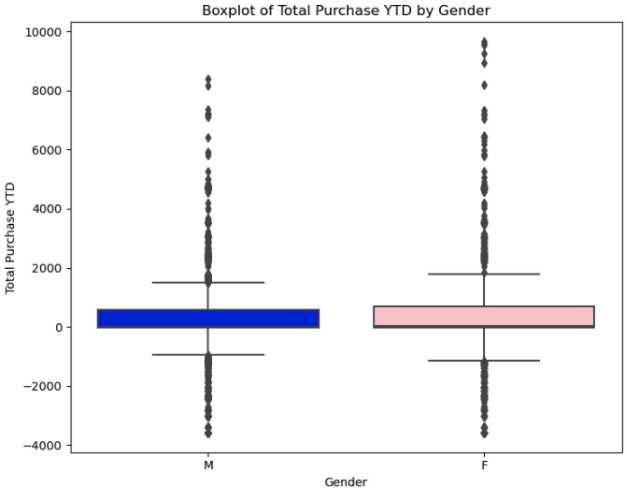
3.2 Försök: hitta skillnad i inkomst och köpvana hos de ”mest entusiastiska” männen och kvinnorna

```
# Count the number of incomde levels
expenditure_counts = joint_df['TotalPurchaseYTD'].value_counts()
print(expenditure_counts)

YIx_female = joint_df.loc[joint_df['Gender'] == 'F', 'TotalPurchaseYTD']
YIx_male = joint_df.loc[joint_df['Gender'] == 'M', 'TotalPurchaseYTD']
# Plot histograms
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
plt.hist(YIx_female, bins=30, alpha=0.7, label='Female', color='pink')
plt.hist(YIx_male, bins=30, alpha=0.7, label='Male', color='blue')
plt.xlabel('Total Purchase YTD')
plt.ylabel('Frequency')
plt.title('Histogram of Total Purchase YTD by Gender')
plt.legend()
# Plot boxplots
plt.subplot(1, 2, 2)
sns.boxplot(x='Gender', y='TotalPurchaseYTD', data=joint_df, palette={'F': 'pink', 'M': 'blue'})
plt.xlabel('Gender')
plt.ylabel('Total Purchase YTD')
plt.title('Boxplot of Total Purchase YTD by Gender')
```

```
plt.tight_layout()
plt.show()
```





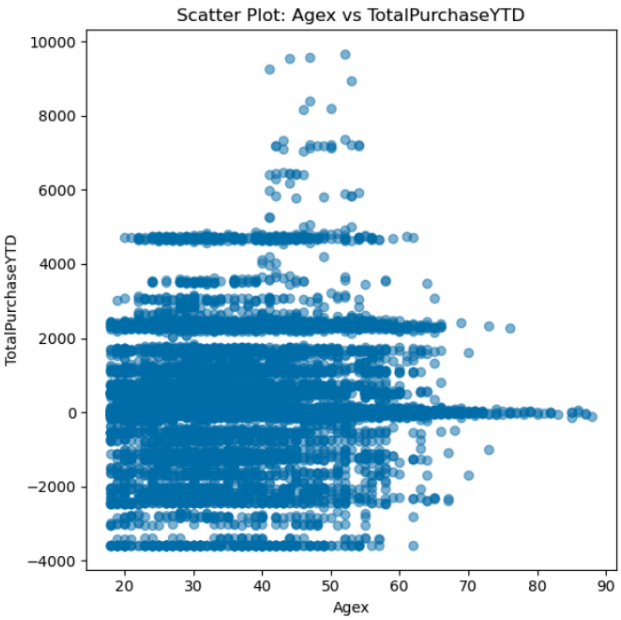
I histogrammen ser vi en viss skiftning i kvinnornas totala köp i det föreliggande året jämfört med männens. Även i vår Boxplot ser vi denna skiftning, med extremvärden som ligger längre bort från ”boxen” jämfört med schemat för männen. Därför finns det en chans att vi åtminstone kan hitta en 95 procentig signifikant skillnad.

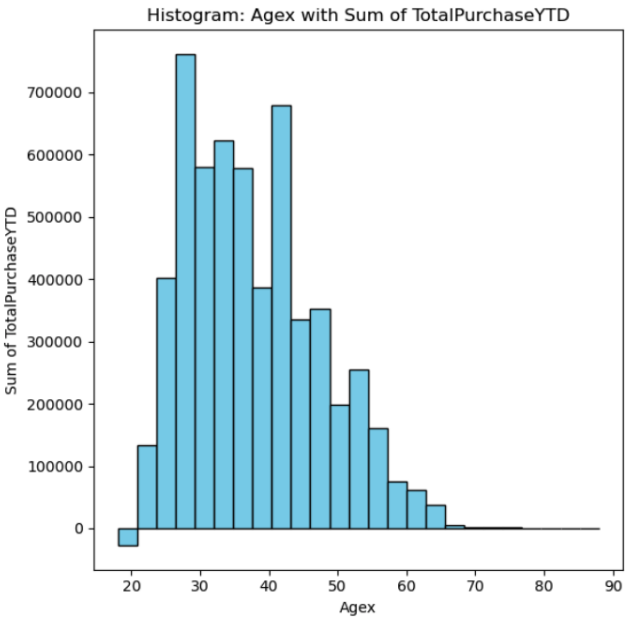
```
T-statistic: 0.7885667920652966
P-value: 0.43037536312561386
Confidence Interval: (-23.324626429997387, 54.72475825253415)
```

Detta betyder att det inte finns någon signifikant skillnad i generella drag mellan hur mycket män och kvinnor spenderar på köp av cykel och tillbehör.

3.3 Försök: skillnad mellan könen i den högre ”spenderarklassen”

Om vi tittar på spridningsdiagrammet för hela populationen, märker vi att de högsta beloppen spenderas av de som är mellan 40 och 55 år gamla.





3.4 Försök: hitta skillnad i köpvana mellan kvinnor och män i de olika inkomstklasserna

Write about a better method, which is the regression and multiple regression.

4 Conclusion