In [ ]:
```
#Importing packages
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import holidays
import gc
import dill
from datetime import date
from pandas.tseries.holiday import USFederalHolidayCalendar as calendar
#Setting large figure size for Seaborn
sns.set(rc={'figure.figsize':(11.7,8.27),"font.size":20,"axes.titlesize":20,"axes.

from sklearn.base import clone
import numpy.random as rng
from sklearn.utils import check_random_state
from sklearn.decomposition import PCA
from functions import *
from sklearn.model_selection import KFold, ParameterGrid
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score, adjusted_rand_score

import plotly.express as px
import plotly.graph_objects as go

#Importing MaxNLocator to restrict matplotlib scale to integers
from matplotlib.ticker import MaxNLocator
```

In [ ]:
```
data_dir = 'data'
def load_oracle_files(dir: str):
    orders = pd.read_excel(dir+'/Orders.xlsx')
    customers = pd.read_excel(dir+'/Customers.xlsx')
    return orders, customers

orders, customers = load_oracle_files(data_dir)
```

1 Data Analysis

1.1 Exploration

In [ ]:
```
print(orders.shape)
orders.head()
```

```
(131706, 8)
```

Out[ ]:

	id	Date	Customer_ID	Transaction_ID	SKU_Category	SKU	Quantity	Sales_Amount
0	1	2021-01-02	2547	1	X52	0EM7L	1.0	3.13
1	2	2021-01-02	822	2	2ML	68BRQ	1.0	5.46
2	3	2021-01-02	3686	3	0H2	CZUZX	1.0	6.35
3	4	2021-01-02	3719	4	0H2	549KK	1.0	5.59
4	5	2021-01-02	9200	5	0H2	K8EHH	1.0	6.88

In [ ]:
```
print(customers.shape)
customers.head()
```

```
(22625, 4)
```

Out[ ]:

	Customer_ID	GENDER	AGE	GEOGRAPHY
0	2547	F	48	Spain
1	822	F	39	Germany
2	3686	F	56	Greece
3	3719	F	49	Spain
4	9200	M	18	Italy

In [ ]:
```
orders.describe()
```

Out[ ]:

	id	Customer_ID	Transaction_ID	Quantity	Sales_Amount
count	131706.000000	131706.000000	131706.000000	131706.000000	131706.000000
mean	65853.500000	12386.450367	32389.604187	1.485318	11.981524
std	38020.391614	6086.447552	18709.901238	3.872666	19.359699
min	1.000000	1.000000	1.000000	0.010000	0.020000
25%	32927.250000	7349.000000	16134.000000	1.000000	4.230000
50%	65853.500000	13496.000000	32620.000000	1.000000	6.920000
75%	98779.750000	17306.000000	48548.000000	1.000000	12.330000
max	131706.000000	22625.000000	64682.000000	400.000000	707.730000

In [ ]:
```
orders[orders.Quantity == 0]
```

Out[ ]:

	id	Date	Customer_ID	Transaction_ID	SKU_Category	SKU	Quantity	Sales_Amount

In [ ]:
```
orders[orders.Quantity > 100].head(2)
```

Out[ ]:

	id	Date	Customer_ID	Transaction_ID	SKU_Category	SKU	Quantity	Sales_Amoun
17084	17085	2021-02-19	6869	8186	LGI	VWLV9	176.0	74.8
64680	64681	2021-06-29	17025	32069	LGI	VWLV9	176.0	74.8

In [ ]:
```
customers.GEOGRAPHY.value_counts()
```

Out[ ]:
```
Germany        7970
Italy          4480
France         2644
Greece         2269
UK             2264
Spain          1651
Netherlands    1347
Name: GEOGRAPHY, dtype: int64
```

In [ ]:
```
customers.GENDER.value_counts()
```

Out[ ]:
```
F    11328
M    11297
Name: GENDER, dtype: int64
```

1.2 Data Cleaning

In [ ]:
```
def clean_data(df: pd.DataFrame, quantity_col: str = 'Quantity'):
    df = df.loc[df[quantity_col] % 1 == 0]
    df[quantity_col] = df[quantity_col].apply(lambda x: int(x))
    return df.set_index('id')
orders = clean_data(orders)
orders.head(2)
```

```
/tmp/ipykernel_916/3184141252.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  df[quantity_col] = df[quantity_col].apply(lambda x: int(x))
```

Out[ ]:

id	Date	Customer_ID	Transaction_ID	SKU_Category	SKU	Quantity	Sales_Amount
1	2021-01-02	2547	1	X52	0EM7L	1	3.13
2	2021-01-02	822	2	2ML	68BRQ	1	5.46

2 Statistical Analysis

In [ ]:
```
item_df = orders.groupby("Customer_ID").agg(first_date=('Date','min'), last_date=(
                                            most_cat = ('SKU_Category', lambda x
item_df.head(2)
```

Out[ ]:

Customer_ID	first_date	last_date	diff_items_bought	most_cat	most_item_SKU
1	2021-01-22	2021-01-22	2	0H2	6OUVC
2	2021-03-24	2021-06-19	2	TVL	2SLS0

In [ ]:
```
item_df.most_cat.value_counts(normalize=True)[0:10].sum()

# Too many categories to efficiently use this data
```

Out[ ]:
```
0.4176481005932879
```

In [ ]:
```
standard_functions = ['min', 'max', 'sum', 'mean', 'std', 'median']

aggregations = {
    'Customer_ID': ['count'],
    'order_value' : standard_functions,
    'diff_items' : standard_functions,
    'date': ['min', 'max'],
    'total_items': standard_functions
}

def aggregate_transactions(df: pd.DataFrame, transact_col: str):
    results = df.groupby(transact_col).agg(Customer_ID = ('Customer_ID', 'max'), di
                                           order_value = ('Sales_Amount','su
    return results.dropna()

def aggregate_results(df: pd.DataFrame, index_col: str, aggregations: dict):
    results = df.groupby(index_col).agg(aggregations)

    #Renaming columns to remove multi indexing
    results.columns = pd.Index([e[0] + '_' + e[1] for e in results.columns])

    results.rename(columns={'Customer_ID_count': 'number_orders'}, inplace=True)

    # Replacing null standard deviation due to single order clients with 0
    results = results.fillna(0)

    return results

def compile_customer_info(cust_df: pd.DataFrame, cust_transact: pd.DataFrame, item_
    cust_df = cust_df.set_index('Customer_ID')

    # Merge our customers data with the aggregated customer transactions data
    results = pd.merge(cust_df, cust_transact, left_index=True, right_index=True, h

    results = pd.merge(results, item_df[['diff_items_bought']], left_index=True, ri

    # Dropping the sum of different items which makes no sense since it is a double
    results.drop(columns='diff_items_sum', inplace=True)

    # Bin the age column
    results["age_bin"] = pd.cut(results["AGE"], bins = [18 , 25, 30, 35, 40, 45, 5(
                                labels=["18-24", "25-29", "30-34", "35-39", "40-44

    results.rename(columns={'order_value_sum': 'total_revenue', 'total_items_sum':

    return results

def is_holiday(calendar, date) -> bool:
    return date in calendar
```

```
def define_holidays(df: pd.DataFrame, country_col: str="GEOGRAPHY") -> None :
    countries = df[country_col].unique()
    for c in countries:
        calendar = holidays.country_holidays(c)
        df.loc[df[country_col] == c, 'is_holiday'] = df.apply(lambda x: is_holiday
    return df

def compile_transaction_info(trans_df, customers_df):
    results = pd.merge(trans_df, customers_df, left_on='Customer_ID', right_index='
    results.drop(columns='Customer_ID_y', inplace=True)
    results.rename(columns={'Customer_ID_x': 'Customer_ID'}, inplace= True)

    # Adding month field and is_holiday boolean field
    results["month"] = results["date"].dt.month
    results["is_holiday"] = False

    # Dropping transactions with no customer records
    results.dropna(inplace=True)

    results = define_holidays(results)
    return results
```

In [ ]:
```
transactions = aggregate_transactions(orders, transact_col='Transaction_ID')

customer_transact = aggregate_results(transactions, index_col='Customer_ID', aggreg

customer_transact.head(2)
```

Out[ ]:

Customer_ID	number_orders	order_value_min	order_value_max	order_value_sum	order_value_mea
1	1	16.29	16.29	16.29	16.29
2	2	7.77	15.00	22.77	11.38

2 rows × 21 columns

In [ ]:
```
customers_df = compile_customer_info(customers, customer_transact, item_df)
customers_df.head(2)
```

Out[ ]:

Customer_ID	GENDER	AGE	GEOGRAPHY	number_orders	order_value_min	order_value_max	tota
2547	F	48	Spain	7	3.13	39.62	
822	F	39	Germany	3	5.46	8.59	

2 rows × 25 columns

In [ ]:
```
transactions = compile_transaction_info(transactions, customers)
transactions.head(2)
```

Out[ ]:

Transaction_ID	Customer_ID	diff_items	total_items	order_value	date	GENDER	AGE	GEOGRAP
1	2547	1	1	3.13	2021-01-02	F	49.0	Sp
2	822	1	1	5.46	2021-01-02	M	26.0	I

In [ ]:
```
customers_df.describe()
```

Out[ ]:

	AGE	number_orders	order_value_min	order_value_max	total_revenue	order_value
count	22586.000000	22586.000000	22586.000000	22586.000000	22586.000000	22586.
mean	37.917692	2.856017	14.252760	34.496467	69.591381	21.
std	12.228208	3.995979	24.107055	50.808639	152.002250	28.
min	18.000000	1.000000	0.030000	0.140000	0.140000	0.
25%	29.000000	1.000000	4.290000	8.870000	10.132500	7.
50%	38.000000	1.000000	7.500000	17.410000	23.800000	13.
75%	46.000000	3.000000	14.750000	37.087500	63.007500	24.
max	75.000000	99.000000	544.790000	707.730000	3949.550000	544.

In [ ]:
```
def generate_categorical_column_stats(df:pd.DataFrame, columns=list[str], summary_
    results = []

    for c in columns:
        c_cap = str(c).capitalize()
        for val in df[c].unique():
            val_cap = str(val).capitalize()
            for sum_col in summary_cols:
                sum_col_cap = str(sum_col).capitalize()
                subset = df[df[c] == val][sum_col]
                results.append({'column': c_cap, 'subset': val_cap, 'variable': su
                results.append({'column': c_cap, 'subset': val_cap, 'variable': su
                results.append({'column': c_cap, 'subset': val_cap, 'variable': su
                results.append({'column': c_cap, 'subset': val_cap, 'variable': su
    return results
```

In [ ]:
```
sum_cols = [ 'total_revenue','number_items_bought', 'diff_items_bought', 'number_o
customer_summary = generate_categorical_column_stats(customers_df, ['age_bin','GEO
geo_sum_cols = ['total_items', 'order_value']
geo_summary = generate_categorical_column_stats(transactions, ['month', 'is_holida
results = pd.DataFrame(customer_summary + geo_summary).sort_values(by=['column', '
results.to_csv("data/output_1.csv")
```

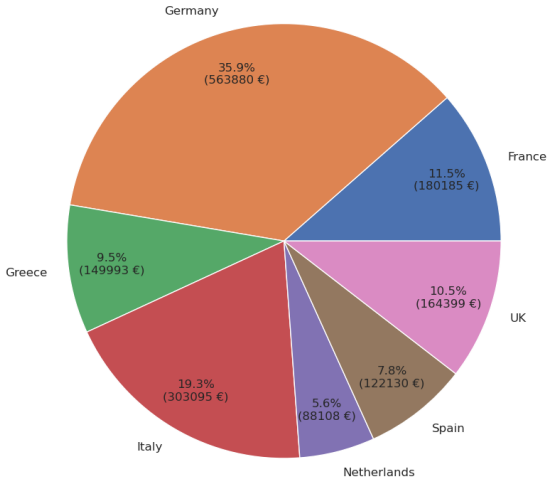
In [ ]:
```
def plot_statistical_data(df: pd.DataFrame, cat_cols: list[str], sum_cols: list[st
    for c in cat_cols:
        for sum_col in sum_cols:
            subset = df[[c,sum_col]]
            cat_name = str(c).replace('_', ' ').capitalize()
            sum_col_name = str(sum_col).replace('_', ' ').capitalize()
            # Sum
            subset.groupby(c).sum().reset_index().plot(x=c, y=sum_col, kind="bar",
```

```
            plt.title('Sum of {} by {}'.format(sum_col_name, cat_name))
            plt.xlabel(cat_name)
            plt.ylabel(sum_col_name)
            plt.show()

            # Mean
            subset.groupby(c).mean().reset_index().plot(x=c, y=sum_col, kind="area
            plt.title('Average {} by {}'.format(sum_col_name, cat_name))
            plt.xlabel(cat_name)
            plt.ylabel(sum_col_name)
            plt.show()

            # Median
            subset.groupby(c).median().reset_index().plot(x=c, y=sum_col, kind="ar
            plt.title('Median {} by {}'.format(sum_col_name, cat_name))
            plt.xlabel(cat_name)
            plt.ylabel(sum_col_name)
            plt.show()

            # Max
            subset.groupby(c).max().reset_index().plot(x=c, y=sum_col, kind="area"
            plt.title('Max {} by {}'.format(sum_col_name, cat_name))
            plt.xlabel(cat_name)
            plt.ylabel(sum_col_name)
            plt.show()

            # Min
            subset.groupby(c).min().reset_index().plot(x=c, y=sum_col, kind="area"
            plt.title('Min {} by {}'.format(sum_col_name, cat_name))
            plt.xlabel(cat_name)
            plt.ylabel(sum_col_name)
            plt.show()
```

In [ ]:

```
geo_summary = customers_df.groupby('GEOGRAPHY').sum().reset_index()
plt.pie(geo_summary['total_revenue'], labels=geo_summary['GEOGRAPHY'],
        pctdistance = 0.8,
        labeldistance = 1.1,
        autopct=lambda x: f"{x:.1f}%\n({(x/100)*sum(geo_summary['total_revenue']):
        textprops={"size": 12},
        radius = 1.2)
plt.legend(loc="best", bbox_to_anchor=(2.5,0), title="Country")
plt.show()
```

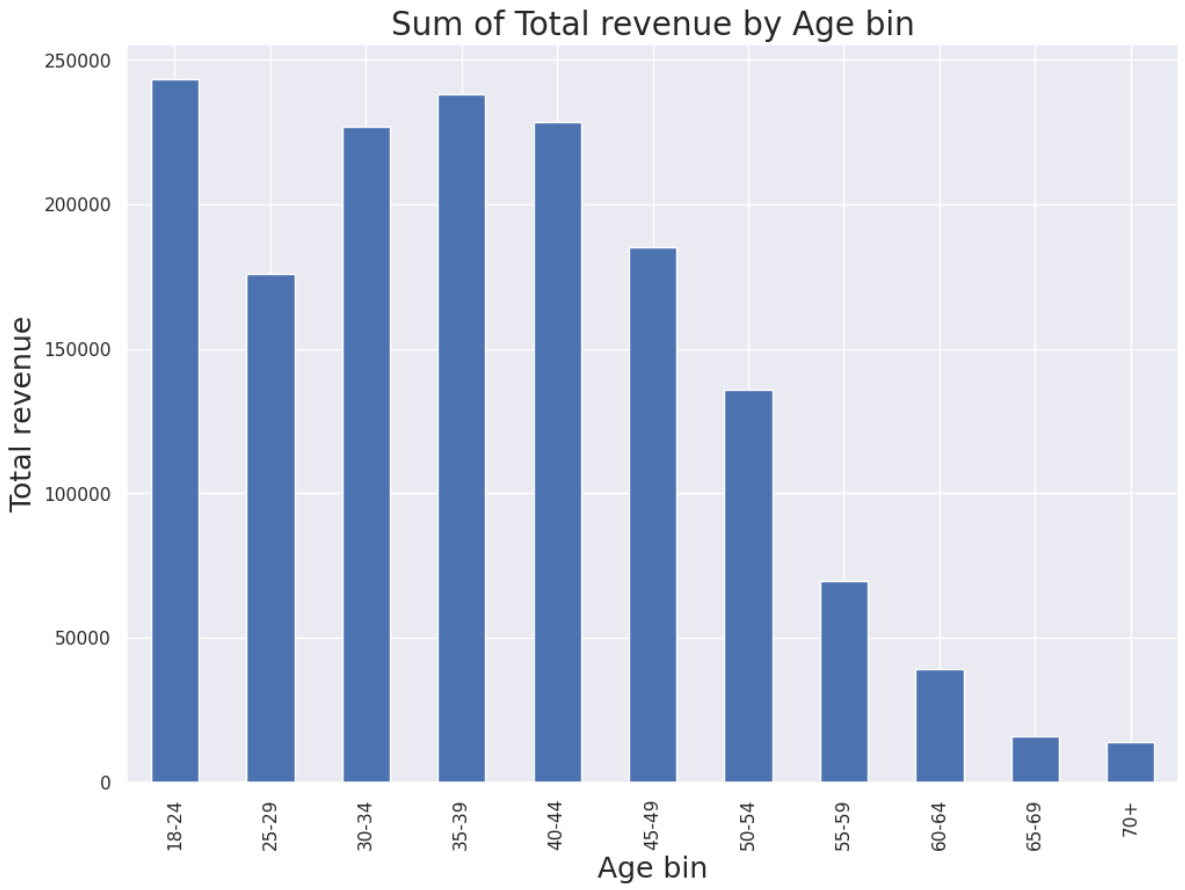
```
/tmp/ipykernel_1634/3955889209.py:1: FutureWarning: The default value of numeric_o
nly in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will
default to False. Either specify numeric_only or select only columns which should
be valid for the function.
  geo_summary = customers_df.groupby('GEOGRAPHY').sum().reset_index()
```

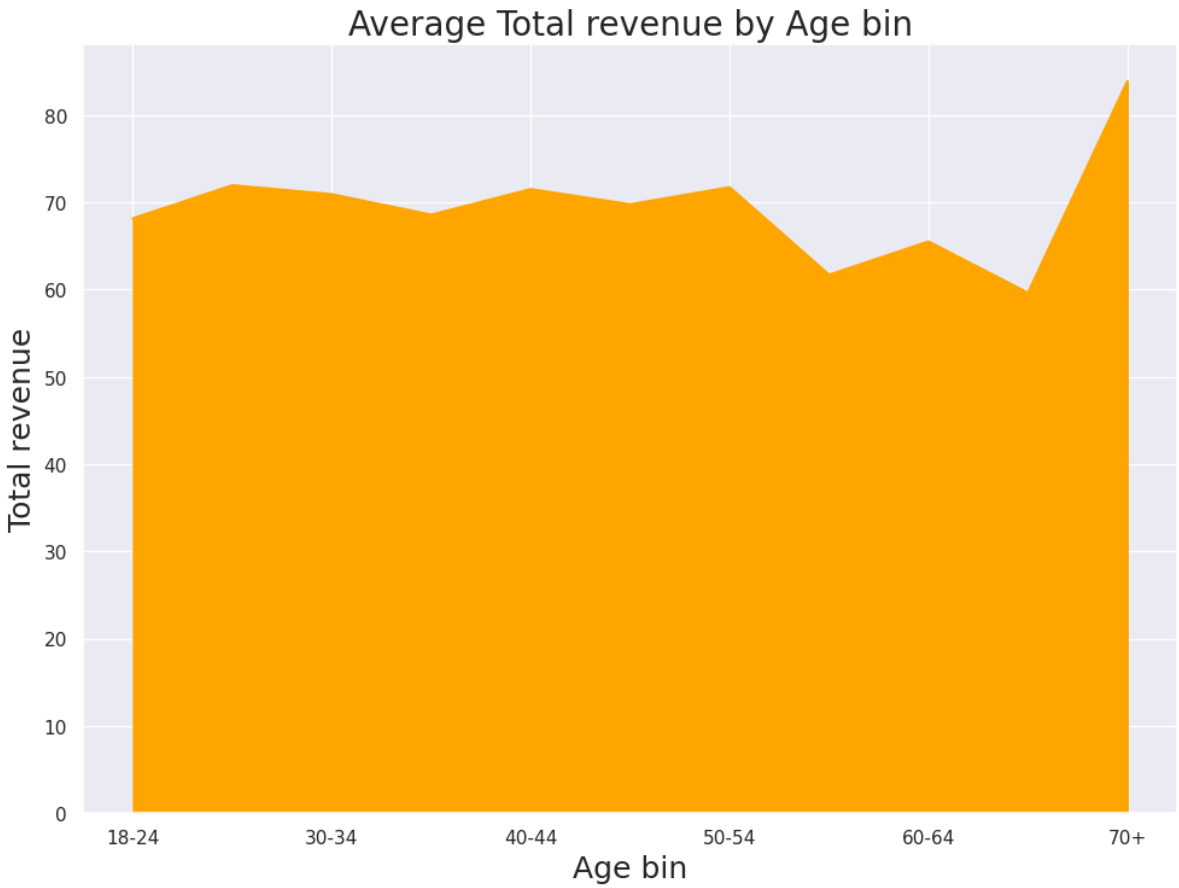


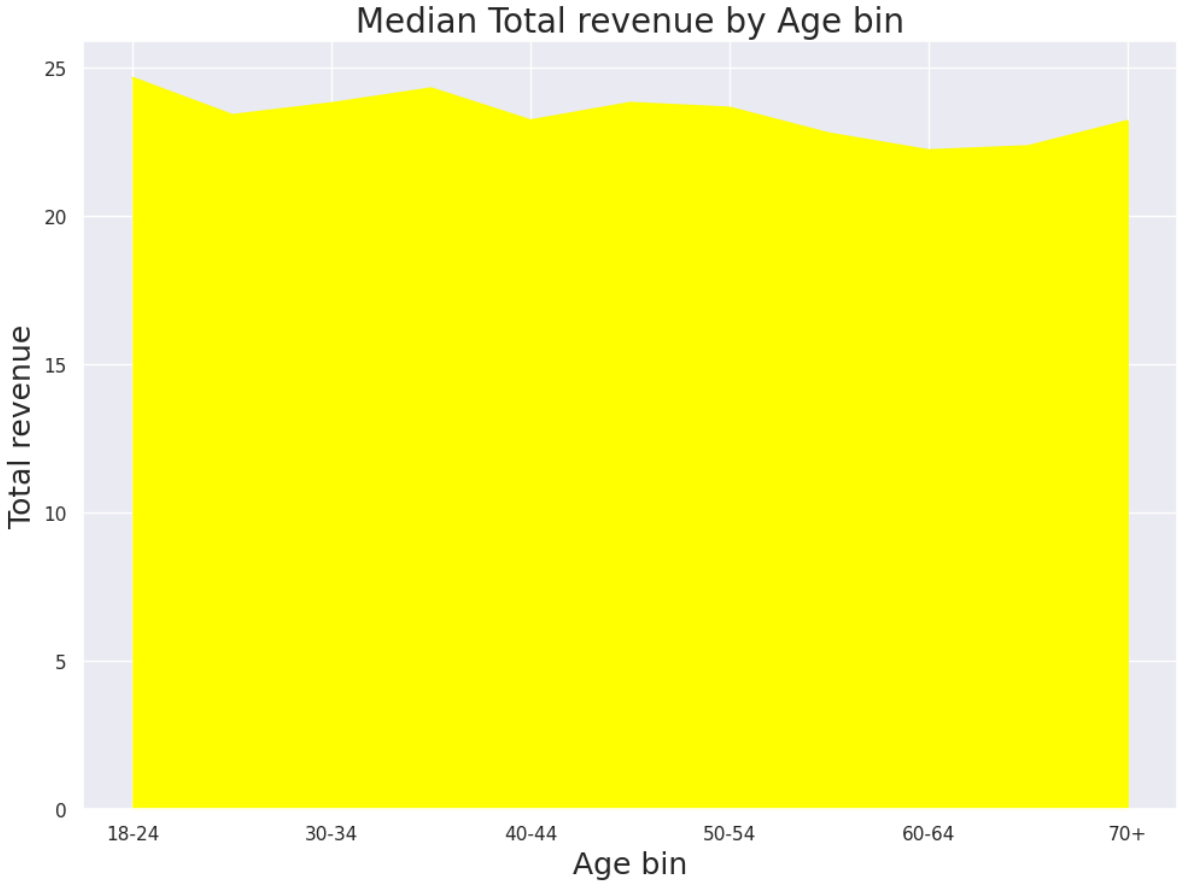
In [ ]:
```
customers_df.total_revenue.sum()
```

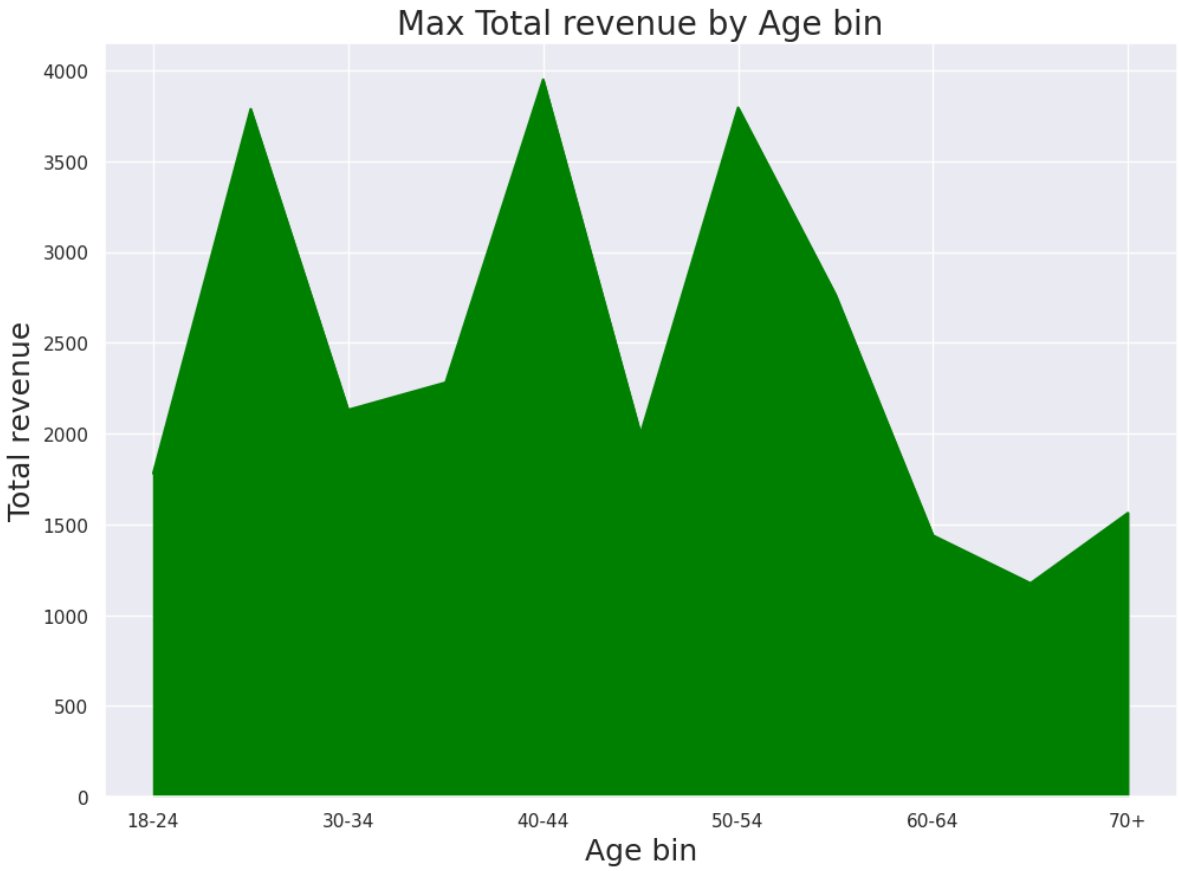
Out[ ]:
```
1571790.9299999997
```

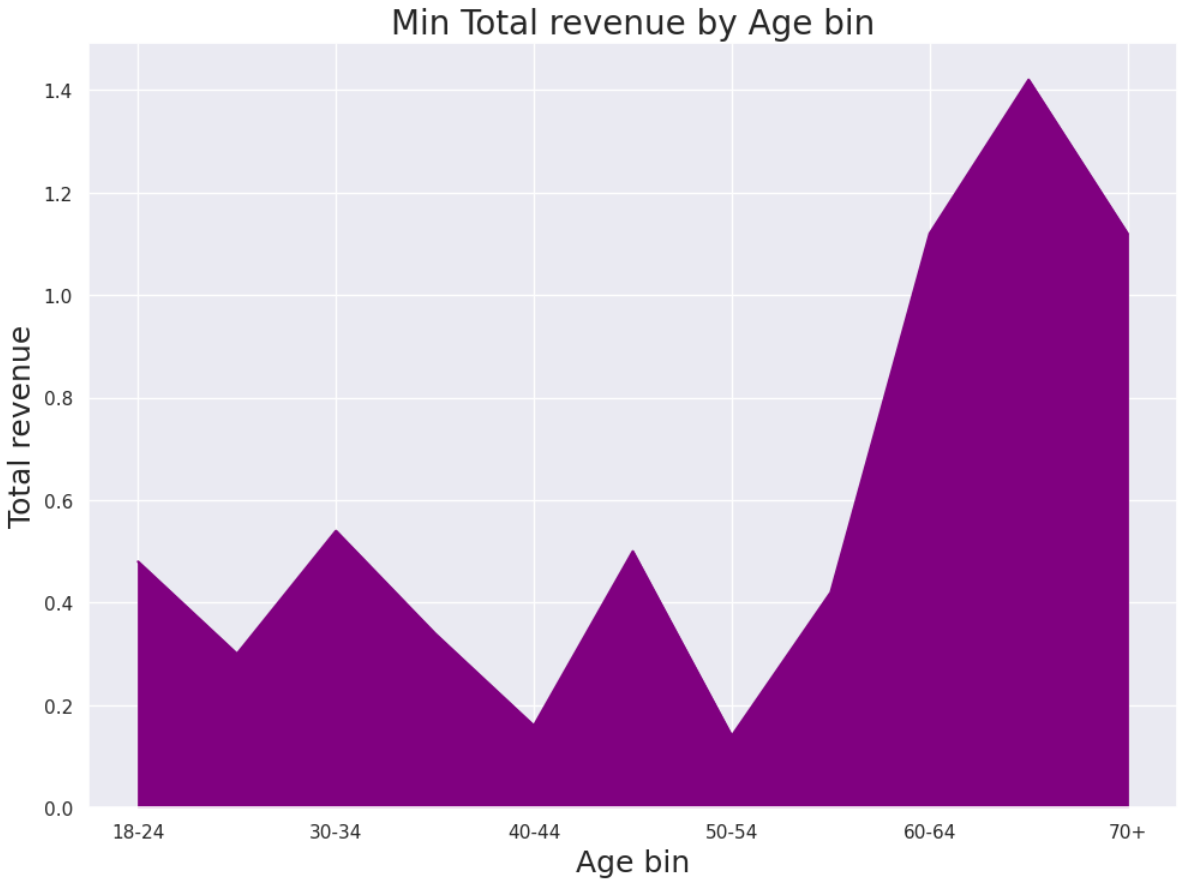
In [ ]:
```
plot_statistical_data(customers_df, ['age_bin','GEOGRAPHY', 'GENDER'], sum_cols)
```

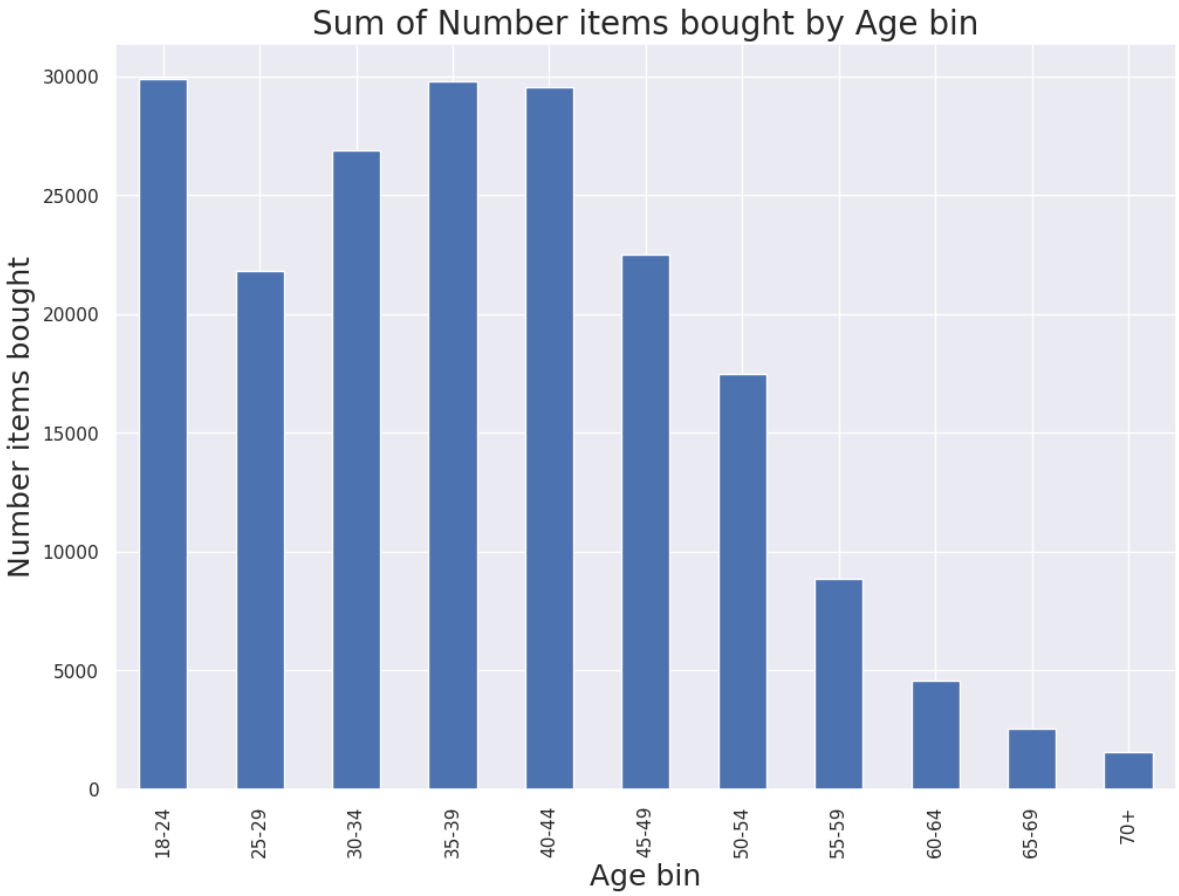


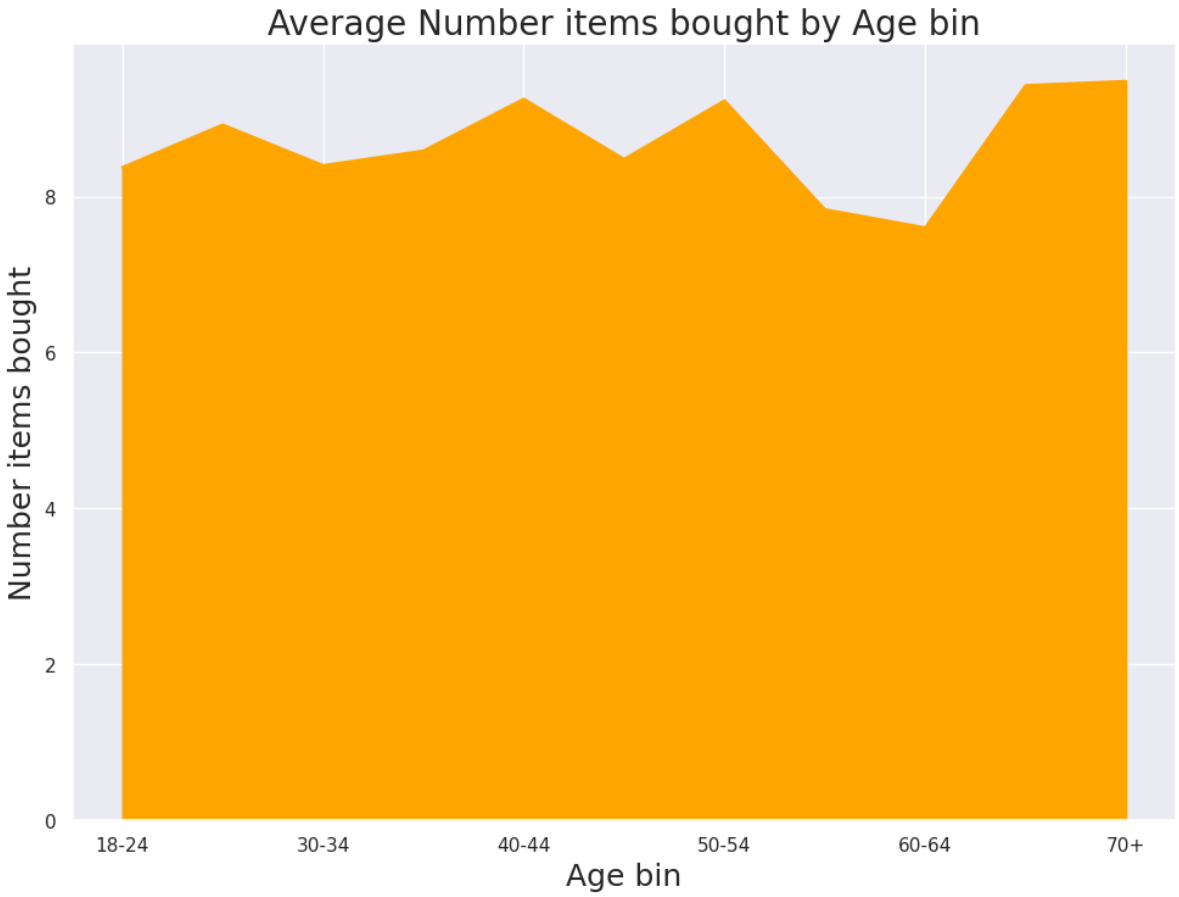

Average Total revenue by Age bin
Total revenue
0
10
20
30
40
50
60
70
80
18-24
30-34
40-44
50-54
60-64
70+
Age bin

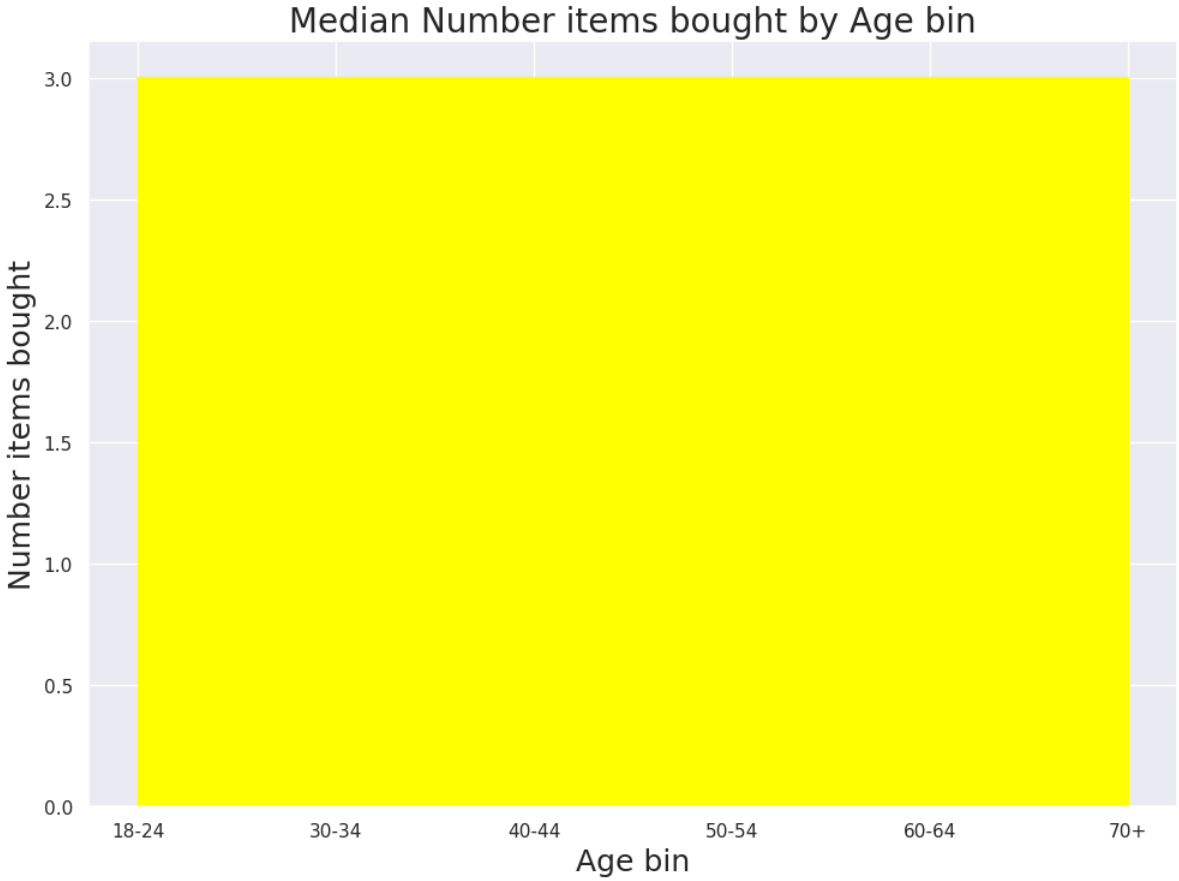

Median Total revenue by Age bin
Total revenue
0
5
10
15
20
25
18-24
30-34
40-44
50-54
60-64
70+
Age bin

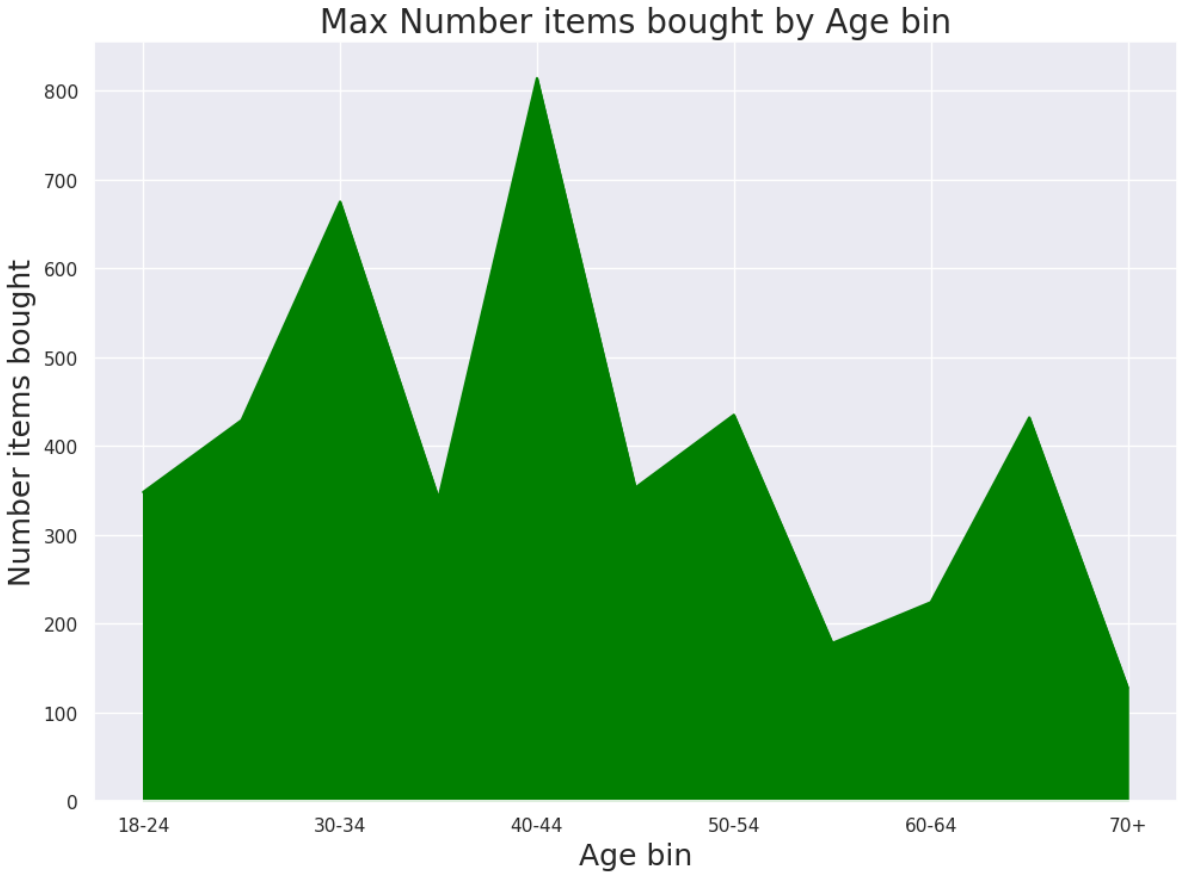

Max Total revenue by Age bin
Total revenue
4000
3500
3000
2500
2000
1500
1000
500
0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

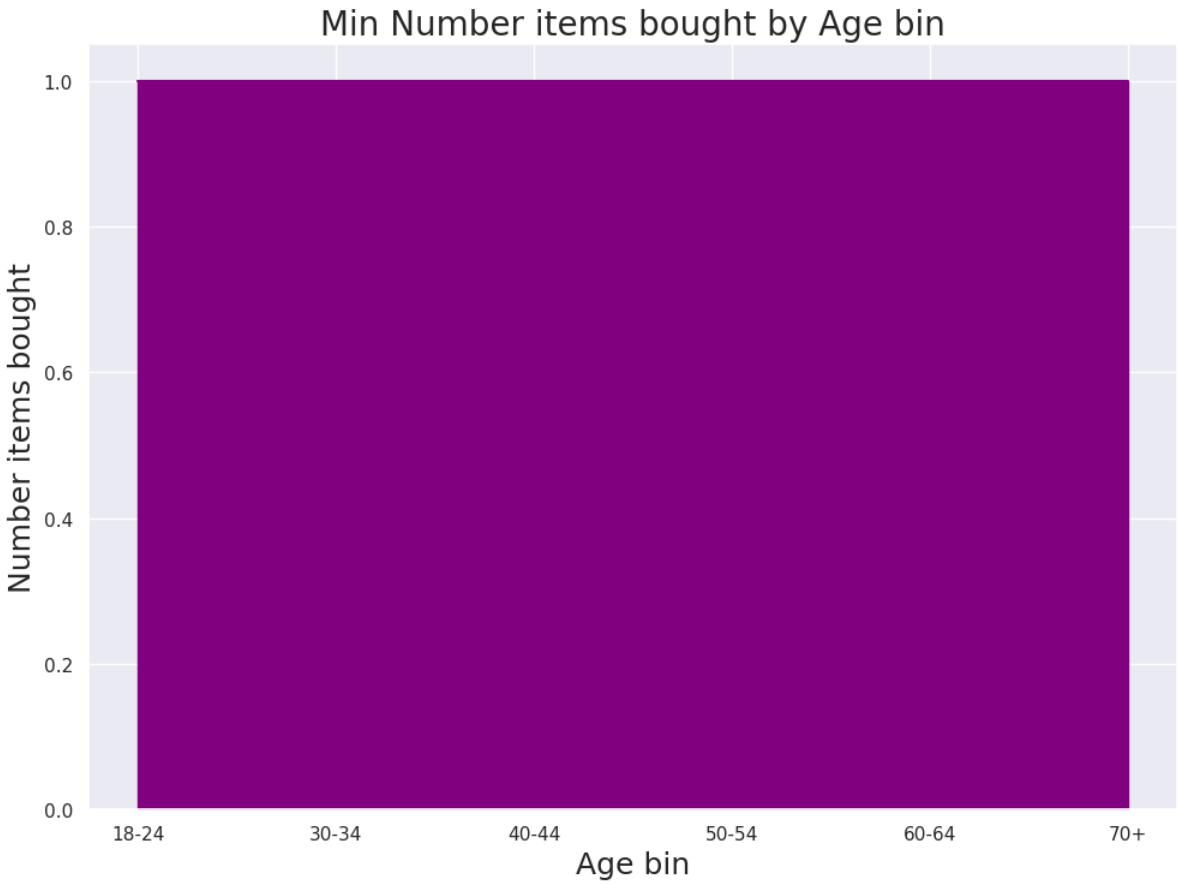

Min Total revenue by Age bin
Total revenue
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

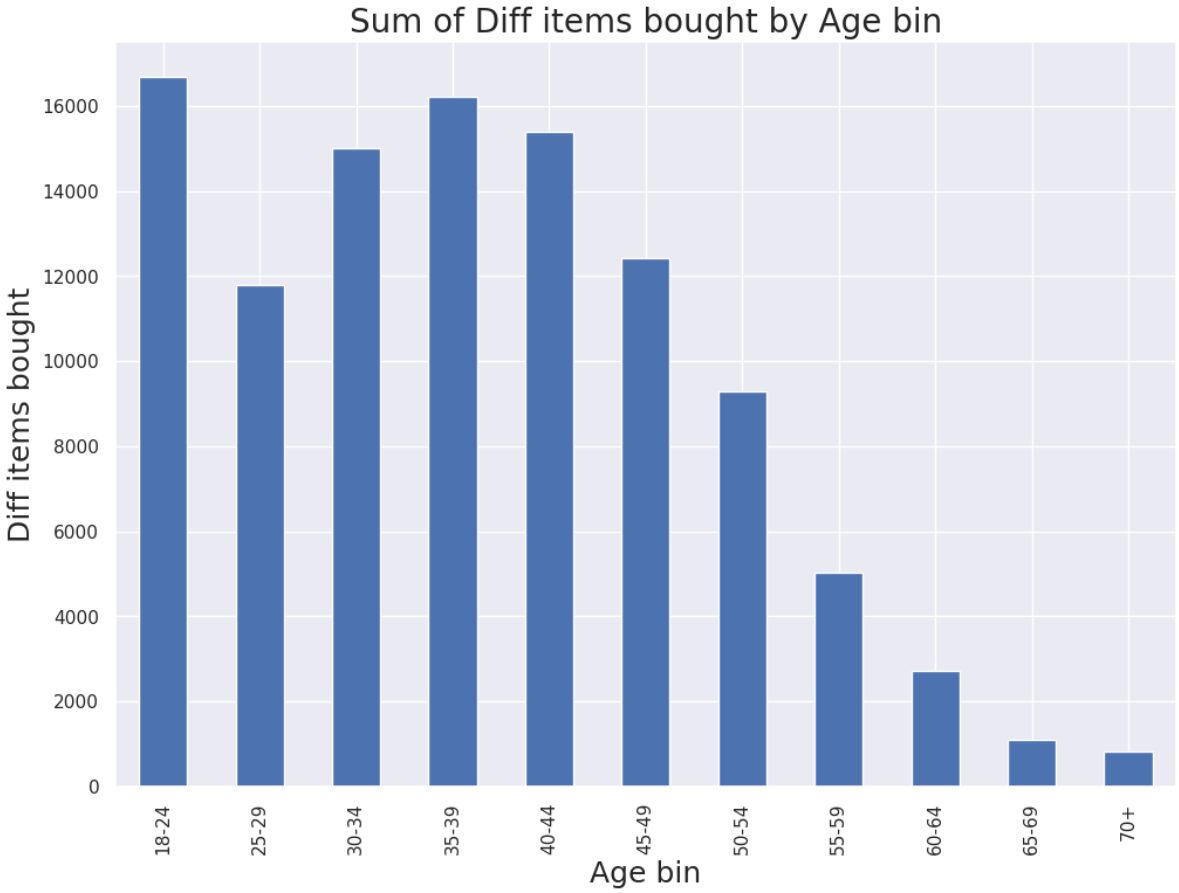

Sum of Number items bought by Age bin
Number items bought
30000
25000
20000
15000
10000
5000
0
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+
Age bin

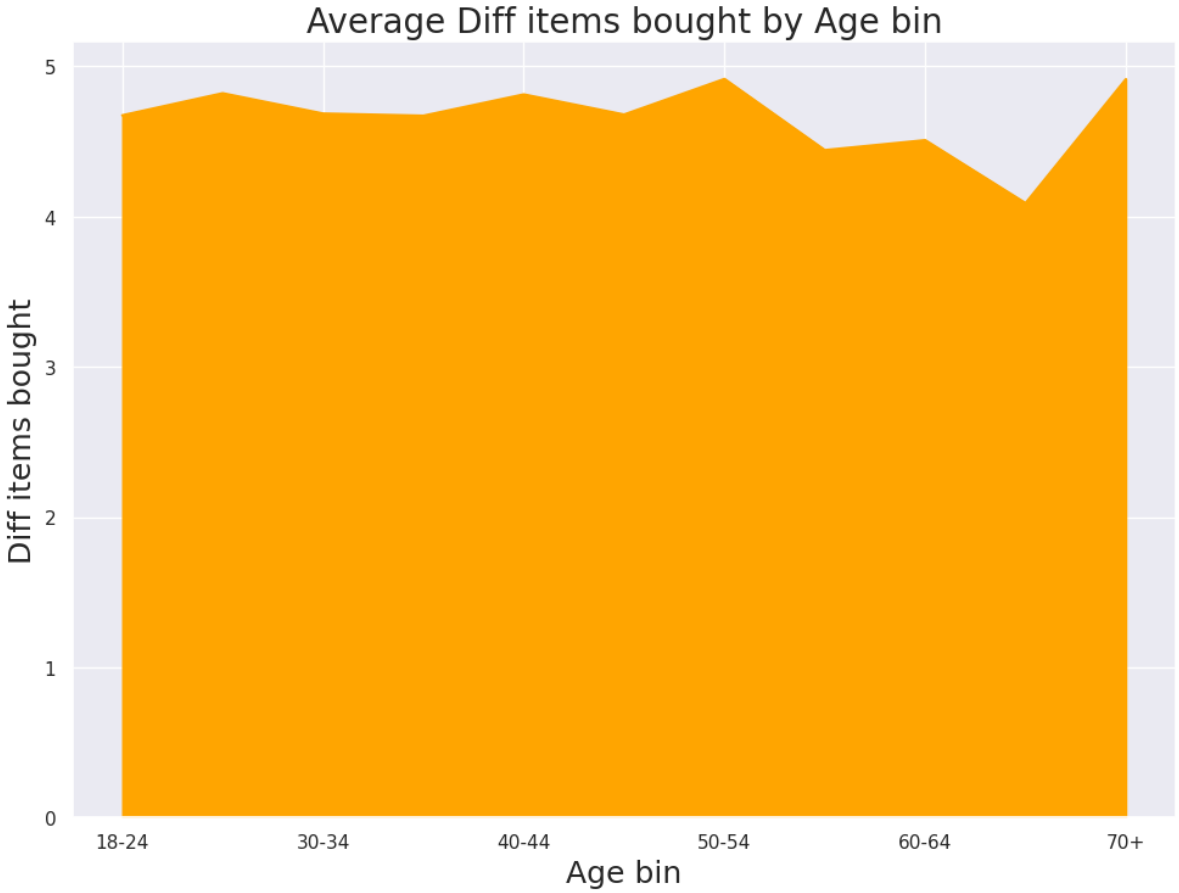

Average Number items bought by Age bin
Number items bought
8
6
4
2
0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

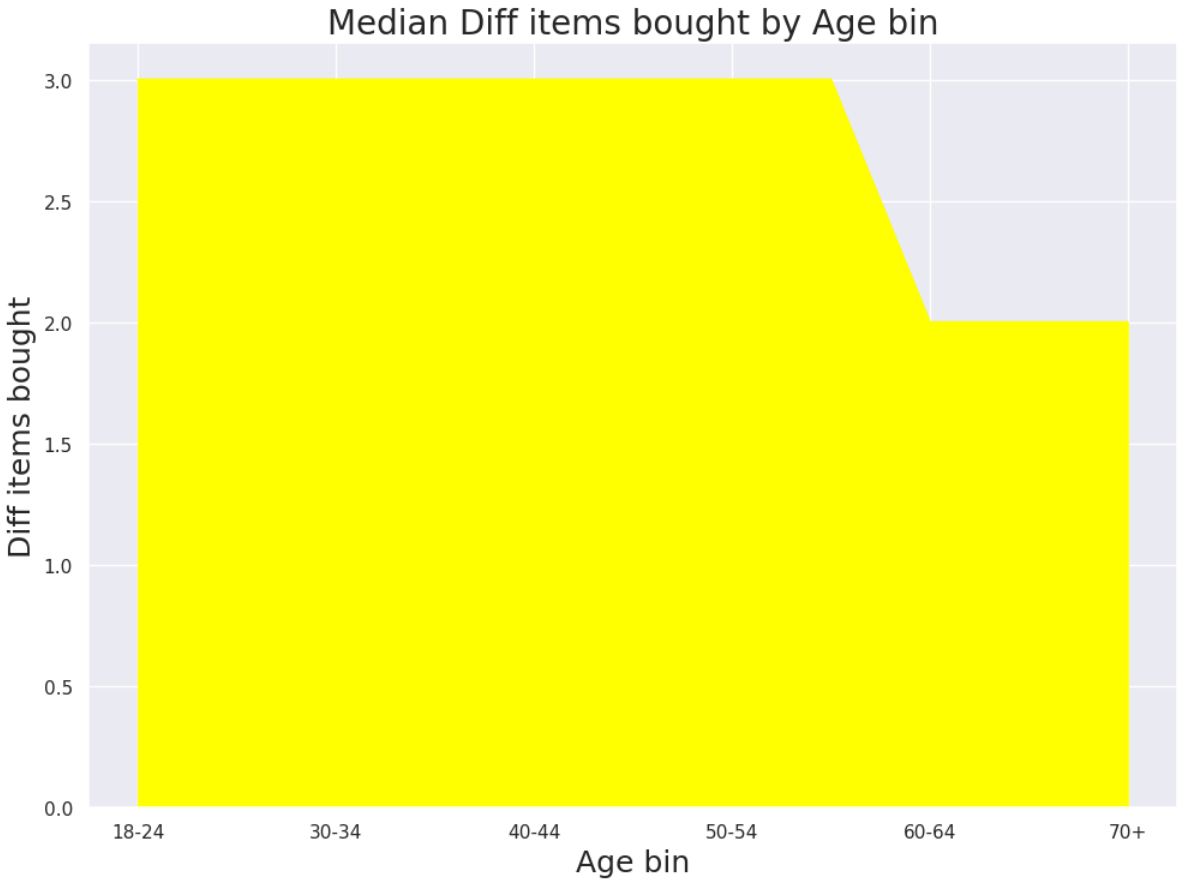

Median Number items bought by Age bin
Number items bought
3.0
2.5
2.0
1.5
1.0
0.5
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

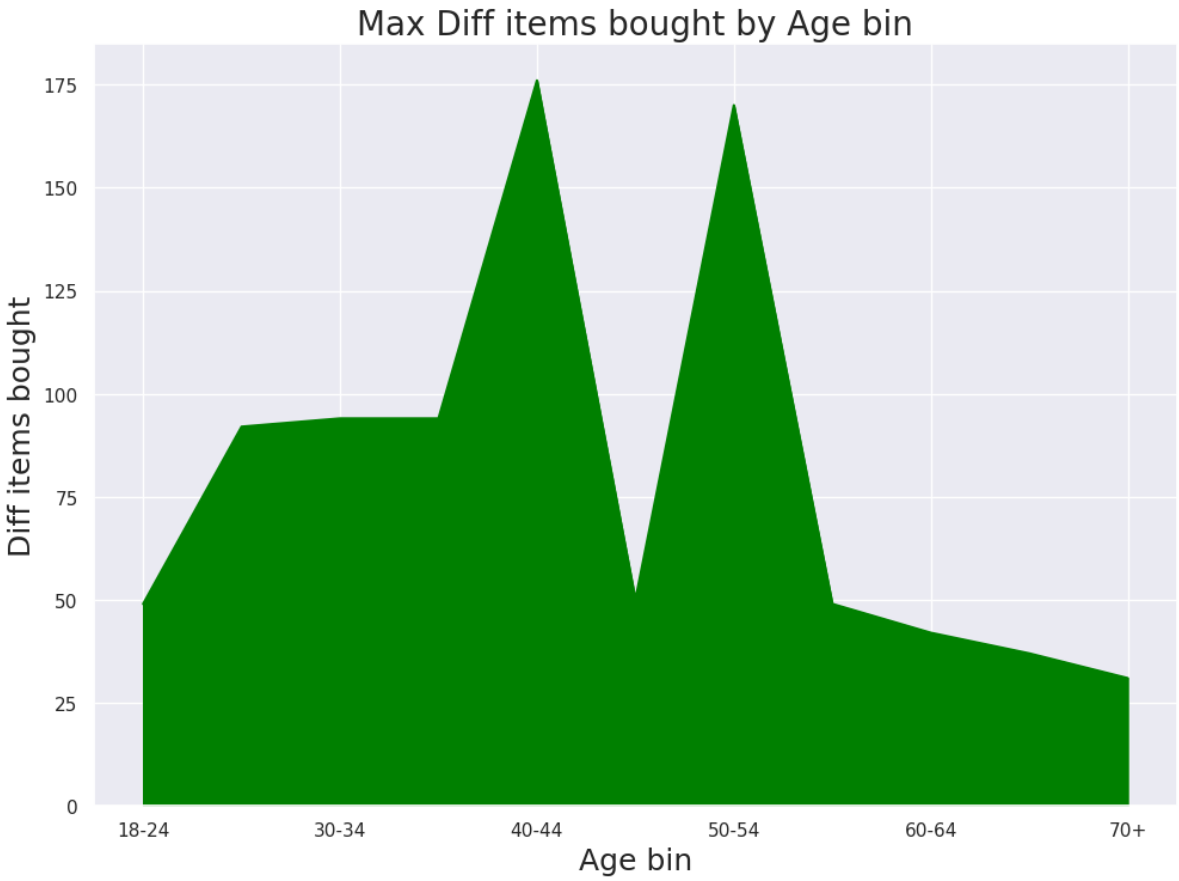

Max Number items bought by Age bin
Number items bought
800
700
600
500
400
300
200
100
0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

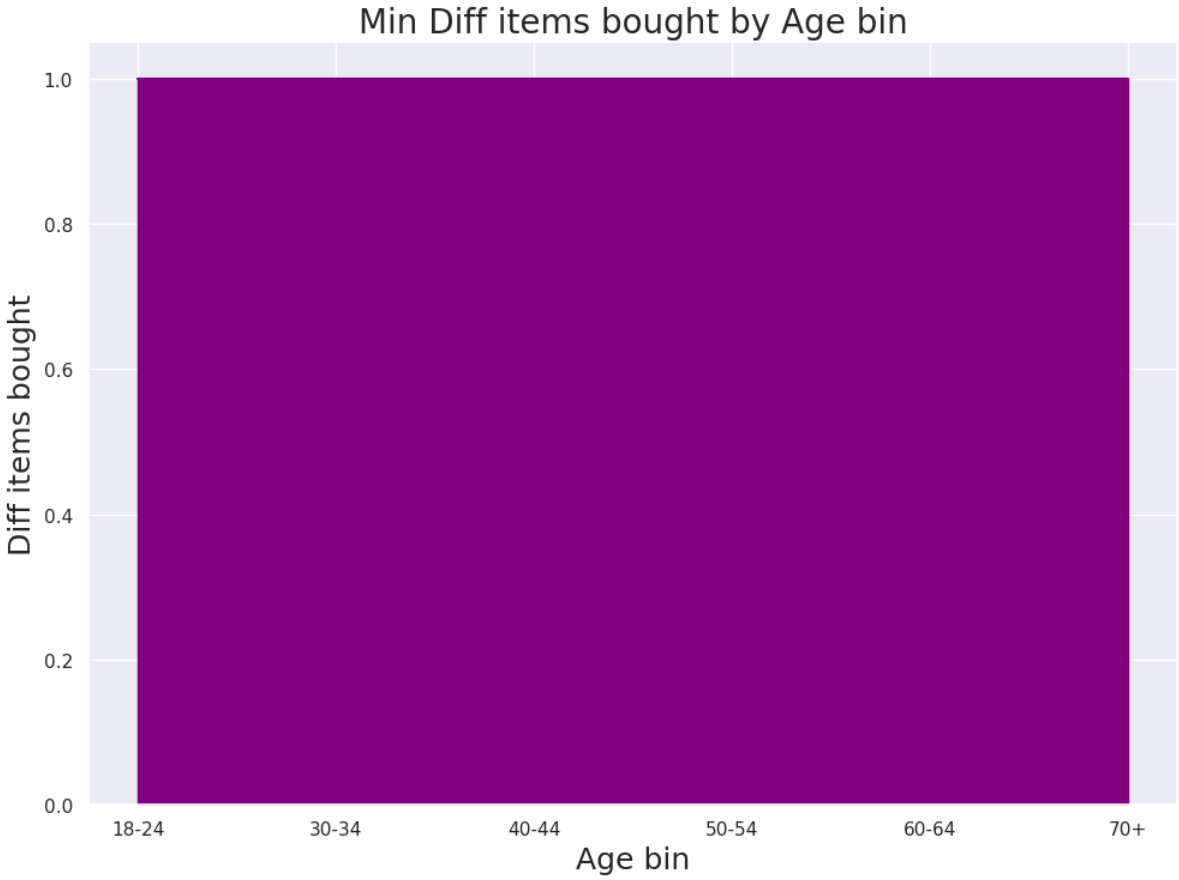

Min Number items bought by Age bin
Number items bought
1.0
0.8
0.6
0.4
0.2
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

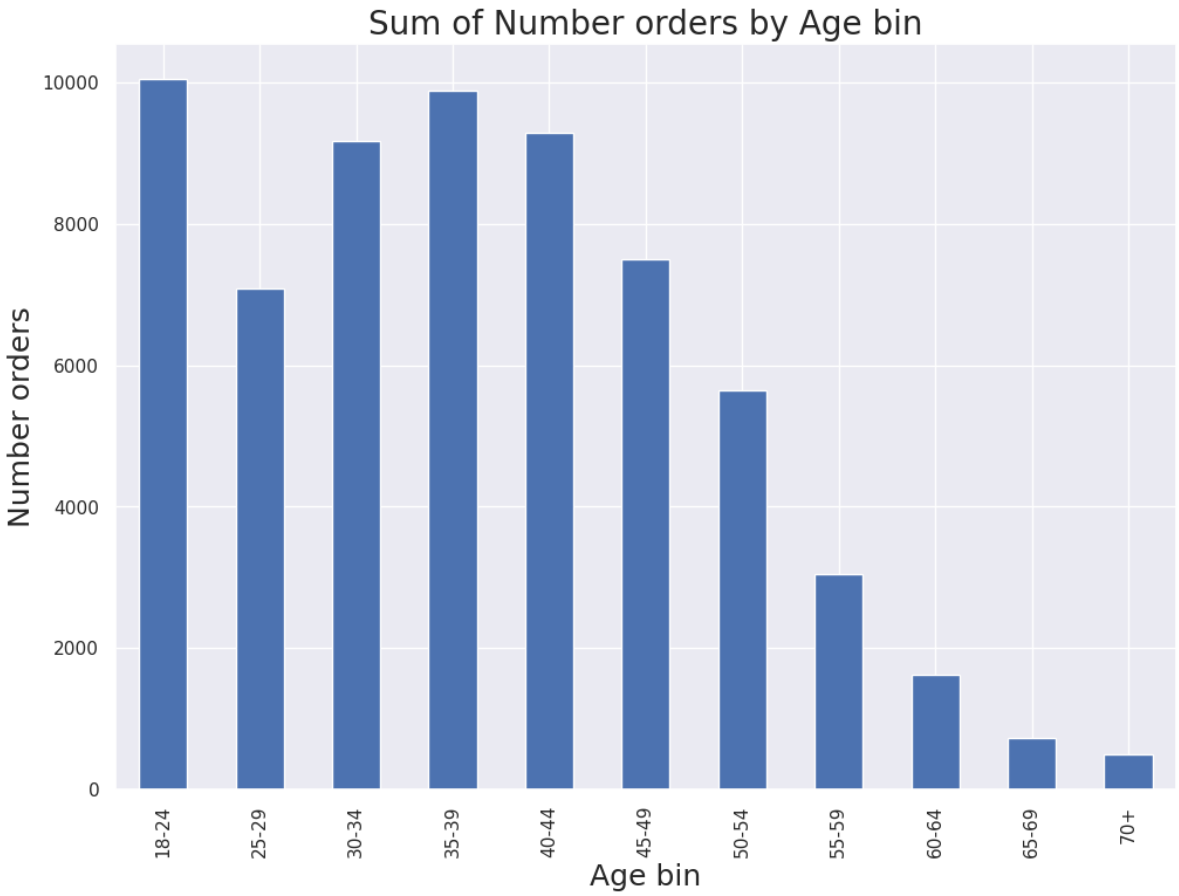

Sum of Diff items bought by Age bin
Diff items bought
16000
14000
12000
10000
8000
6000
4000
2000
0
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+
Age bin

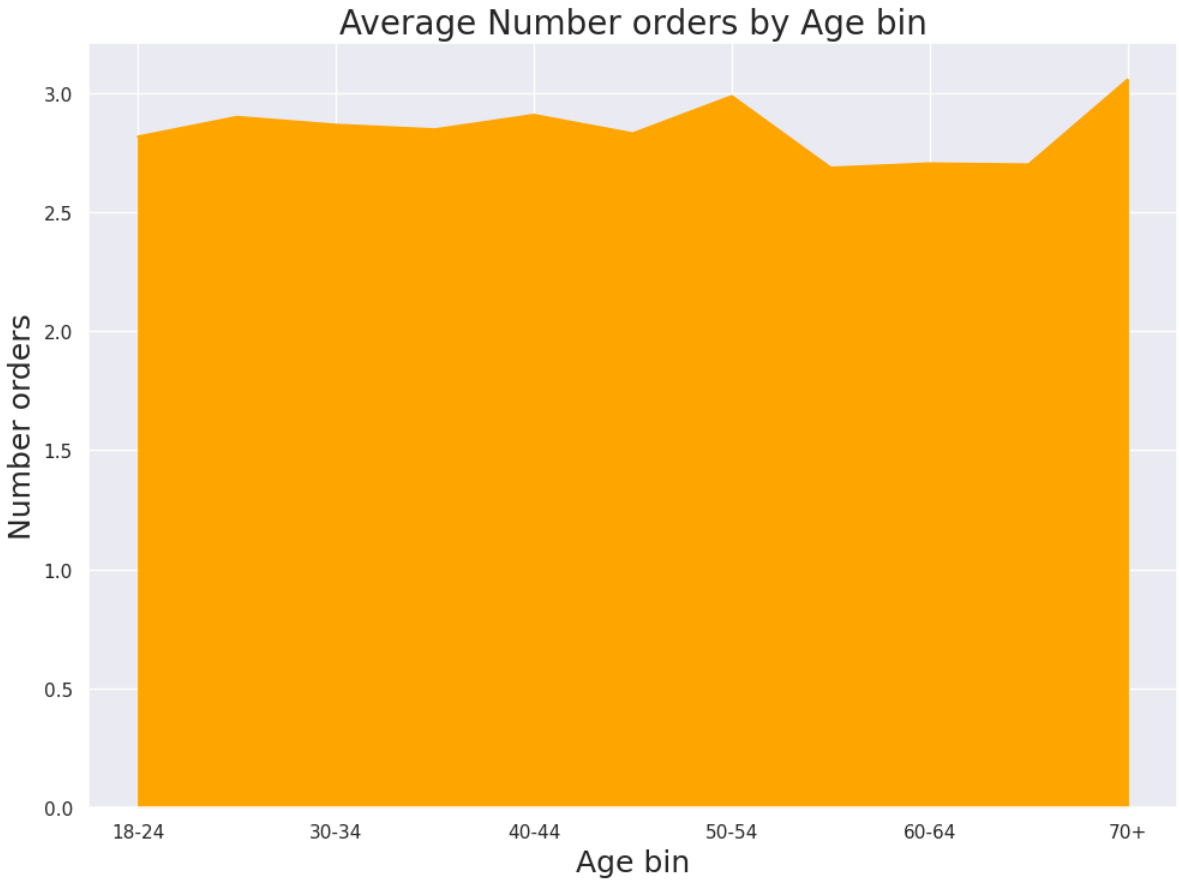

Average Diff items bought by Age bin
Diff items bought
0
1
2
3
4
5
18-24
30-34
40-44
50-54
60-64
70+
Age bin

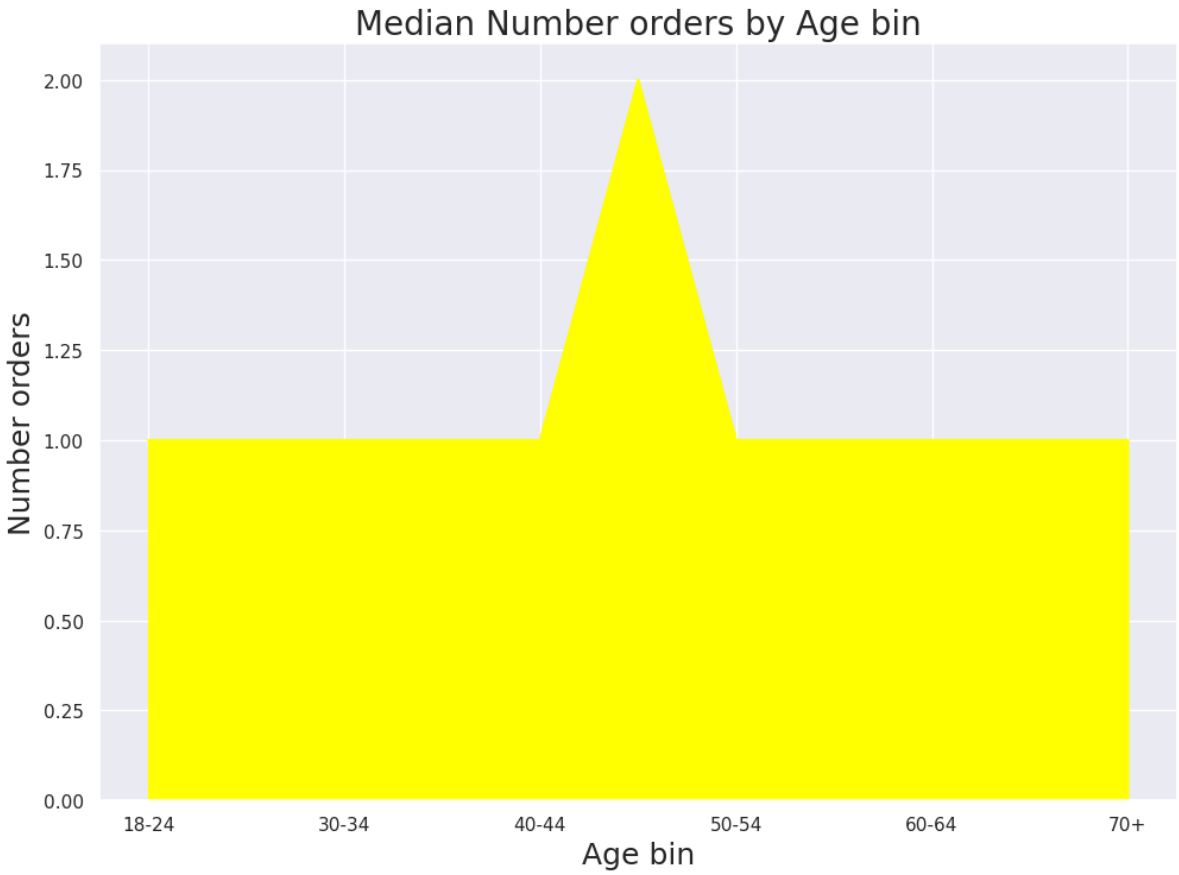

Median Diff items bought by Age bin
Diff items bought
0.0
0.5
1.0
1.5
2.0
2.5
3.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

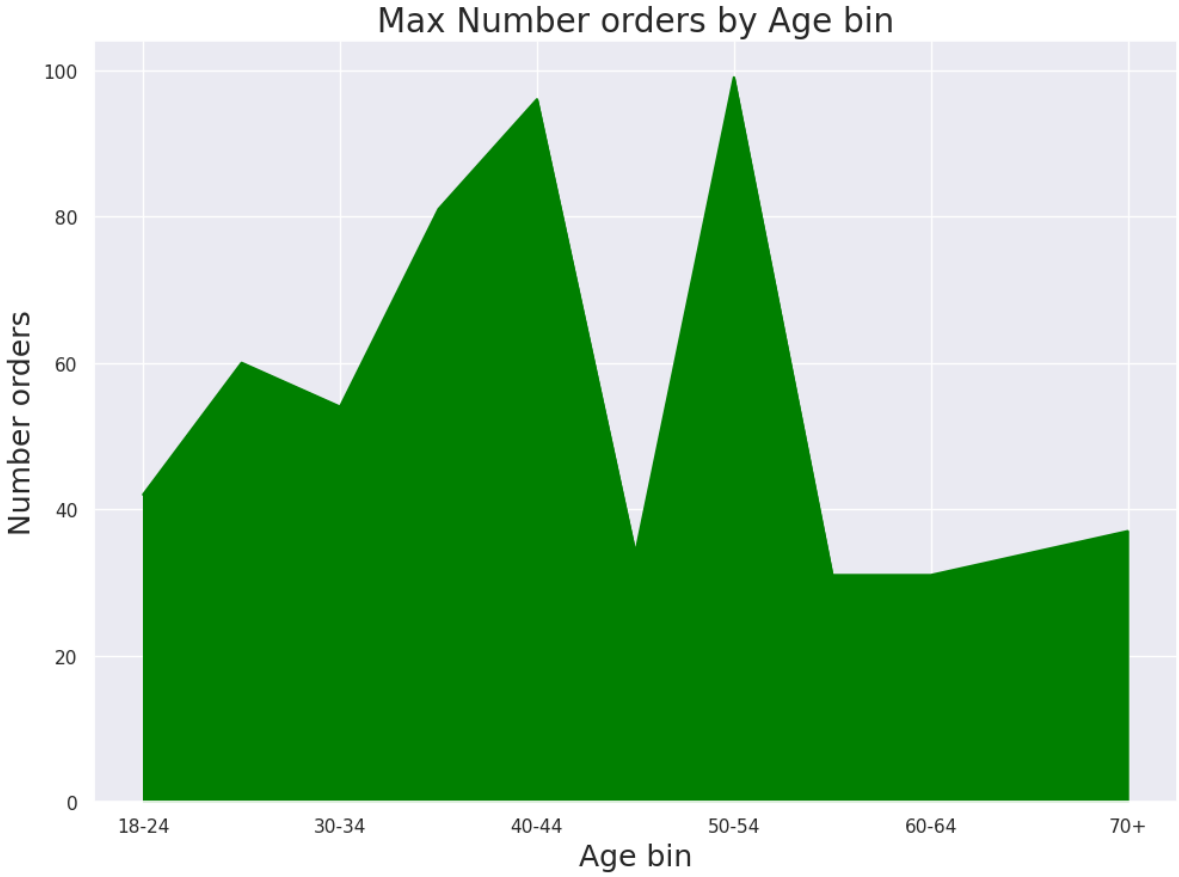

Max Diff items bought by Age bin
Diff items bought
175
150
125
100
75
50
25
0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

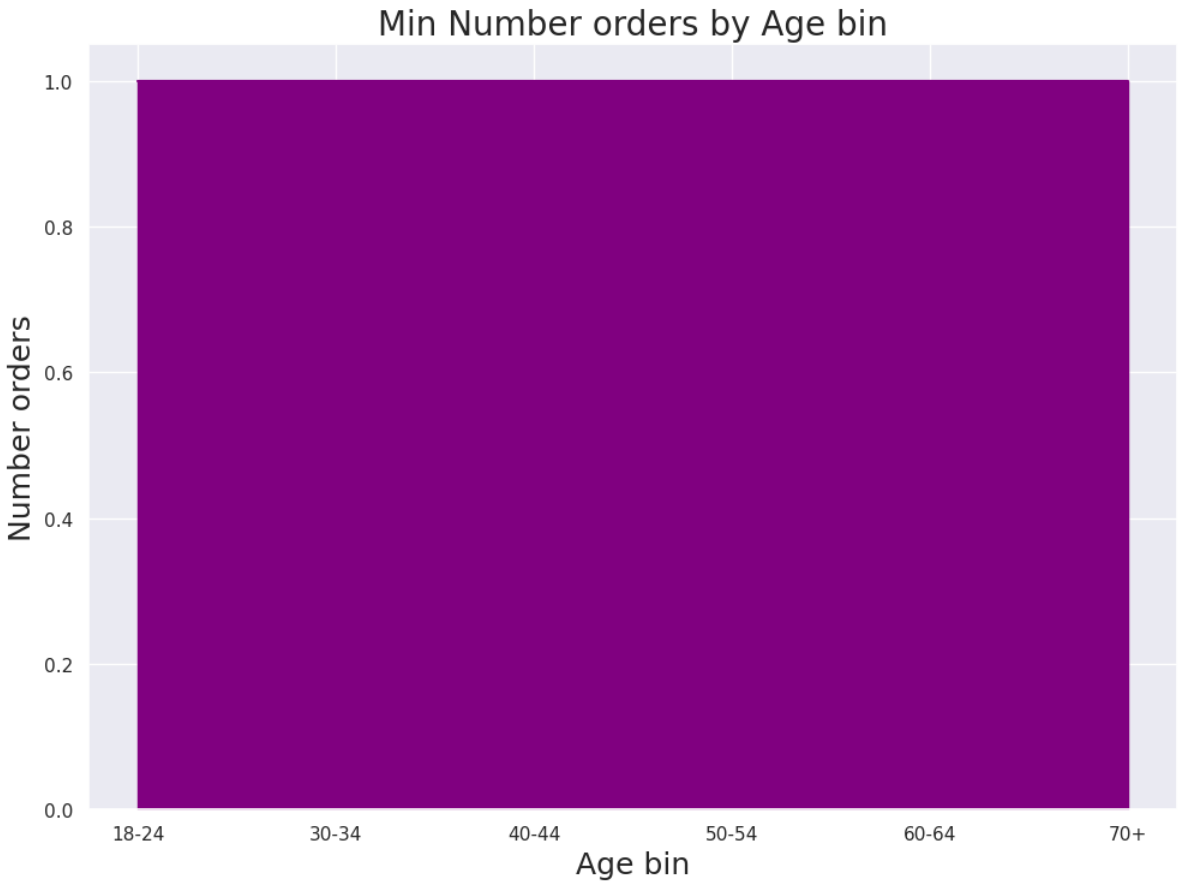

Min Diff items bought by Age bin
Diff items bought
1.0
0.8
0.6
0.4
0.2
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

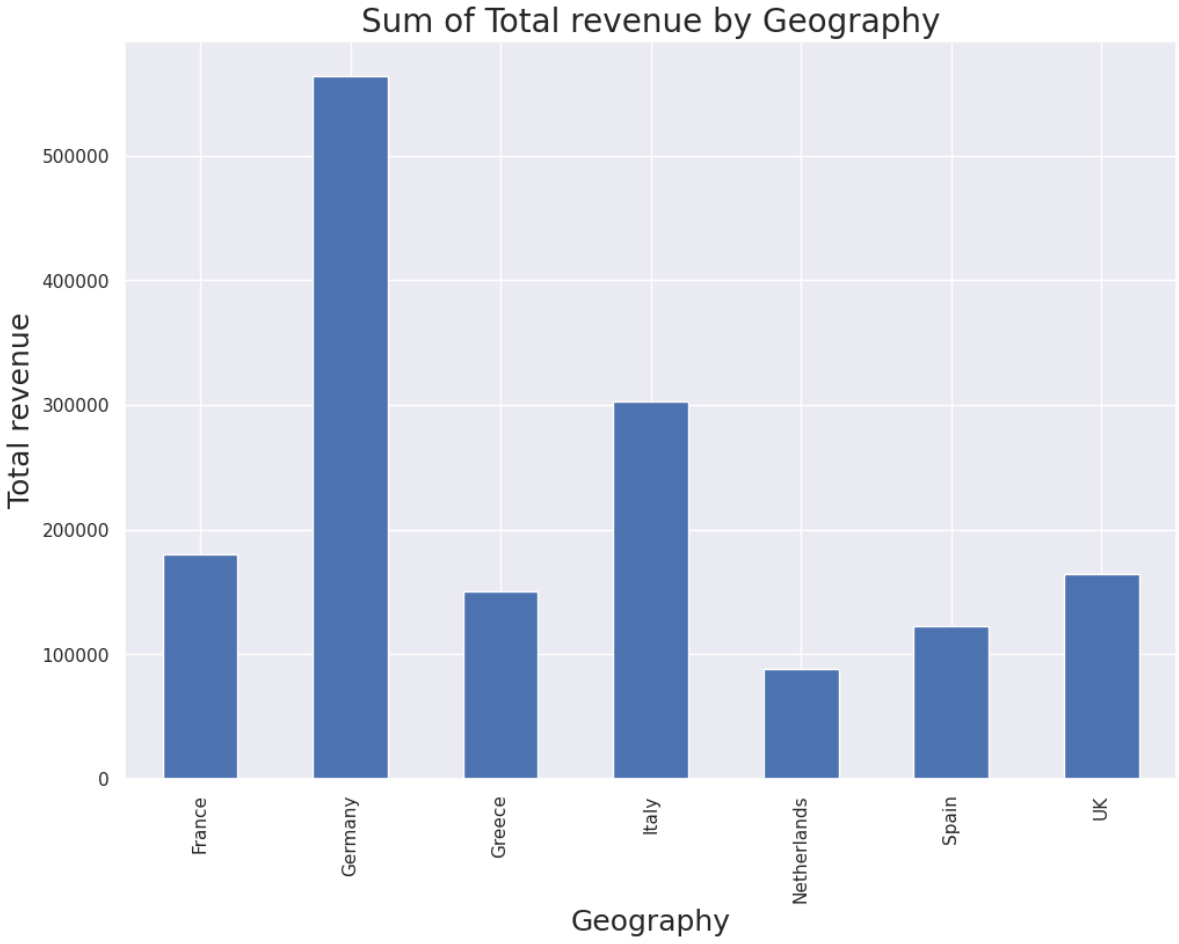

Sum of Number orders by Age bin
Number orders
10000
8000
6000
4000
2000
0
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+
Age bin

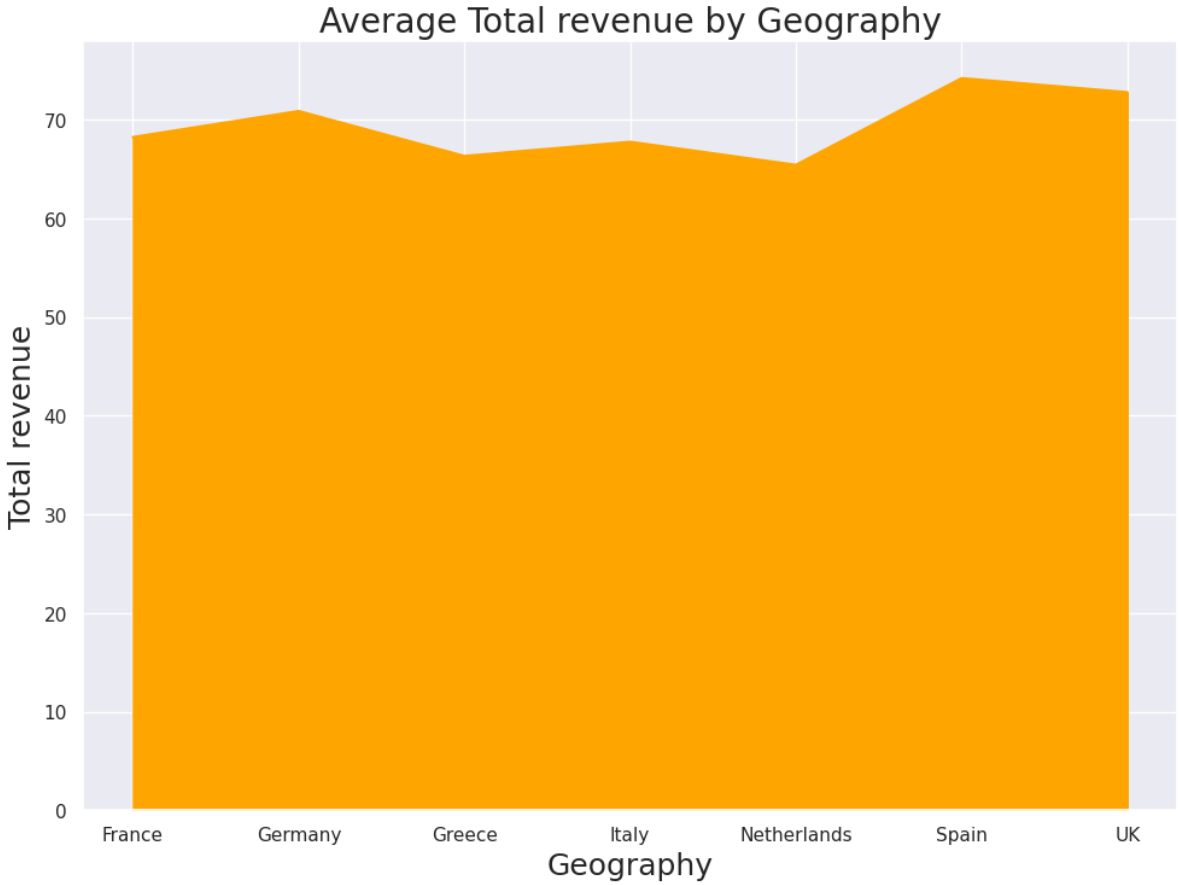

Average Number orders by Age bin
Number orders
3.0
2.5
2.0
1.5
1.0
0.5
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

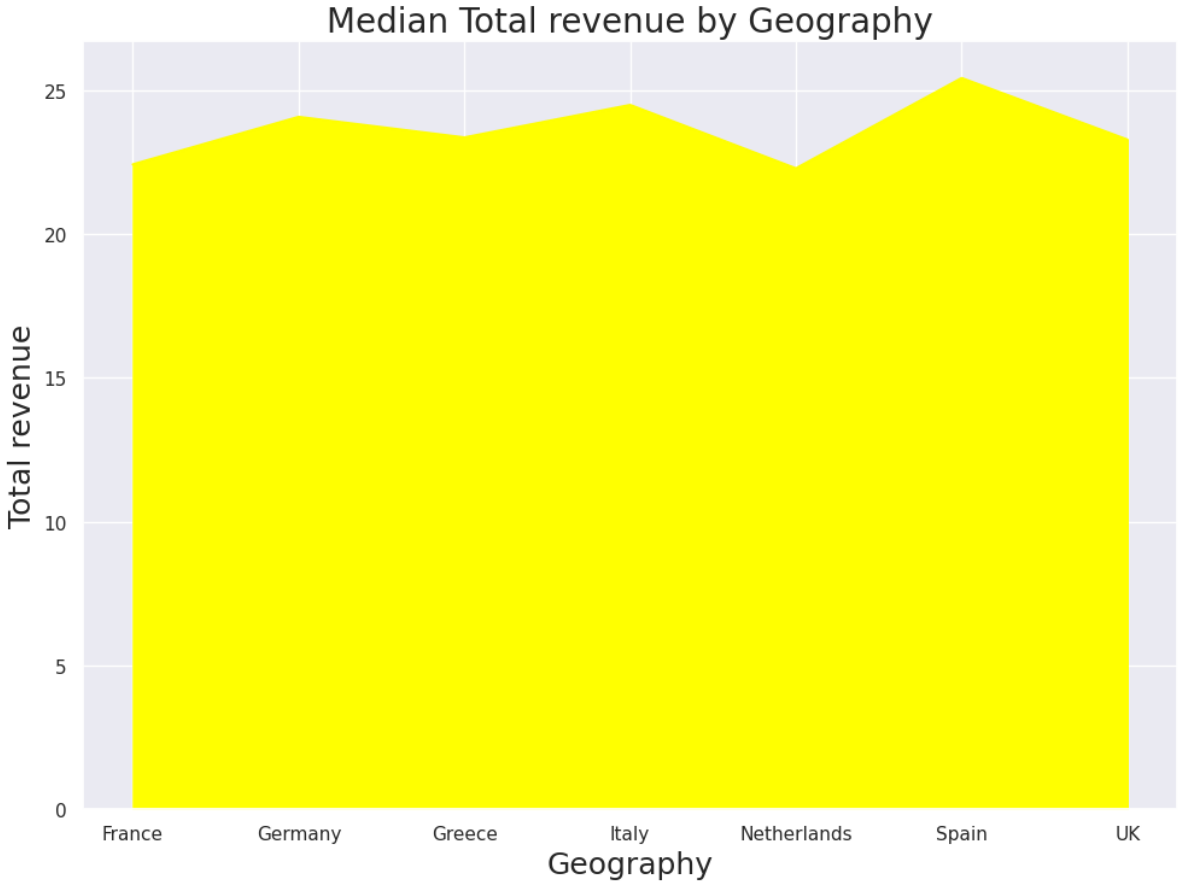

Median Number orders by Age bin
Number orders
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
18-24
30-34
40-44
50-54
60-64
70+
Age bin

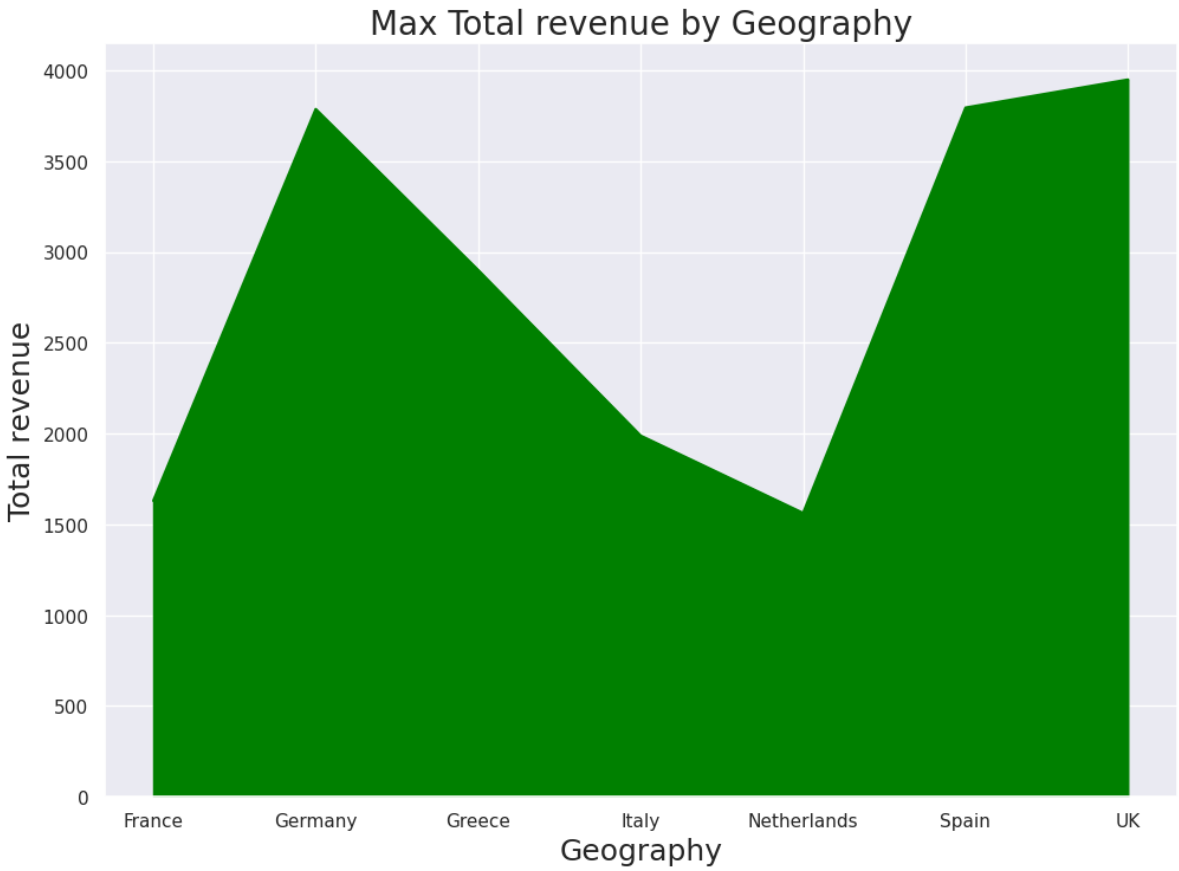

Max Number orders by Age bin
Number orders
100
80
60
40
20
0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

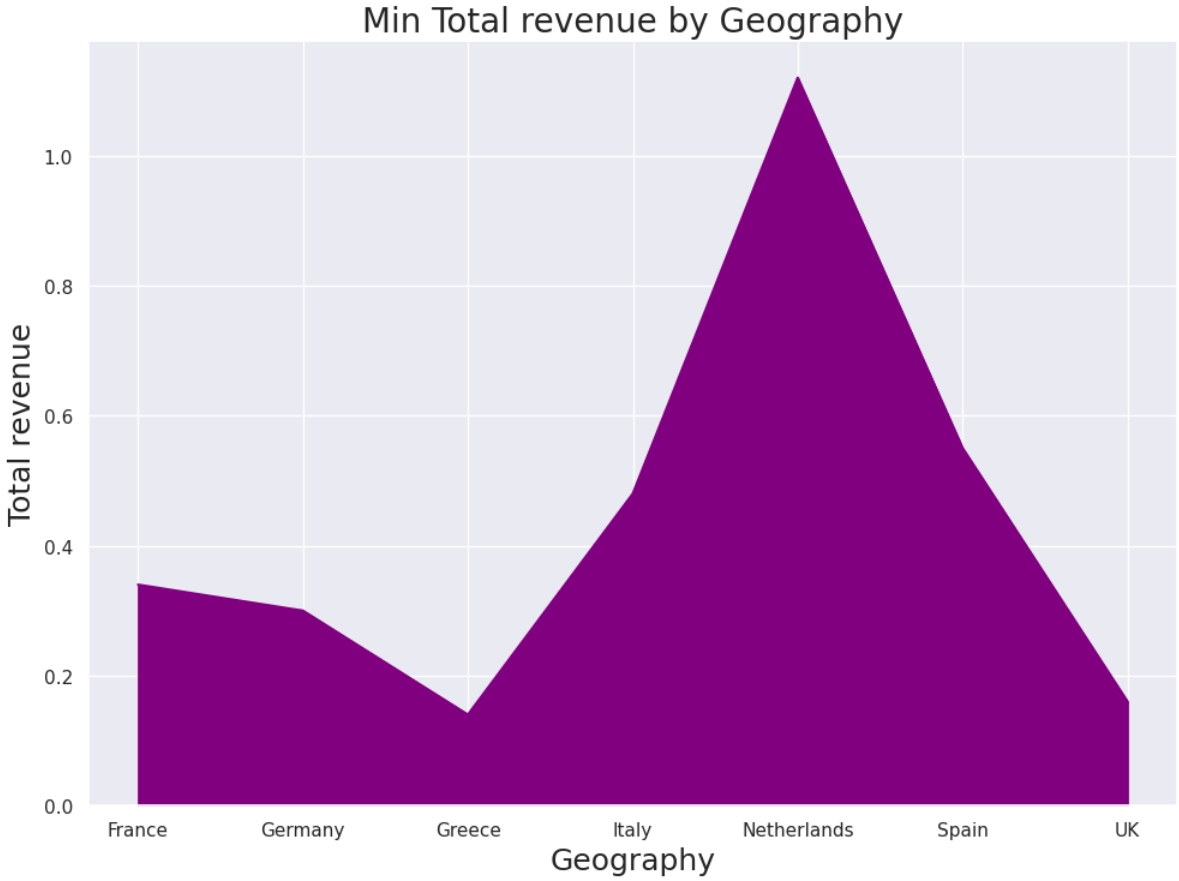

Min Number orders by Age bin
Number orders
1.0
0.8
0.6
0.4
0.2
0.0
18-24
30-34
40-44
50-54
60-64
70+
Age bin

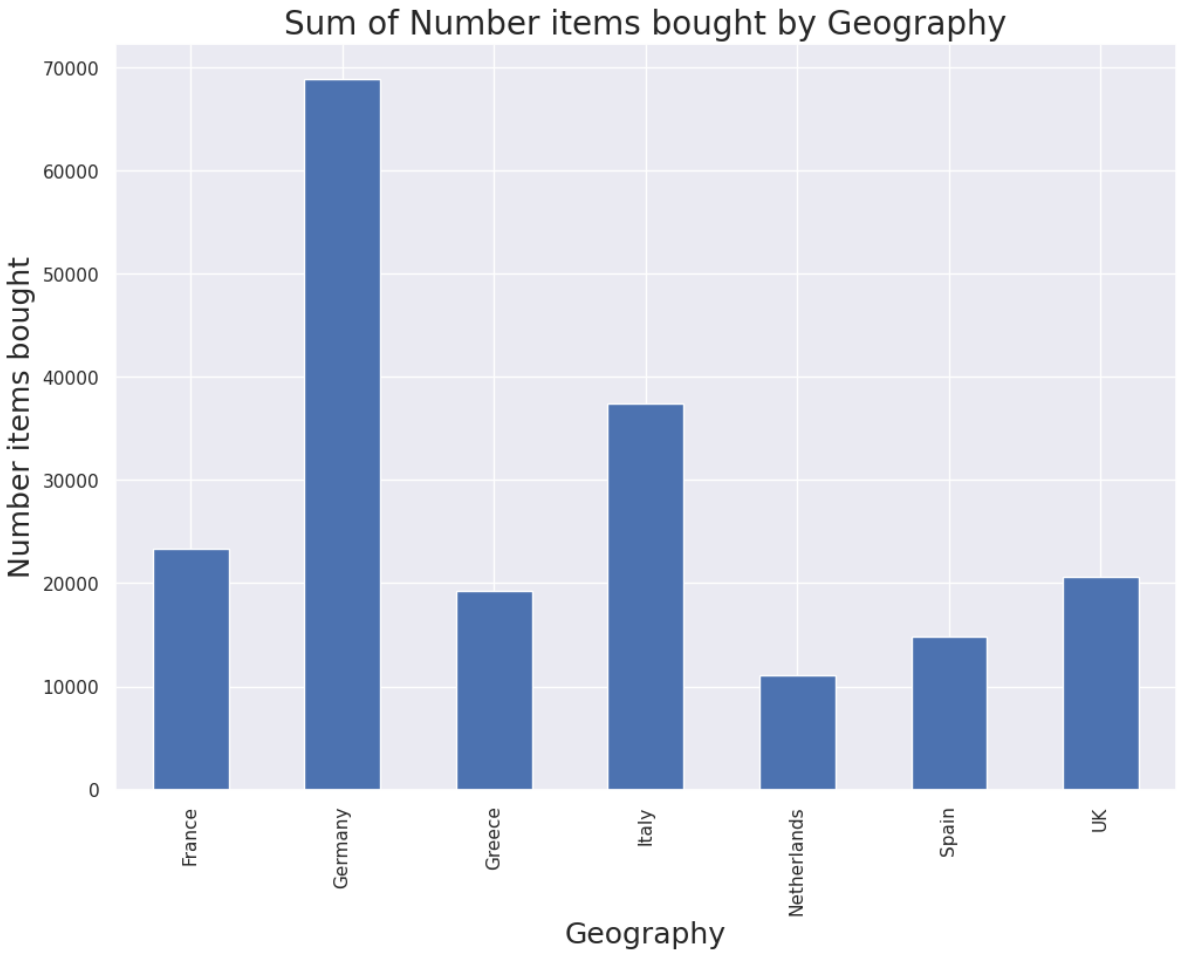

Sum of Total revenue by Geography
Total revenue
500000
400000
300000
200000
100000
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

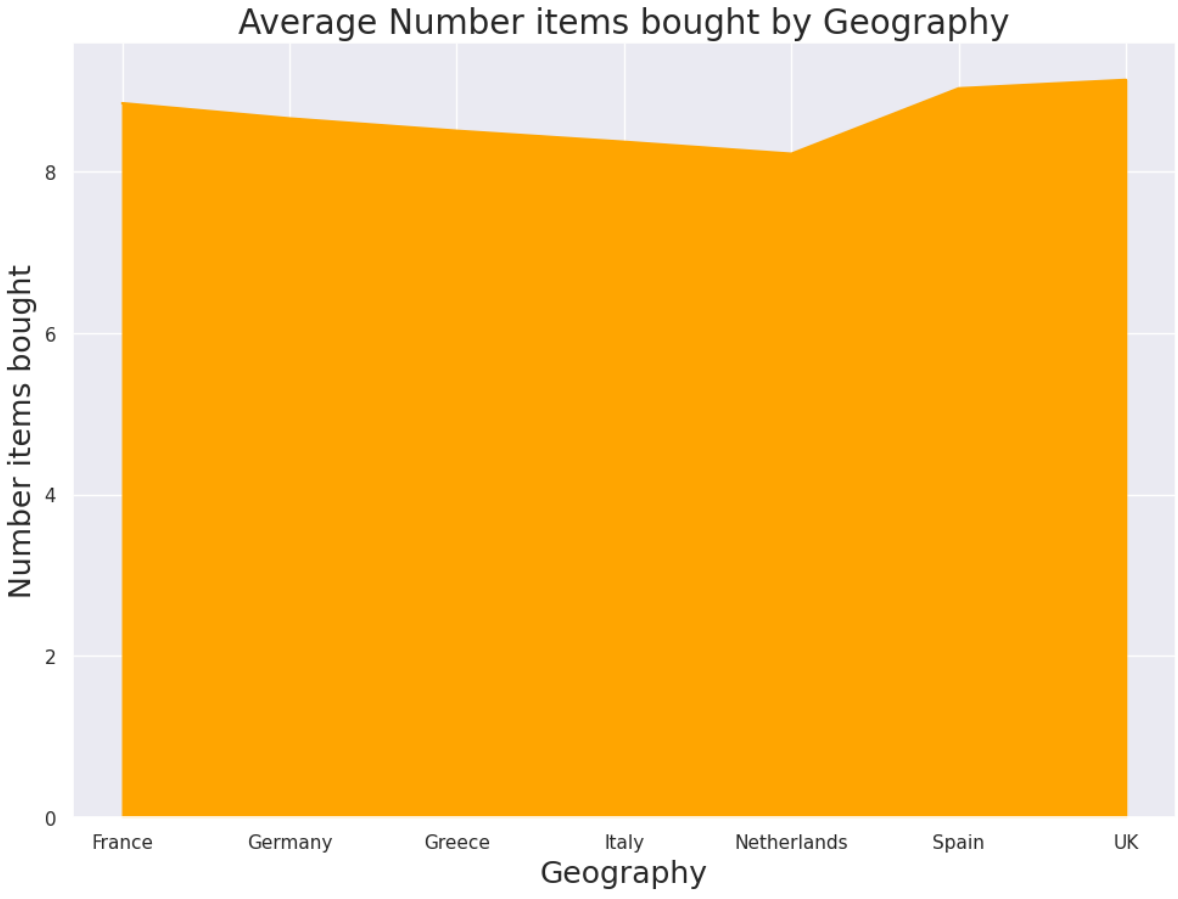

Average Total revenue by Geography
Total revenue
0
10
20
30
40
50
60
70
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

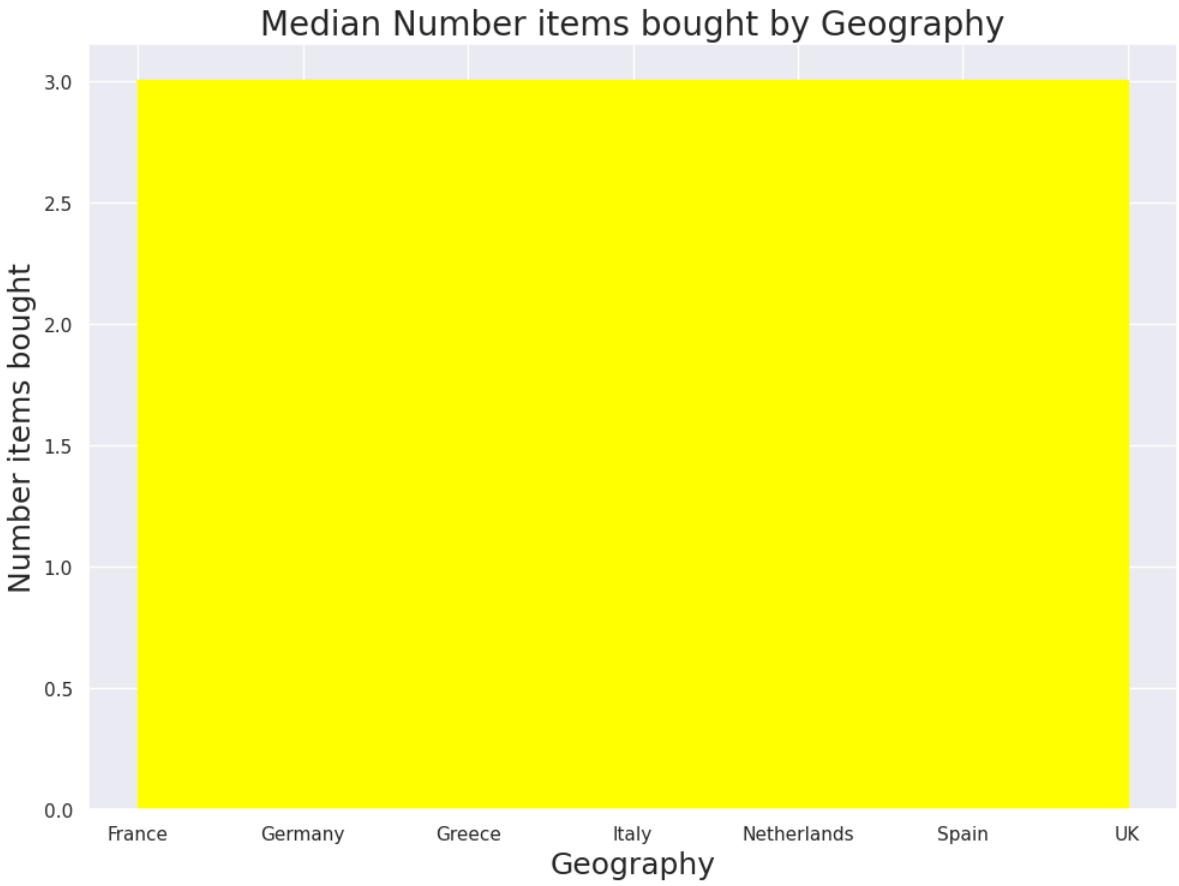

Median Total revenue by Geography
Total revenue
0
5
10
15
20
25
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

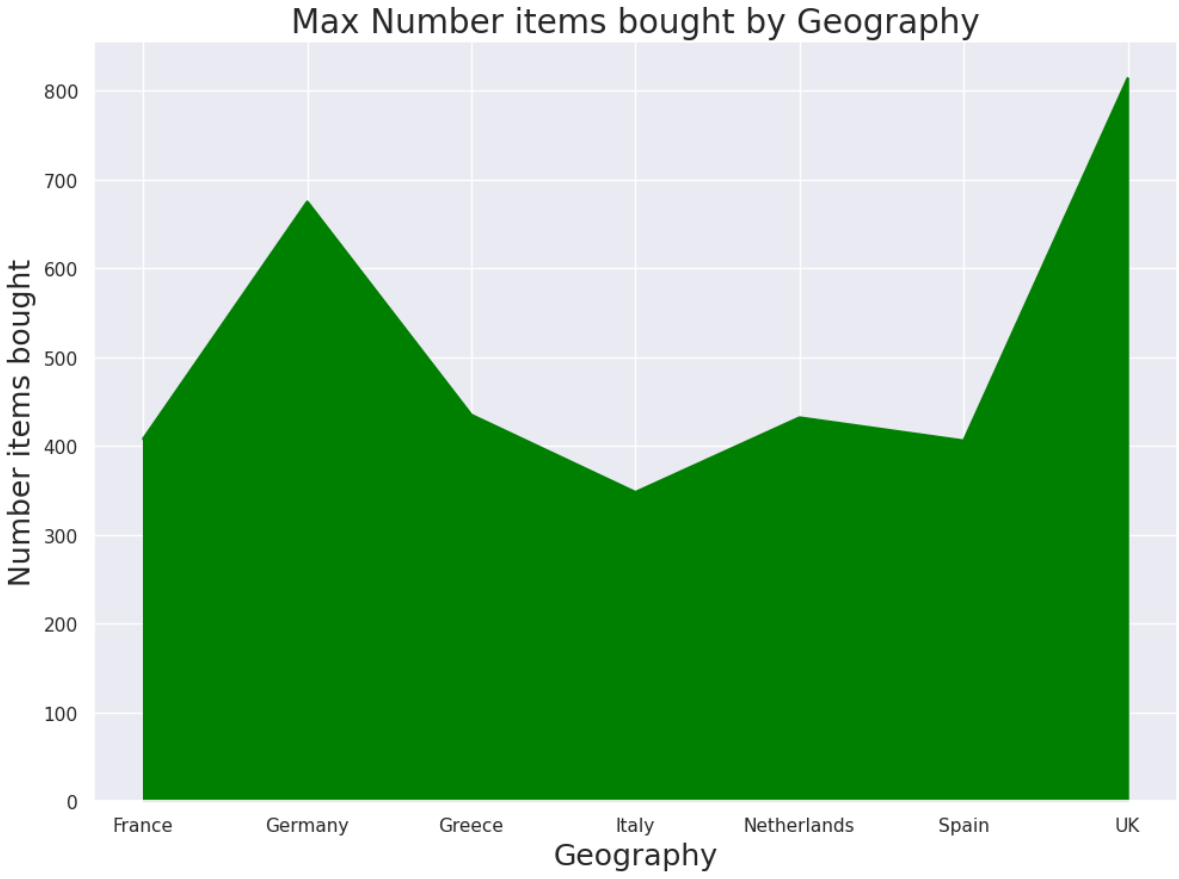

Max Total revenue by Geography
Total revenue
4000
3500
3000
2500
2000
1500
1000
500
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

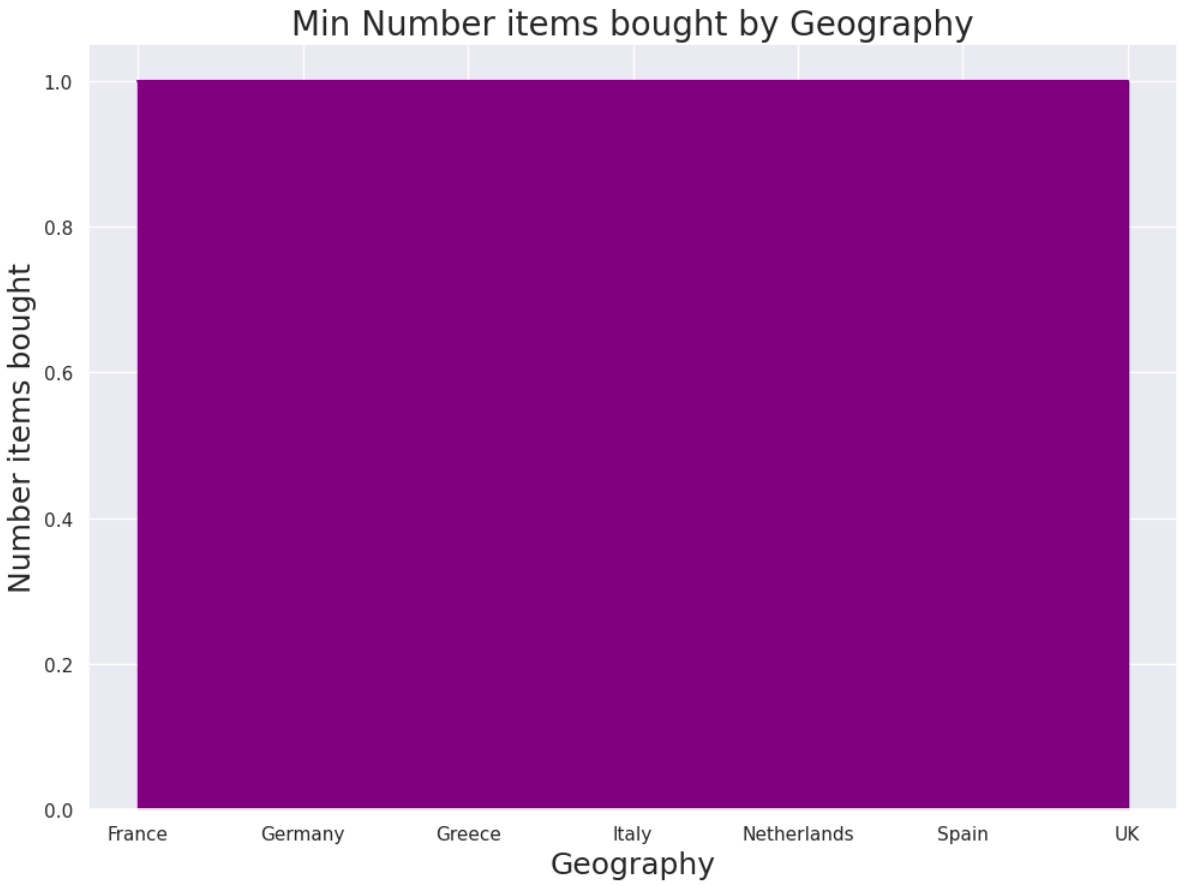

Min Total revenue by Geography
Total revenue
1.0
0.8
0.6
0.4
0.2
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

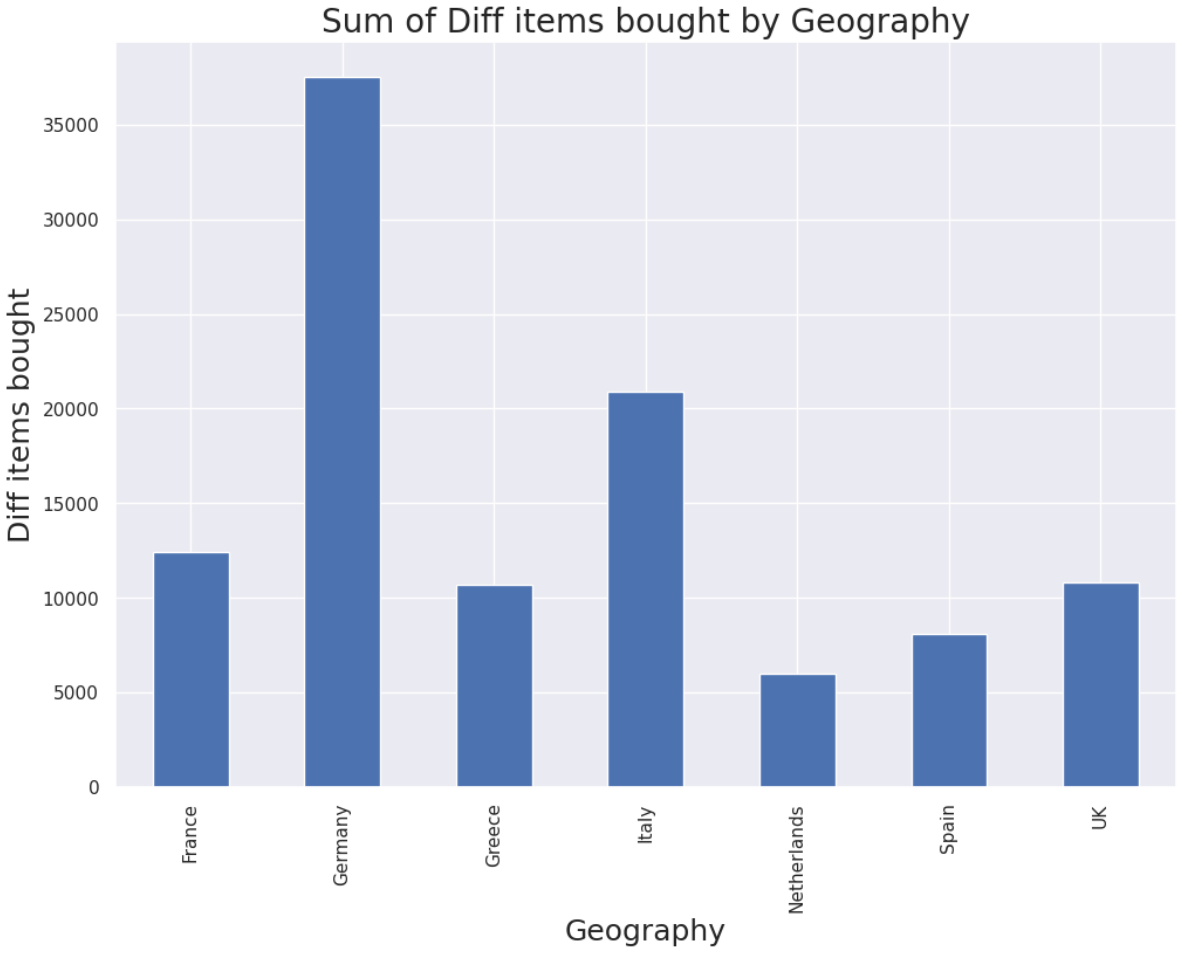

Sum of Number items bought by Geography
Number items bought
70000
60000
50000
40000
30000
20000
10000
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

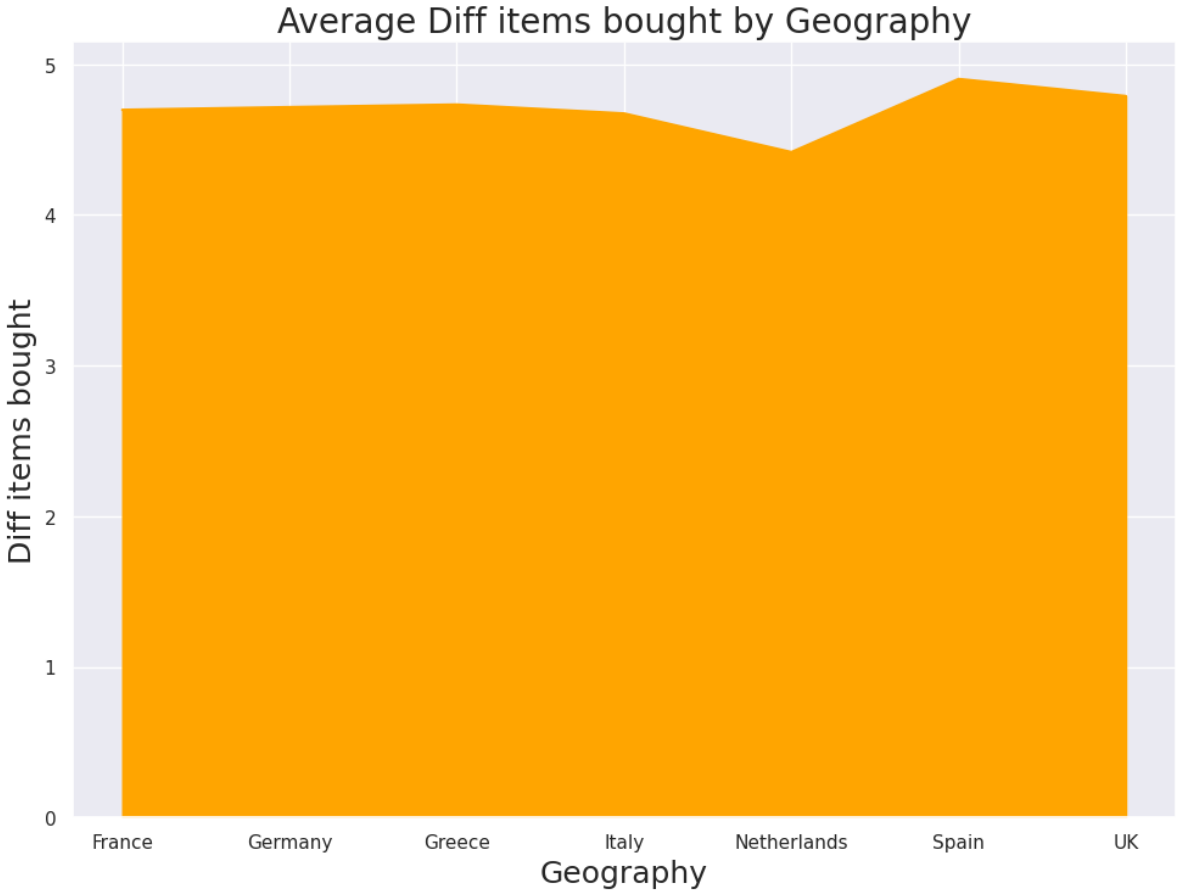

Average Number items bought by Geography
Number items bought
8
6
4
2
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

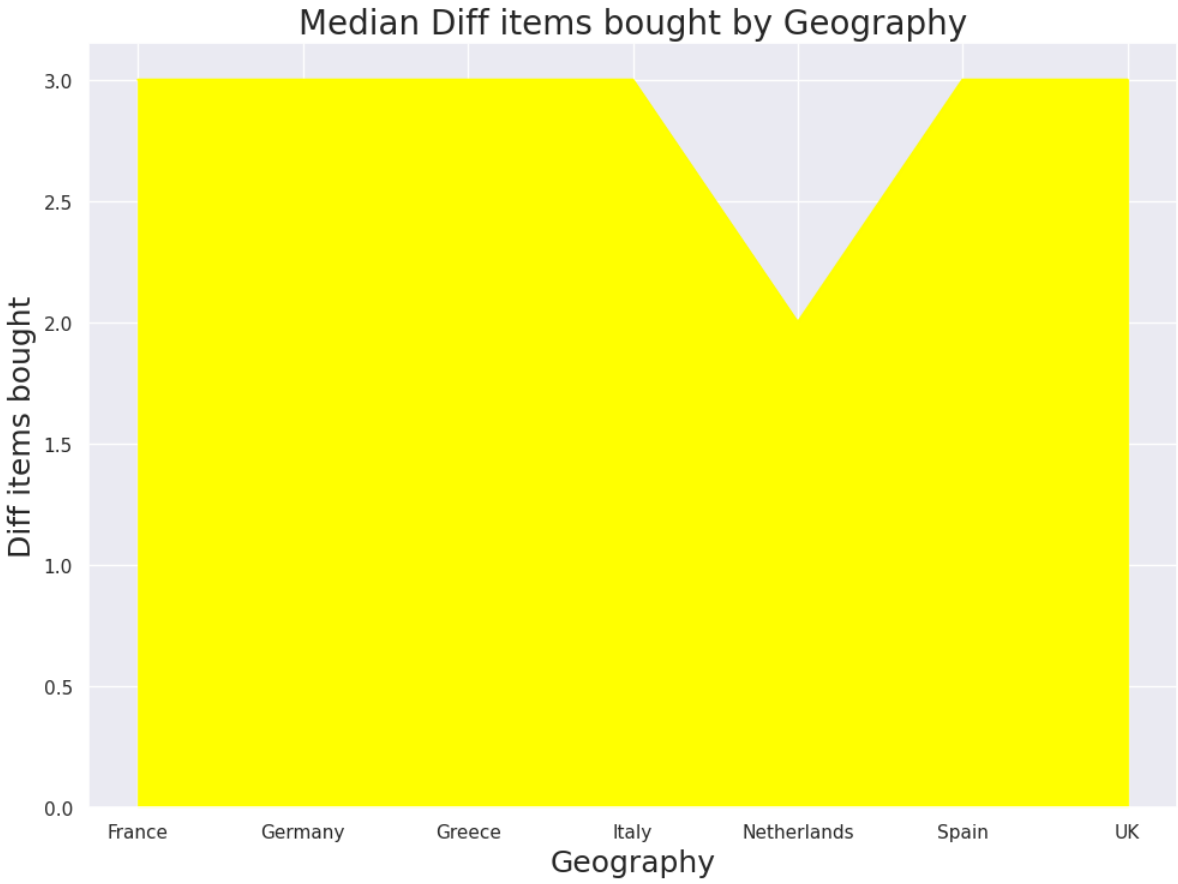

Median Number items bought by Geography
Number items bought
3.0
2.5
2.0
1.5
1.0
0.5
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

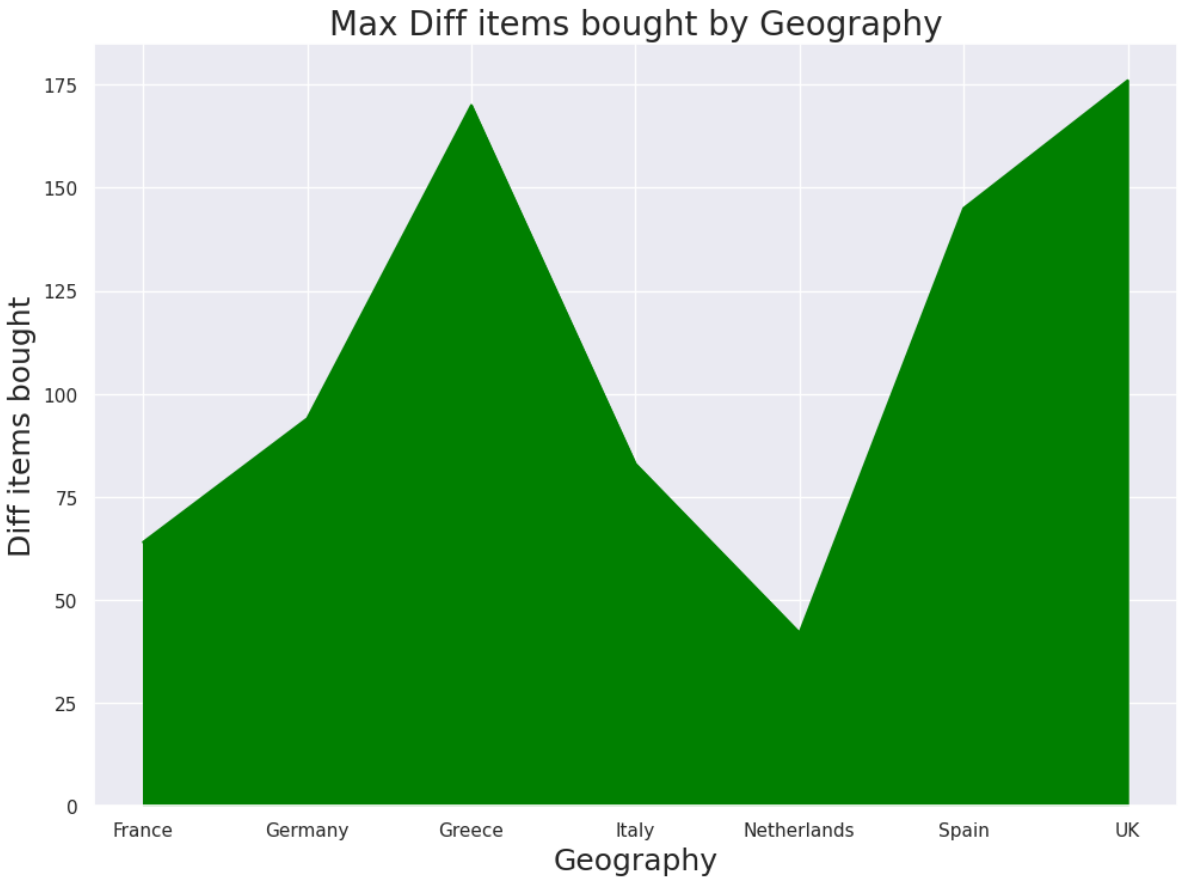

Max Number items bought by Geography
Number items bought
800
700
600
500
400
300
200
100
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

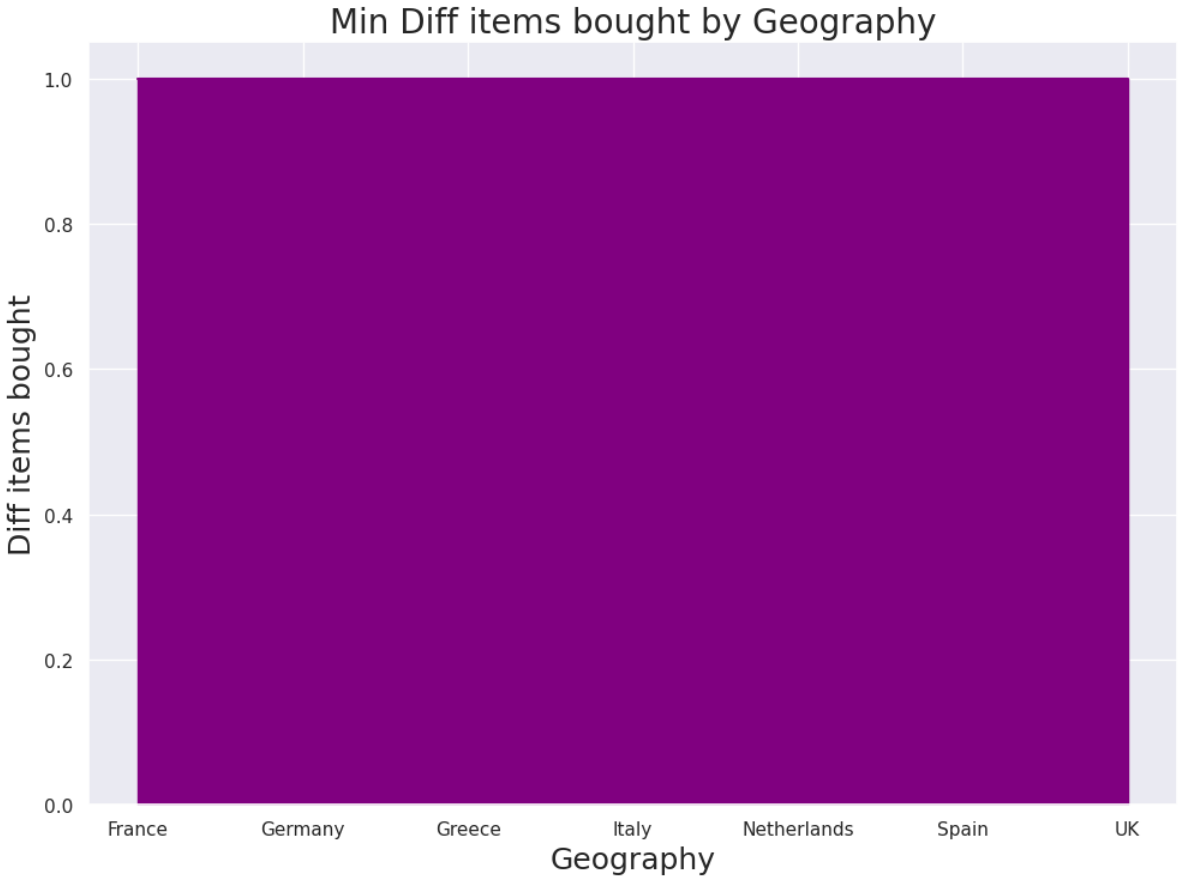

Min Number items bought by Geography
Number items bought
1.0
0.8
0.6
0.4
0.2
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

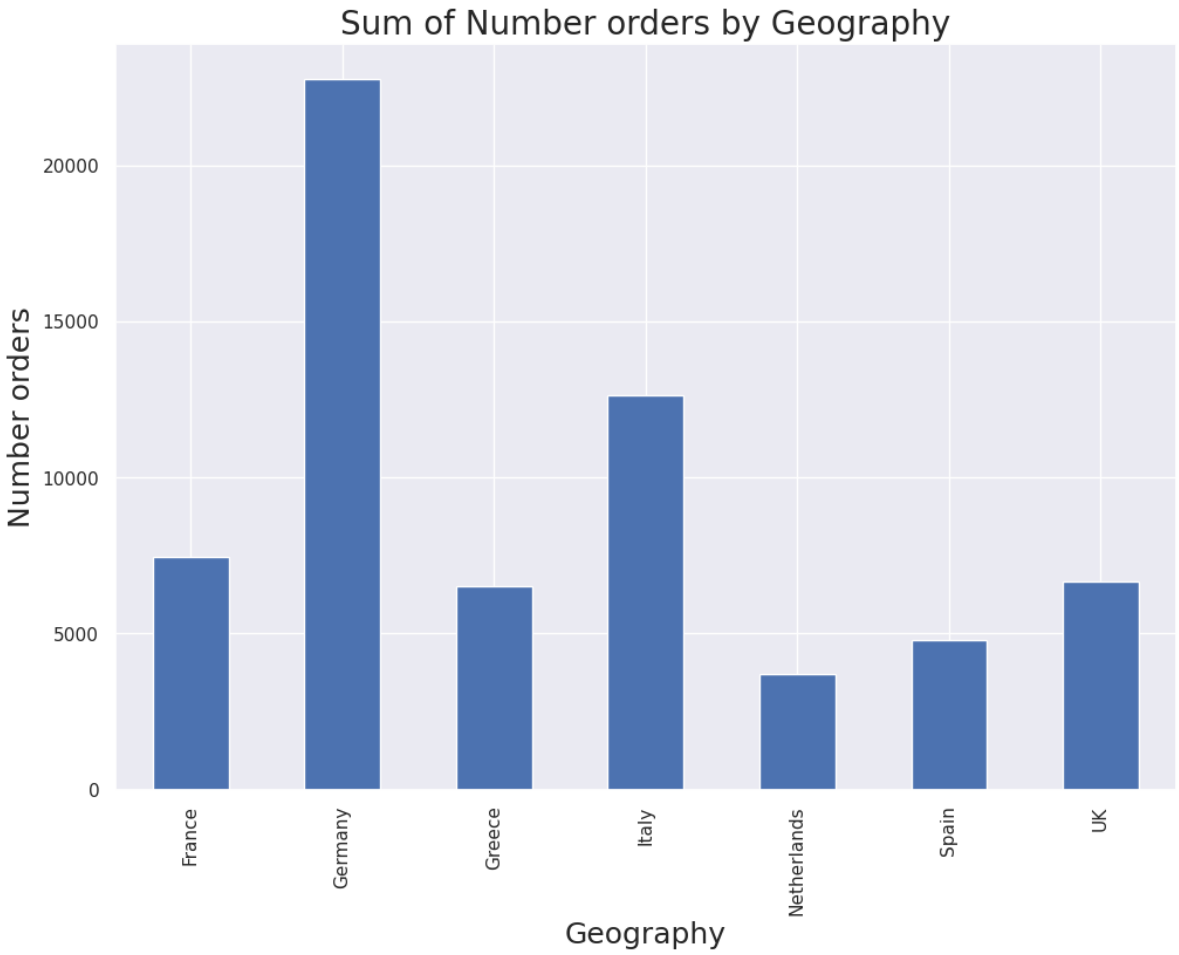

Sum of Diff items bought by Geography
Diff items bought
35000
30000
25000
20000
15000
10000
5000
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

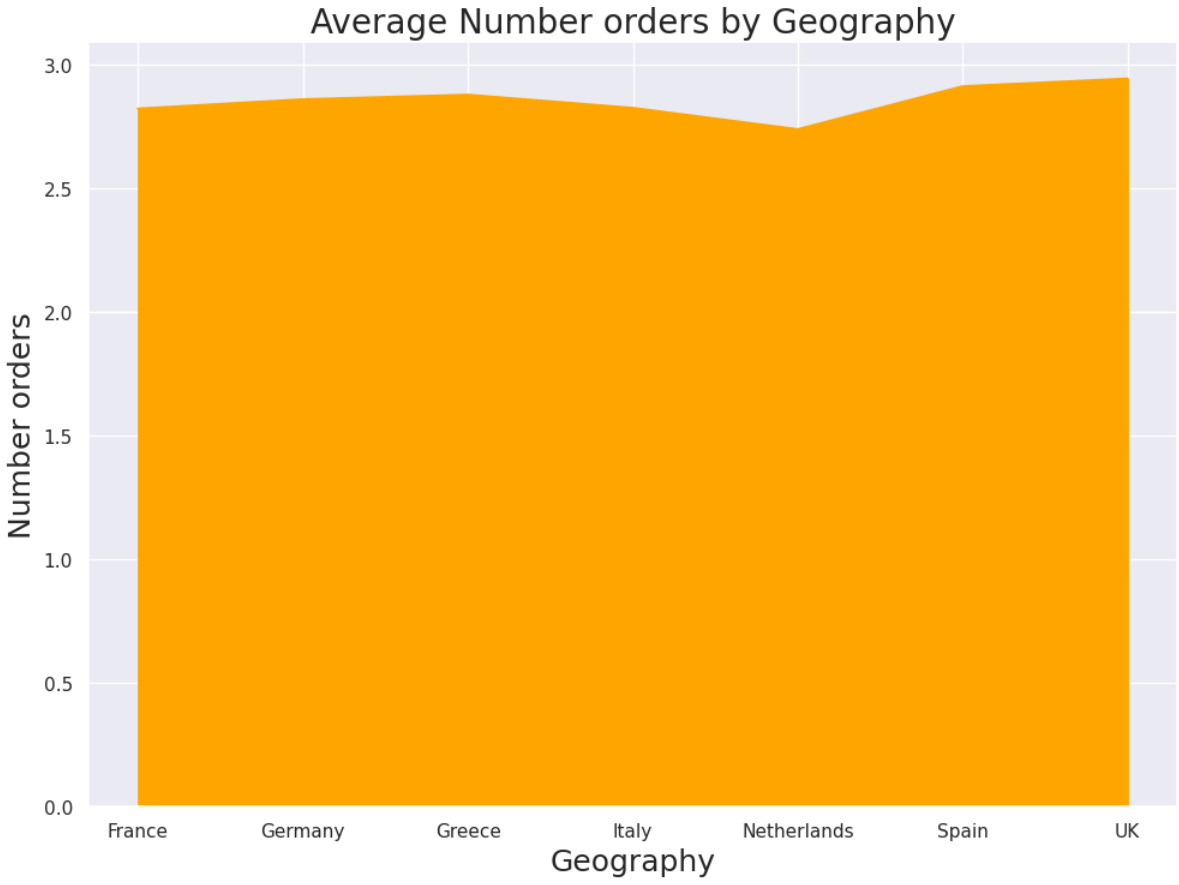

Average Diff items bought by Geography
Diff items bought
0
1
2
3
4
5
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

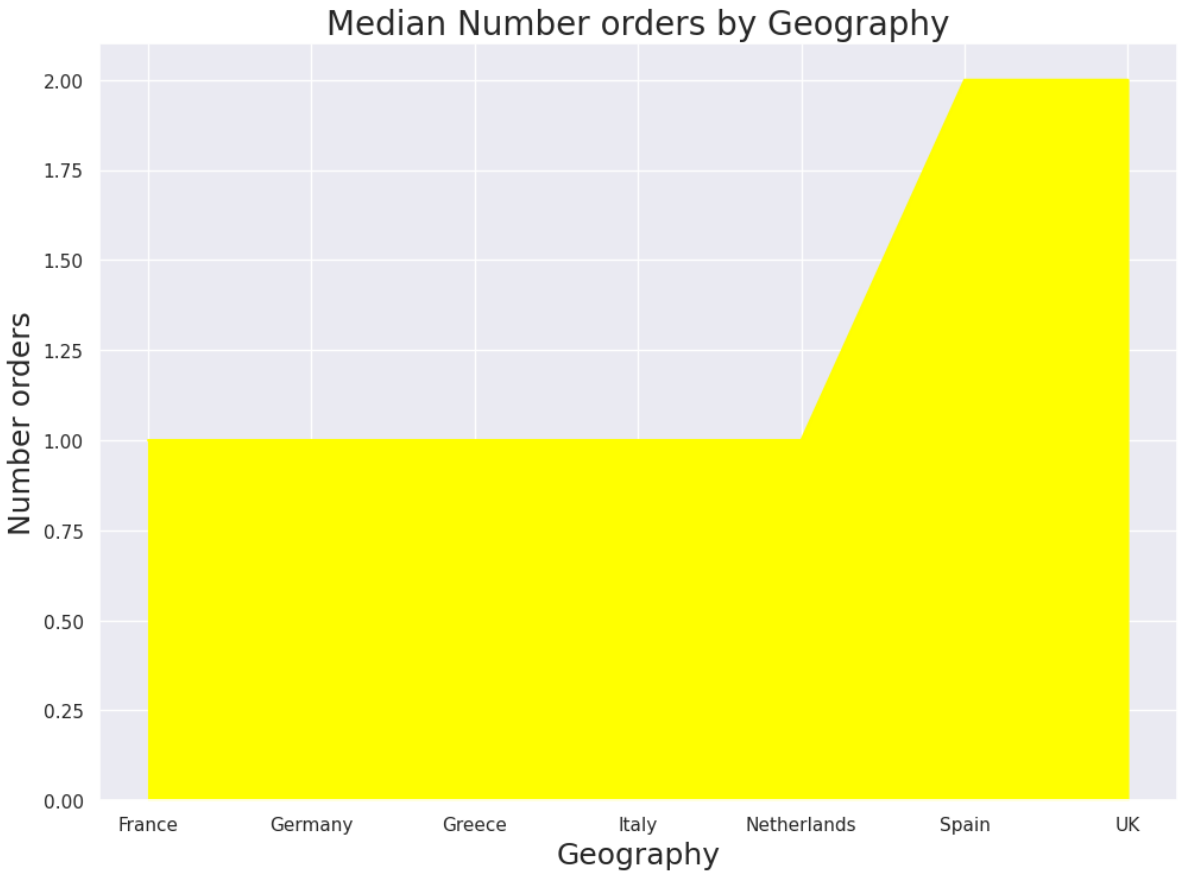

Median Diff items bought by Geography
Diff items bought
0.0
0.5
1.0
1.5
2.0
2.5
3.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

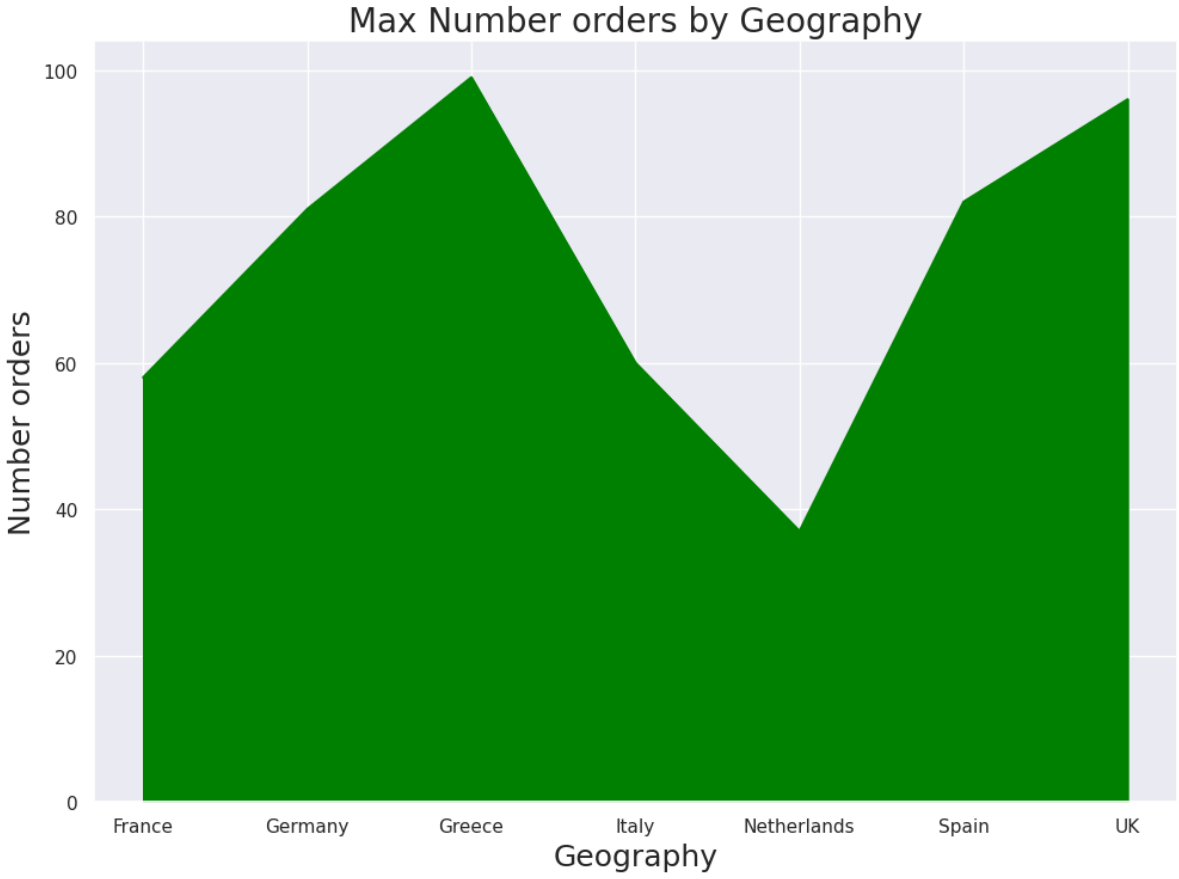

Max Diff items bought by Geography
Diff items bought
175
150
125
100
75
50
25
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

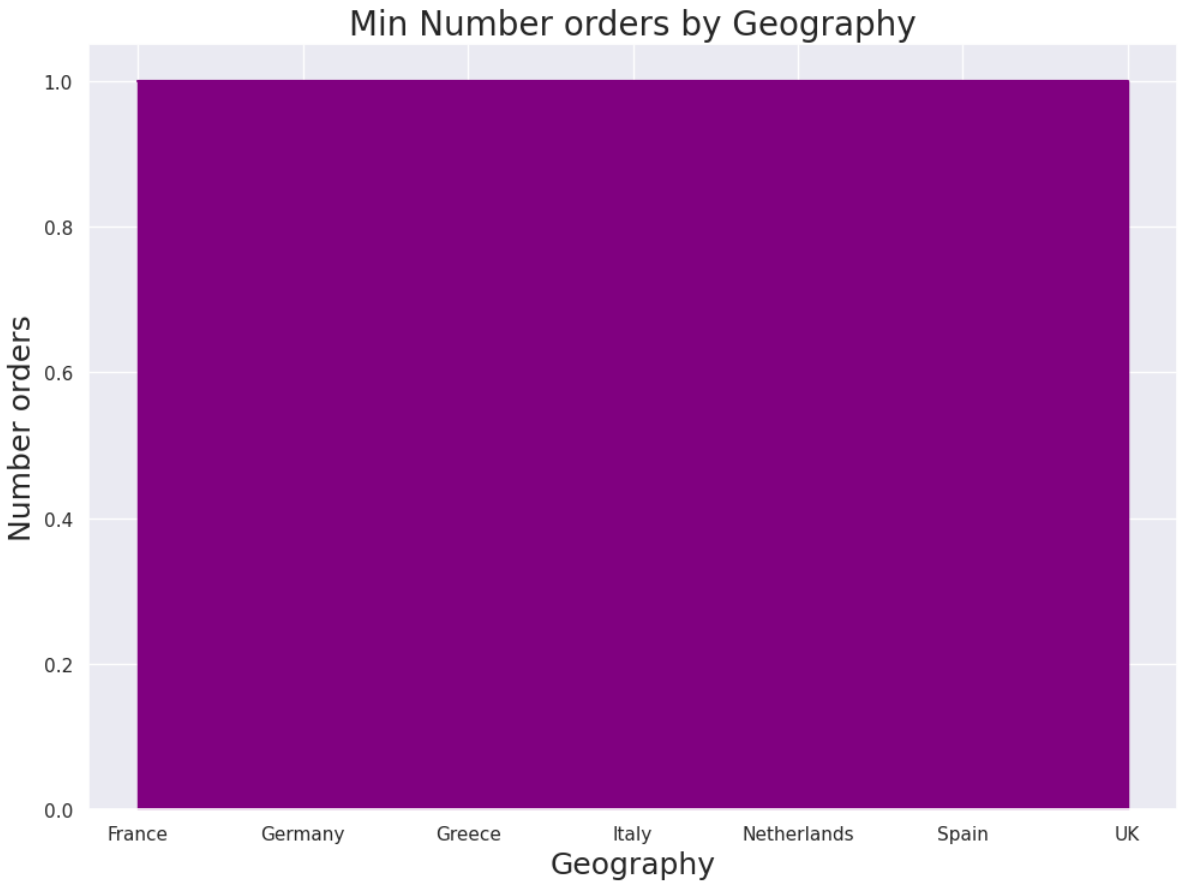

Min Diff items bought by Geography
Diff items bought
1.0
0.8
0.6
0.4
0.2
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

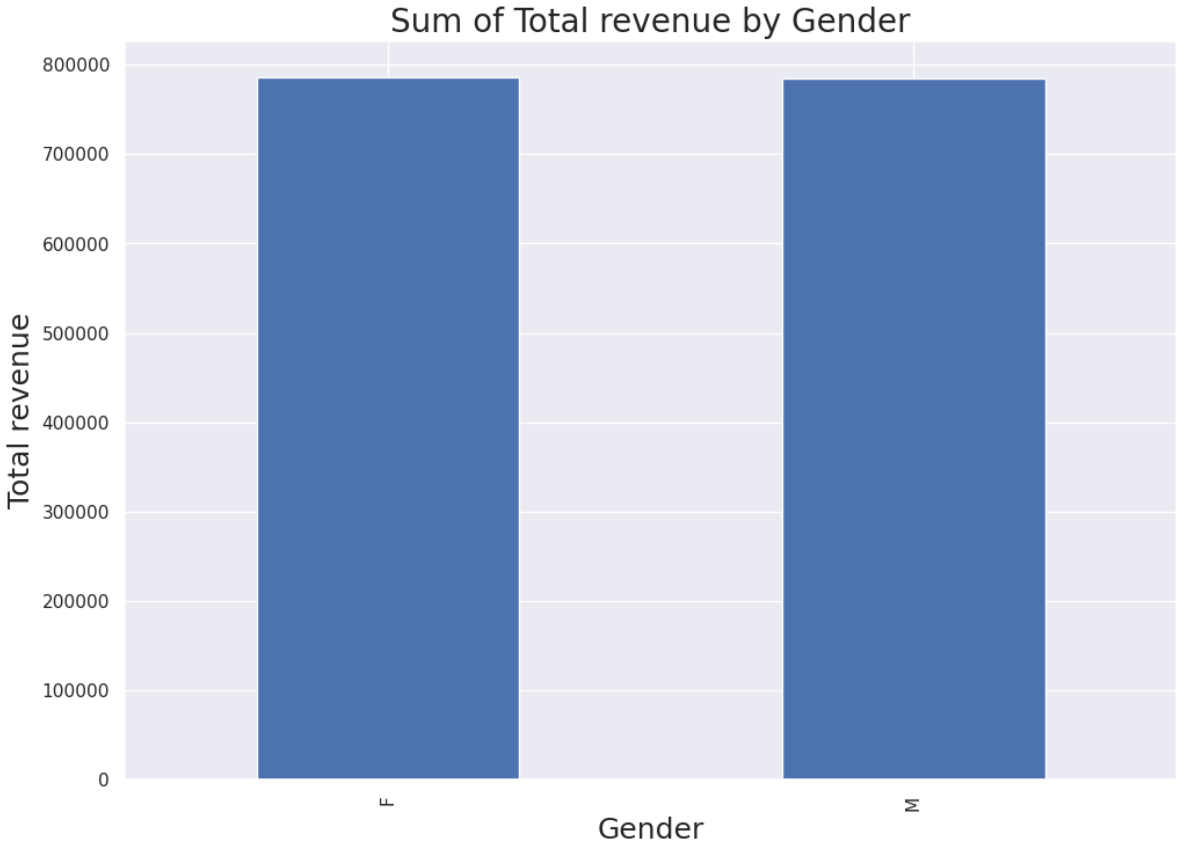

Sum of Number orders by Geography
Number orders
20000
15000
10000
5000
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

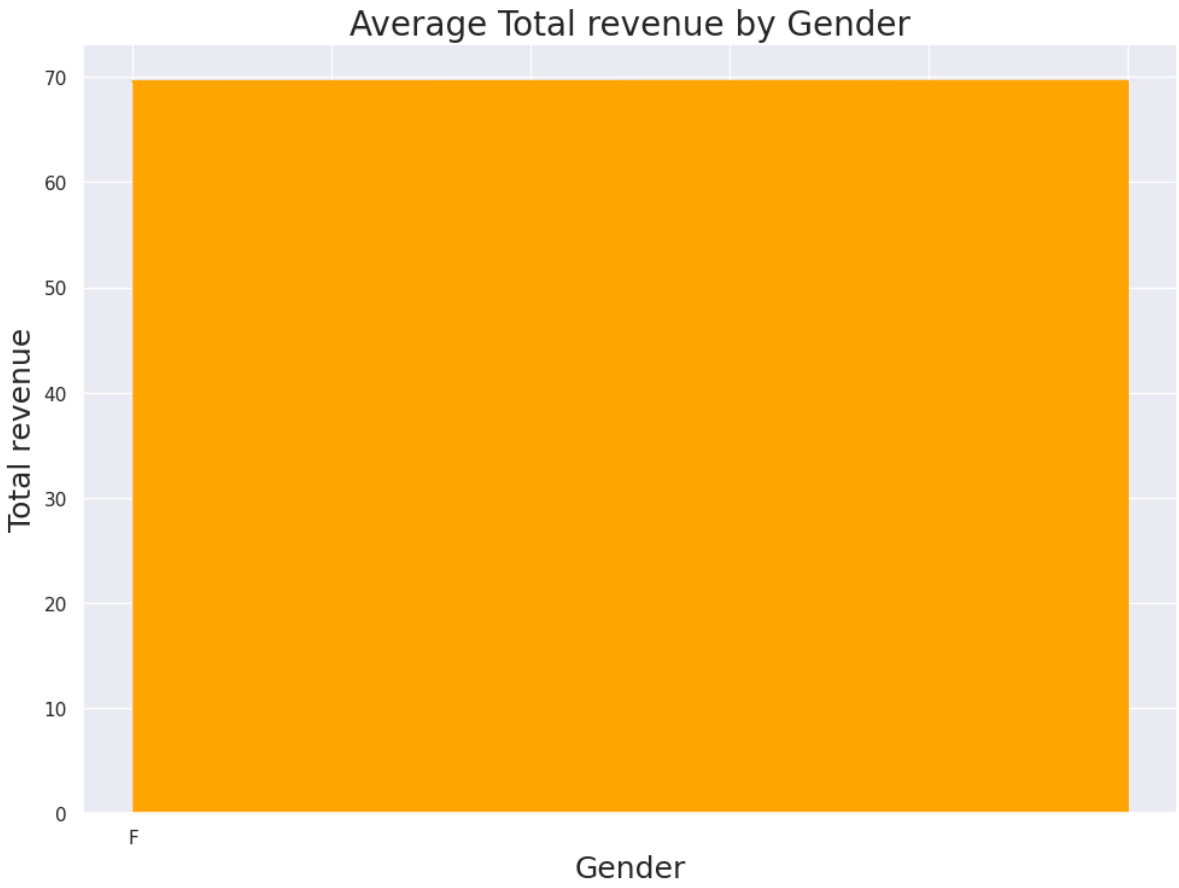

Average Number orders by Geography
Number orders
3.0
2.5
2.0
1.5
1.0
0.5
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

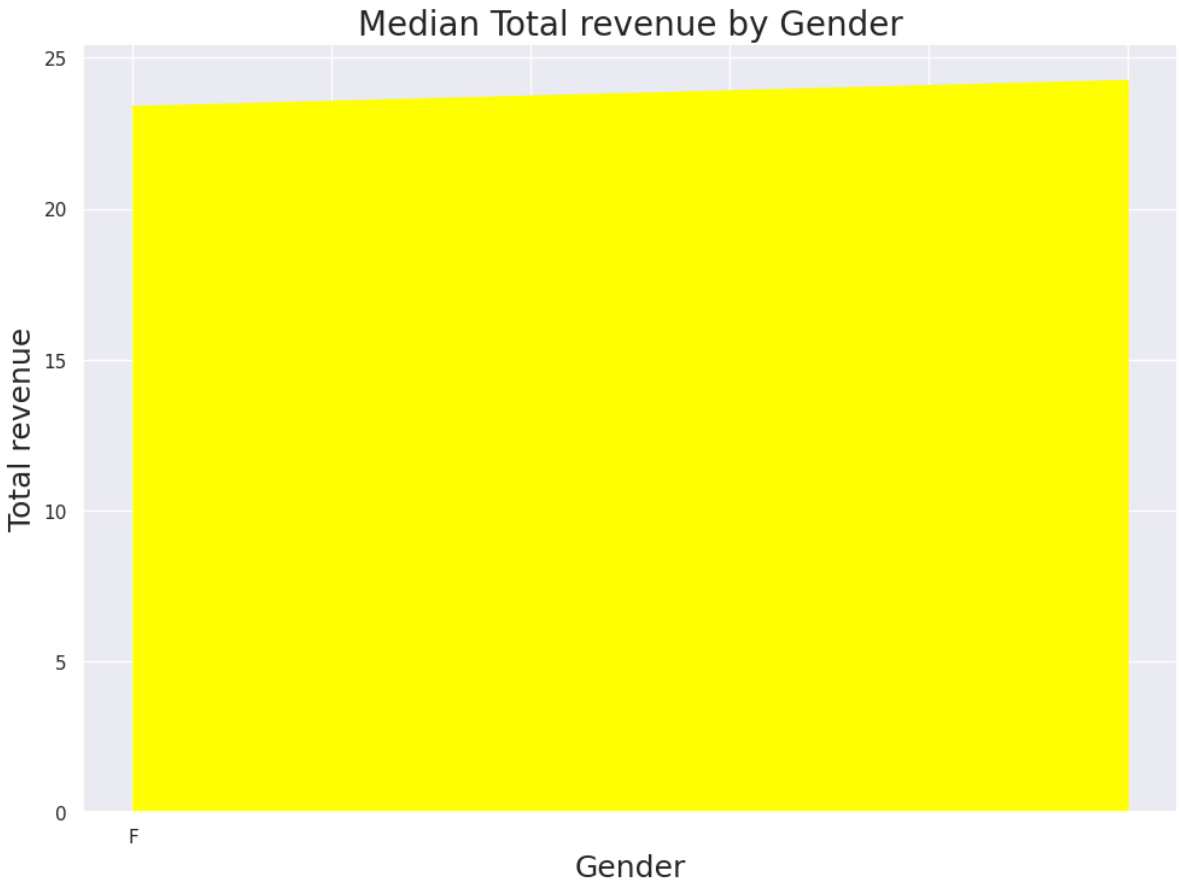

Median Number orders by Geography
Number orders
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

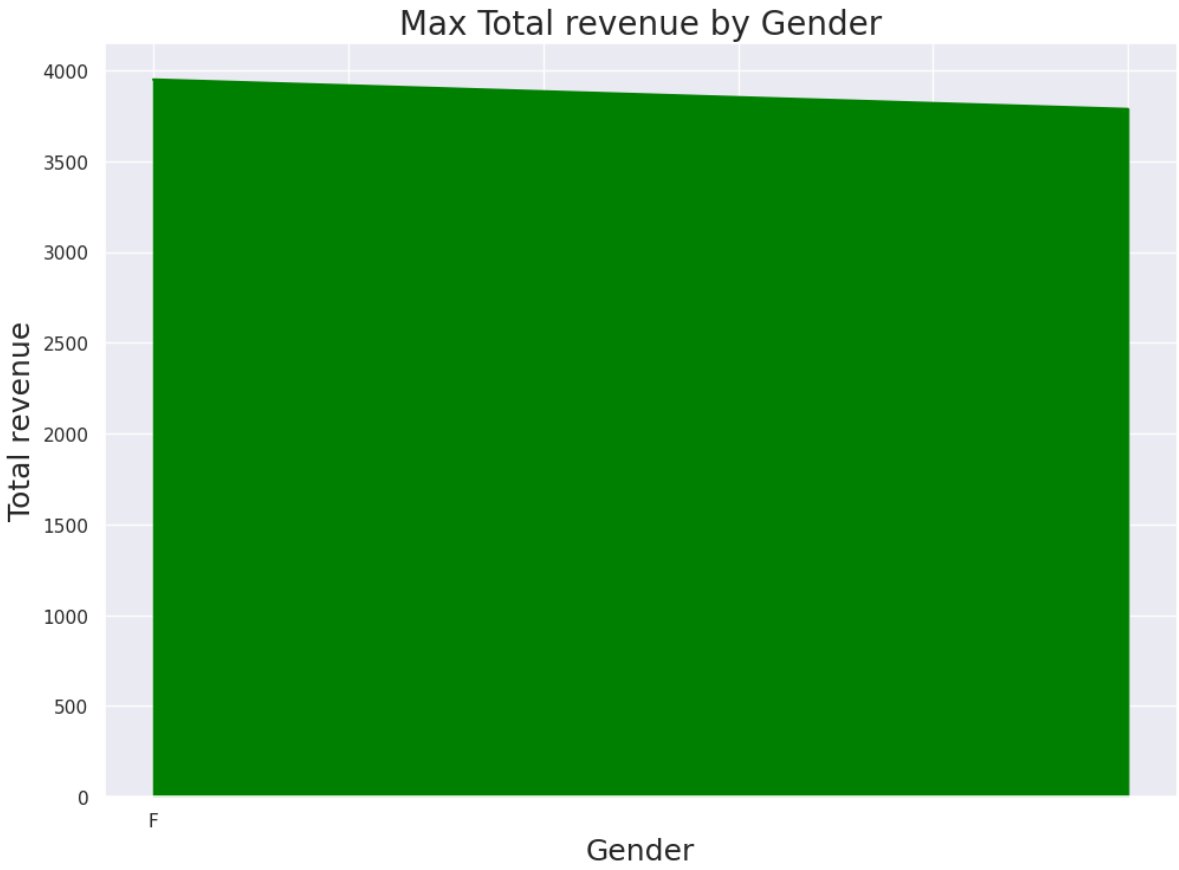

Max Number orders by Geography
Number orders
100
80
60
40
20
0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

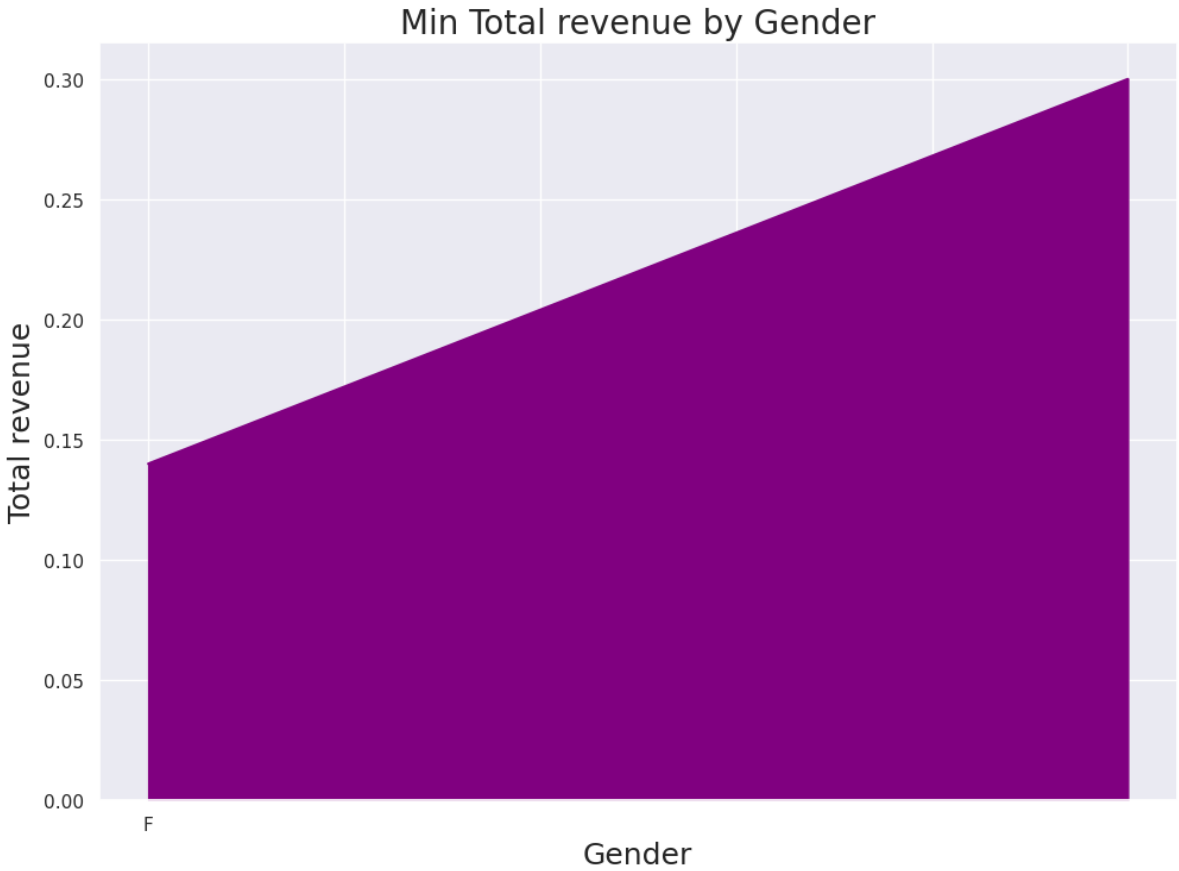

Min Number orders by Geography
Number orders
1.0
0.8
0.6
0.4
0.2
0.0
France
Germany
Greece
Italy
Netherlands
Spain
UK
Geography

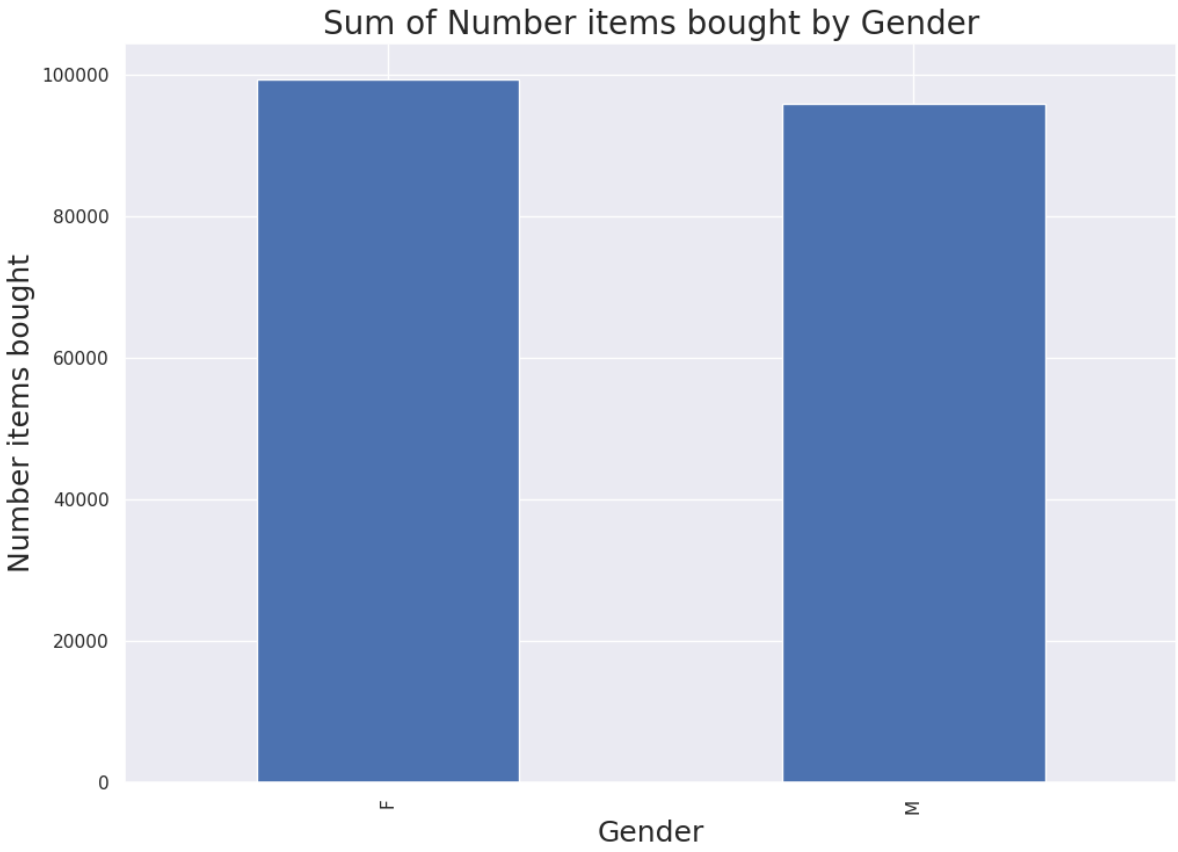

Sum of Total revenue by Gender
Total revenue
800000
700000
600000
500000
400000
300000
200000
100000
0
F
M
Gender


Average Total revenue by Gender
Total revenue
70
60
50
40
30
20
10
0
F
Gender

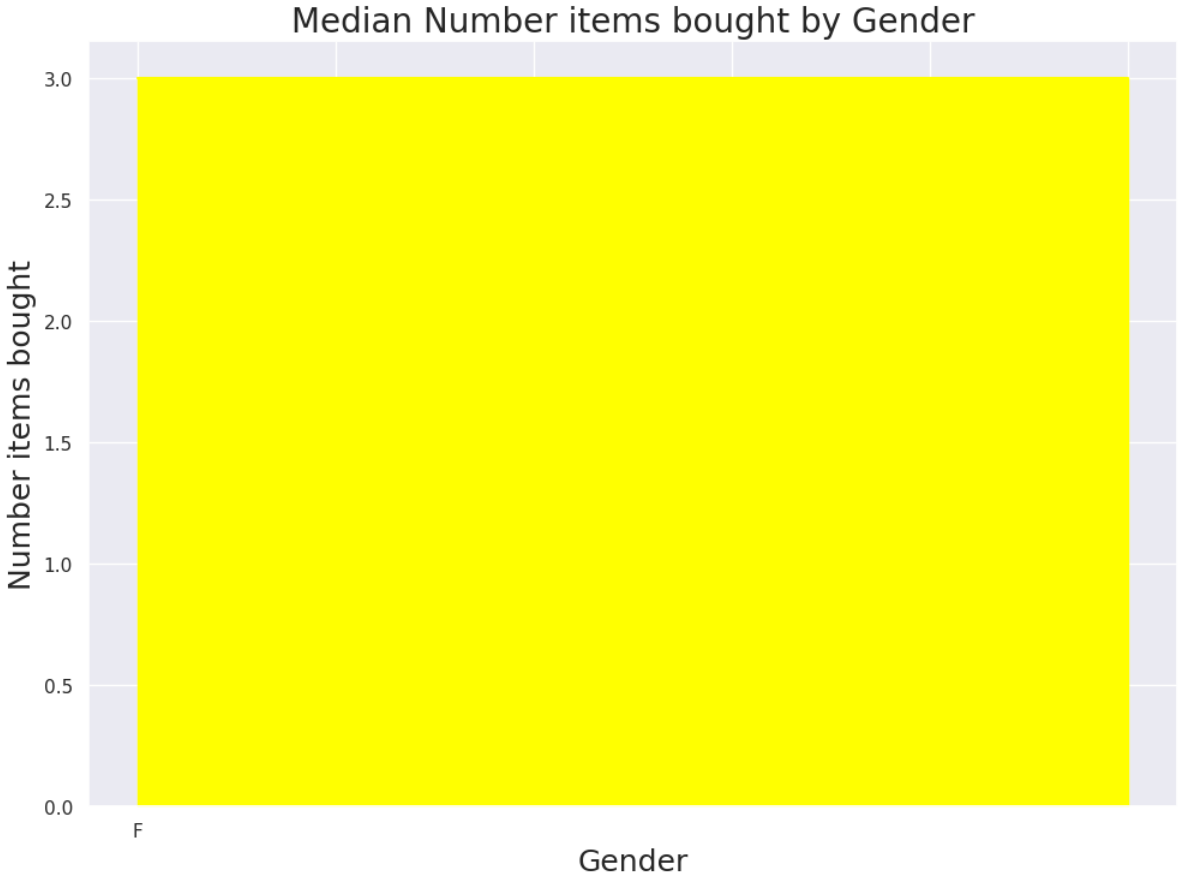

Median Total revenue by Gender
Total revenue
25
20
15
10
5
0
F
Gender

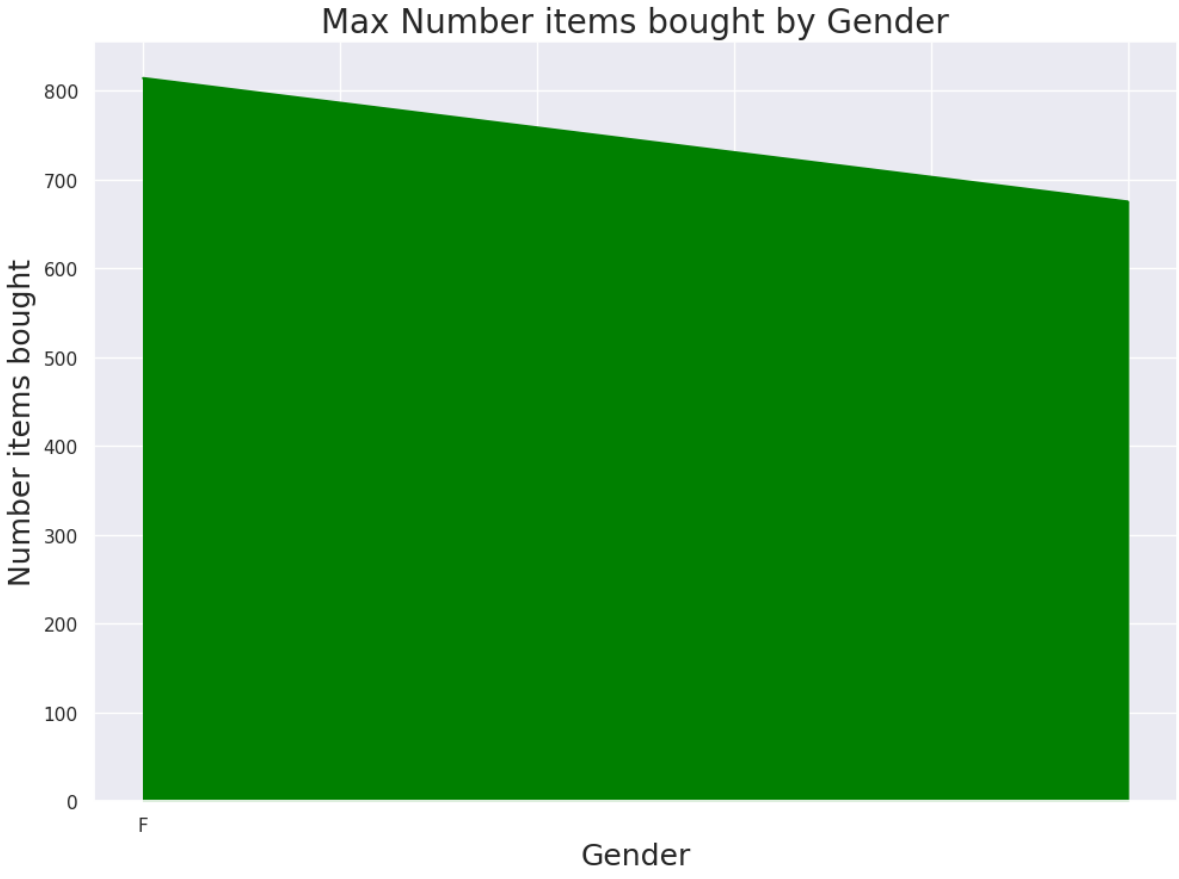

Max Total revenue by Gender
Total revenue
4000
3500
3000
2500
2000
1500
1000
500
0
F
Gender


Min Total revenue by Gender
Total revenue
0.30
0.25
0.20
0.15
0.10
0.05
0.00
F
Gender

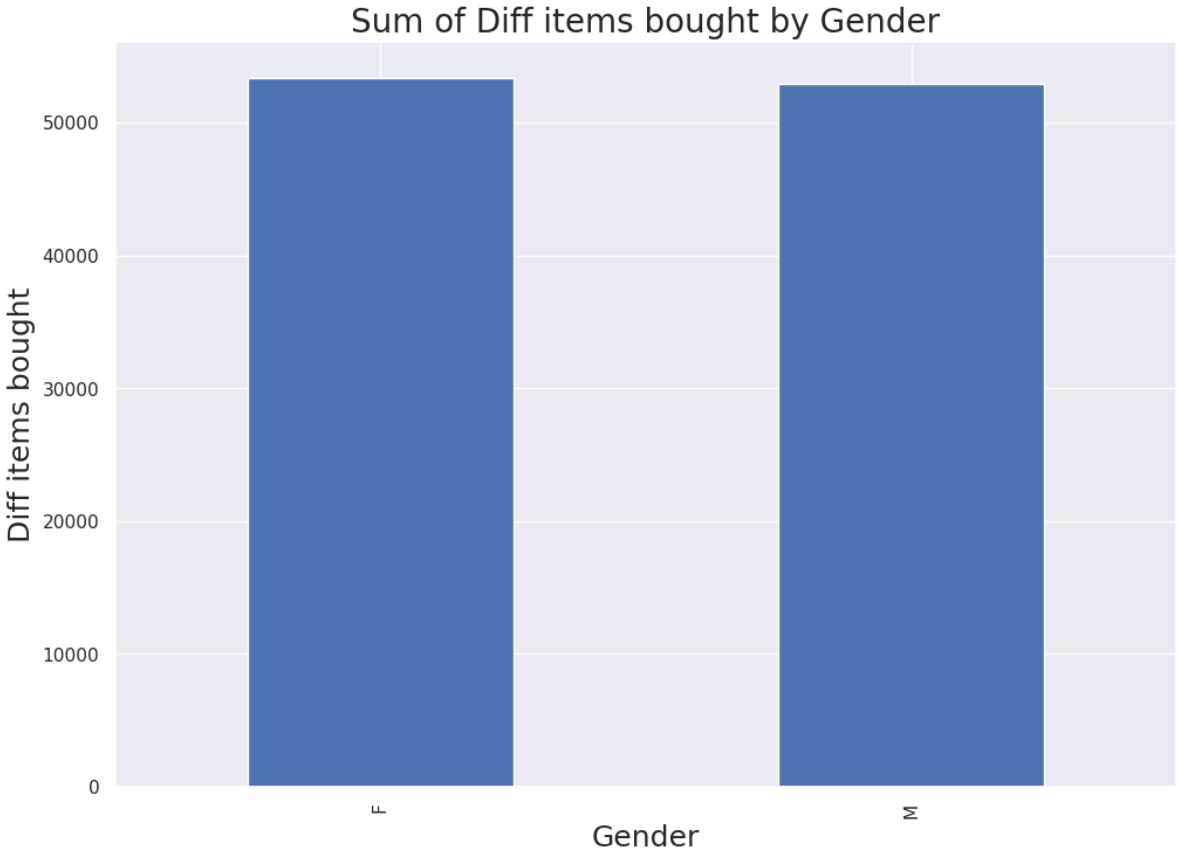

Sum of Number items bought by Gender
Number items bought
100000
80000
60000
40000
20000
0
F
M
Gender


Average Number items bought by Gender
Number items bought
8
6
4
2
0
F
Gender

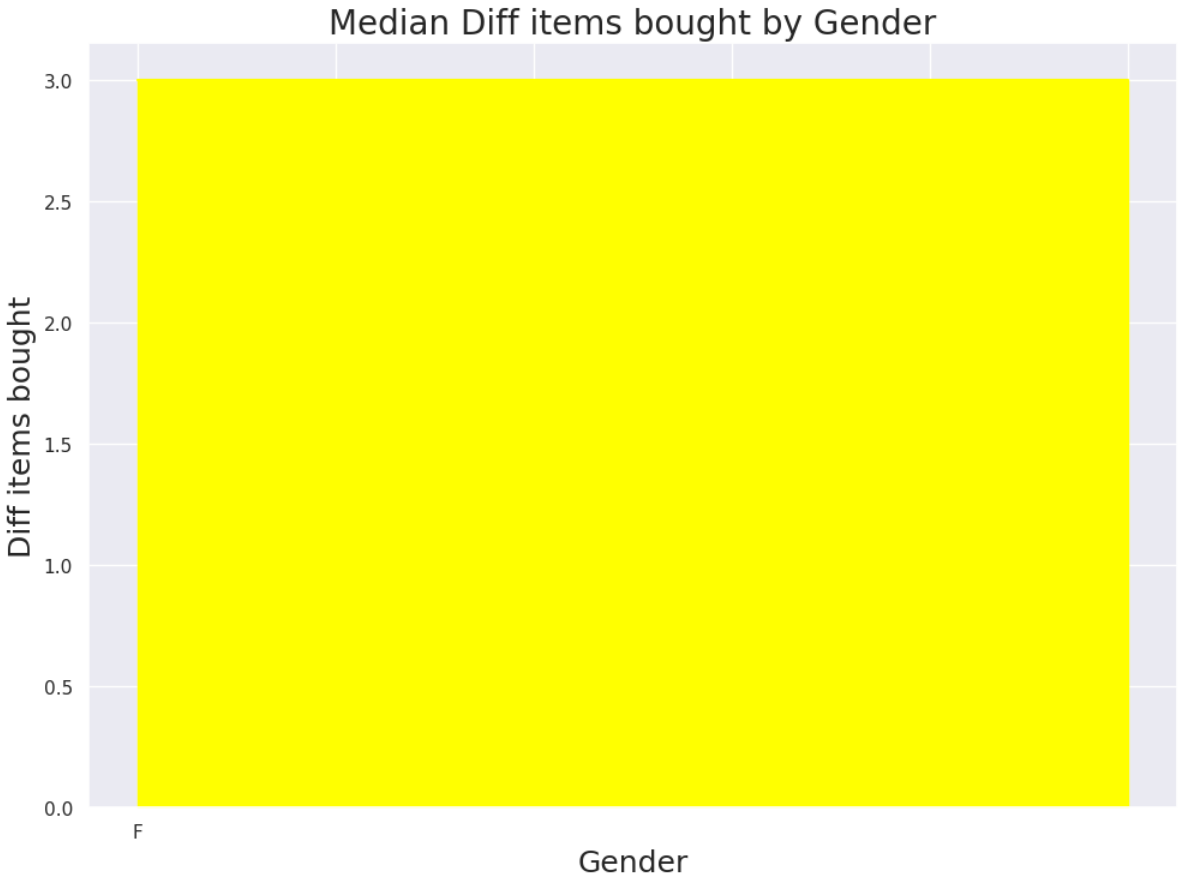

Median Number items bought by Gender
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Number items bought
F
Gender

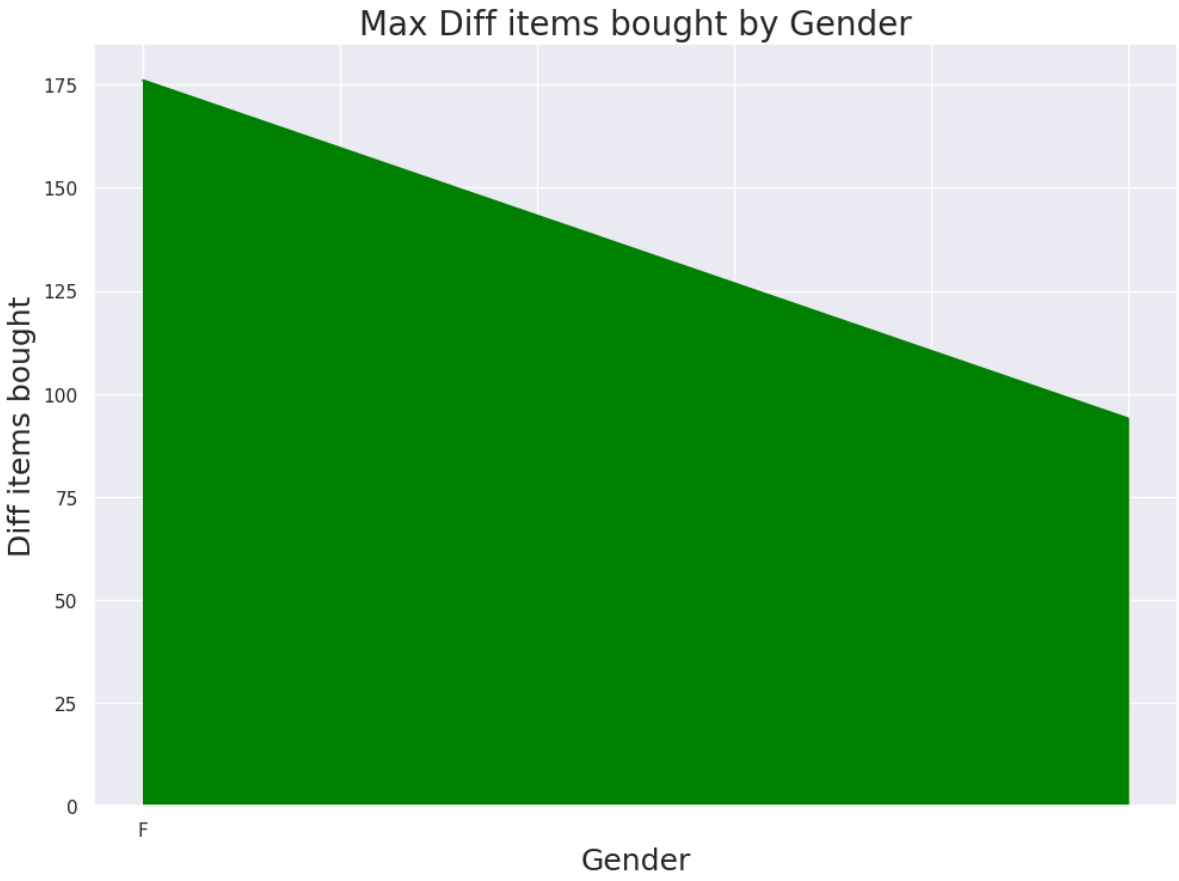

Max Number items bought by Gender
800
700
600
500
400
300
200
100
0
Number items bought
F
Gender


Min Number items bought by Gender
Number items bought
1.0
0.8
0.6
0.4
0.2
0.0
F
Gender

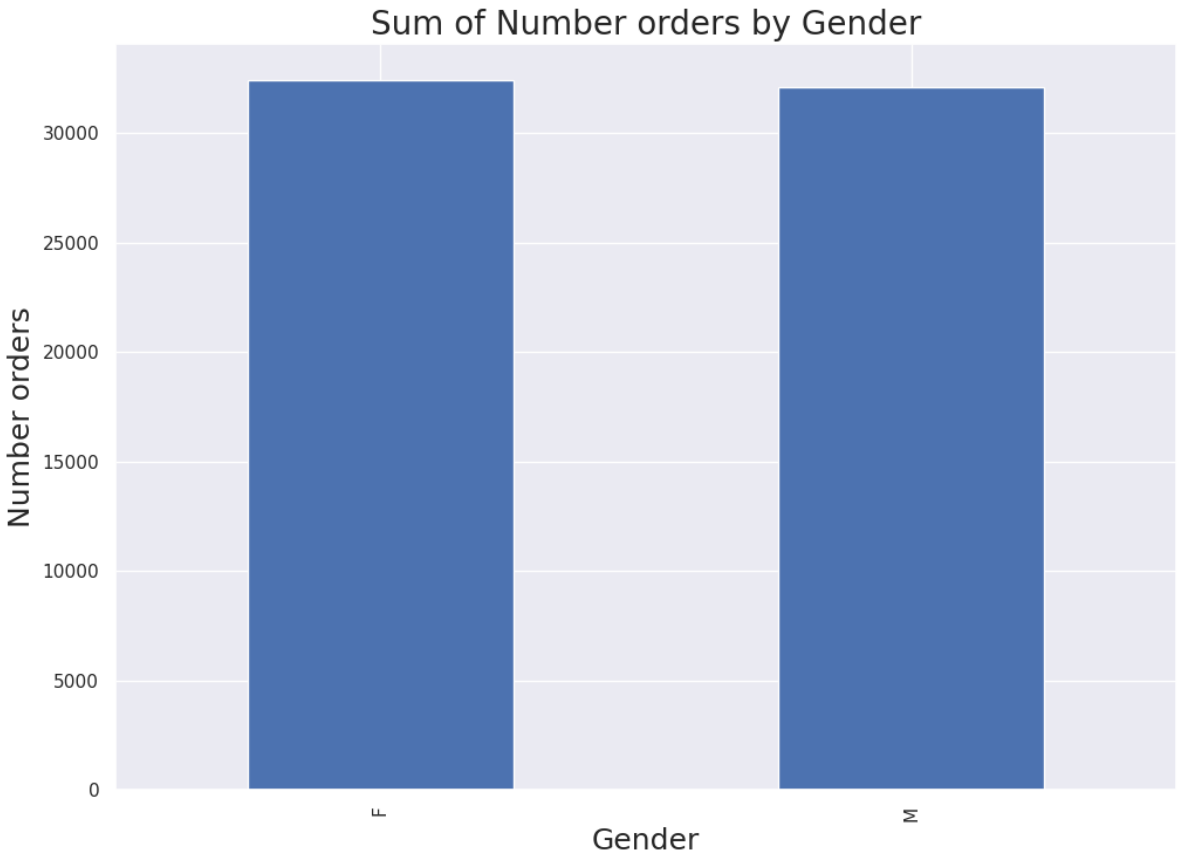

Sum of Diff items bought by Gender
Diff items bought
50000
40000
30000
20000
10000
0
F
M
Gender

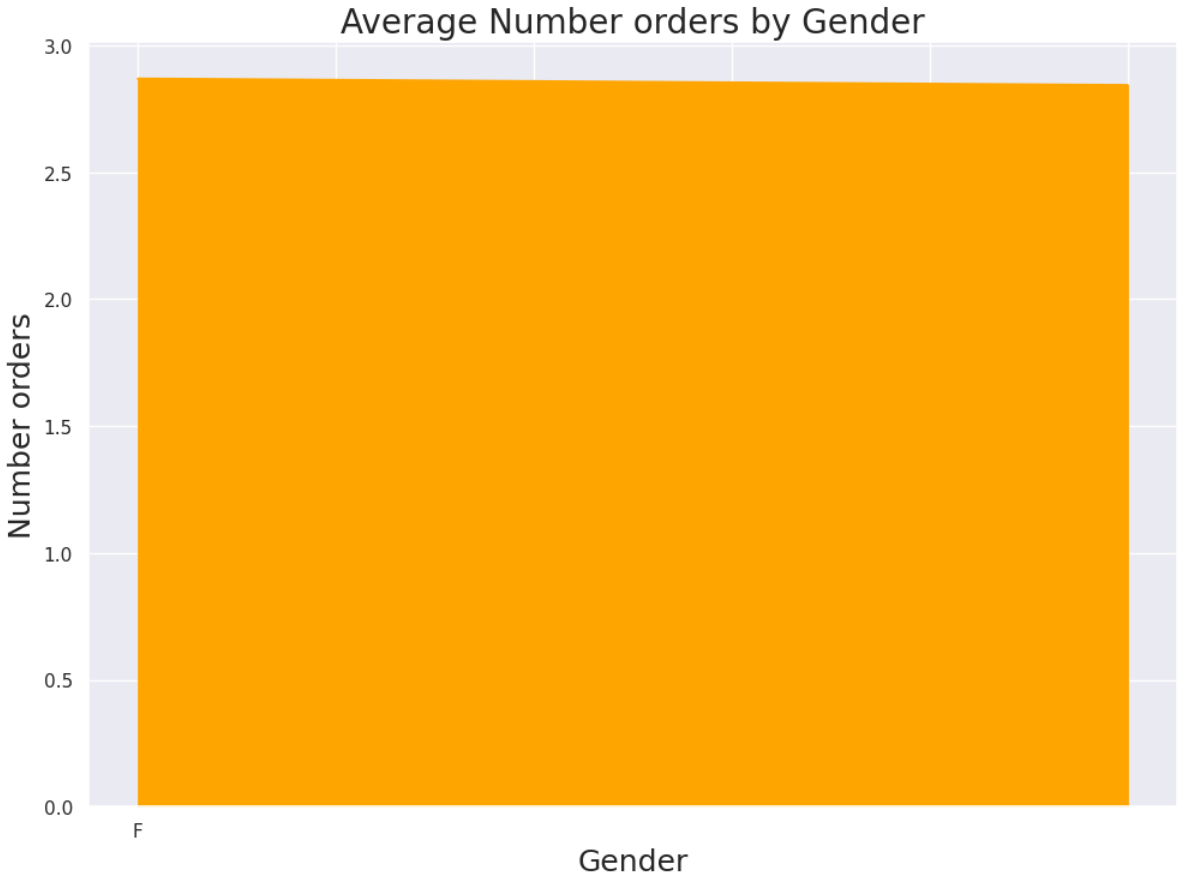

Average Diff items bought by Gender
Diff items bought
0
1
2
3
4
F
Gender

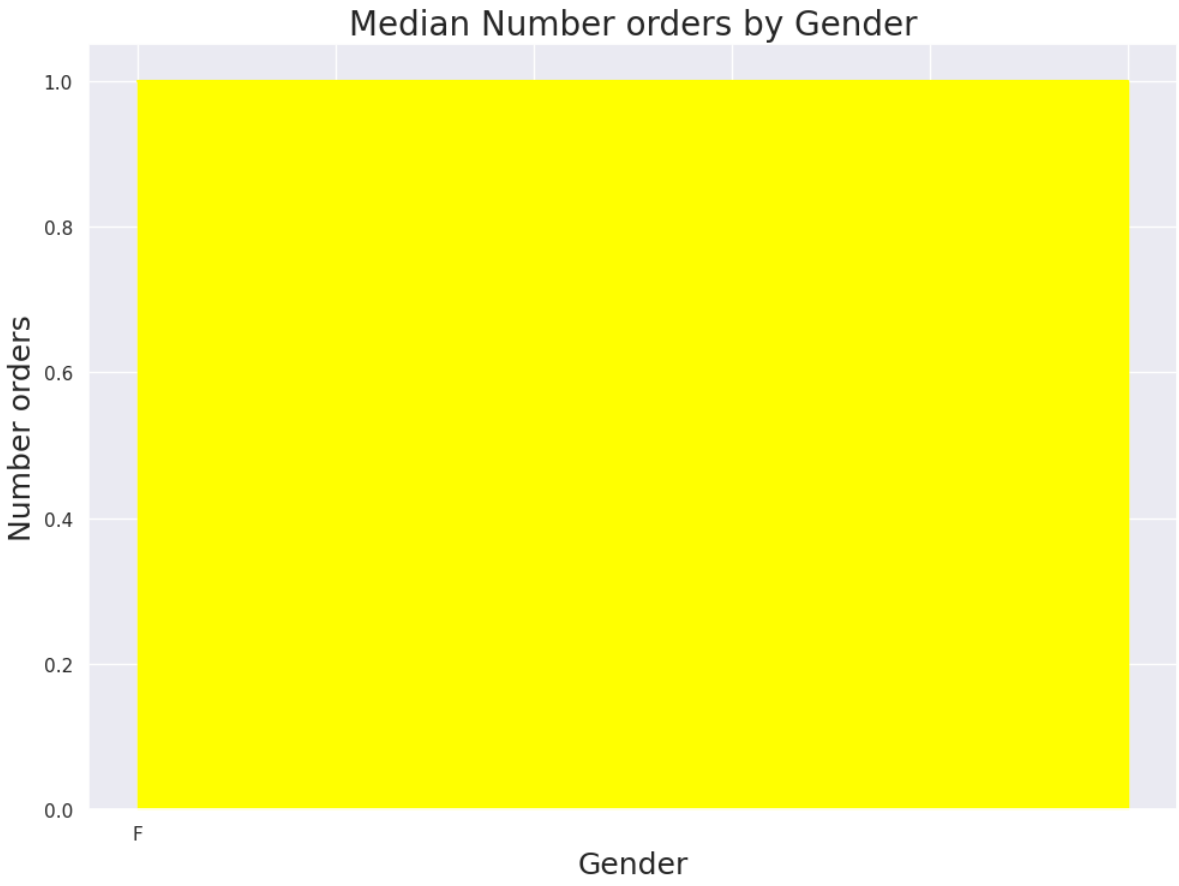

Median Diff items bought by Gender
Diff items bought
0.0
0.5
1.0
1.5
2.0
2.5
3.0
F
Gender

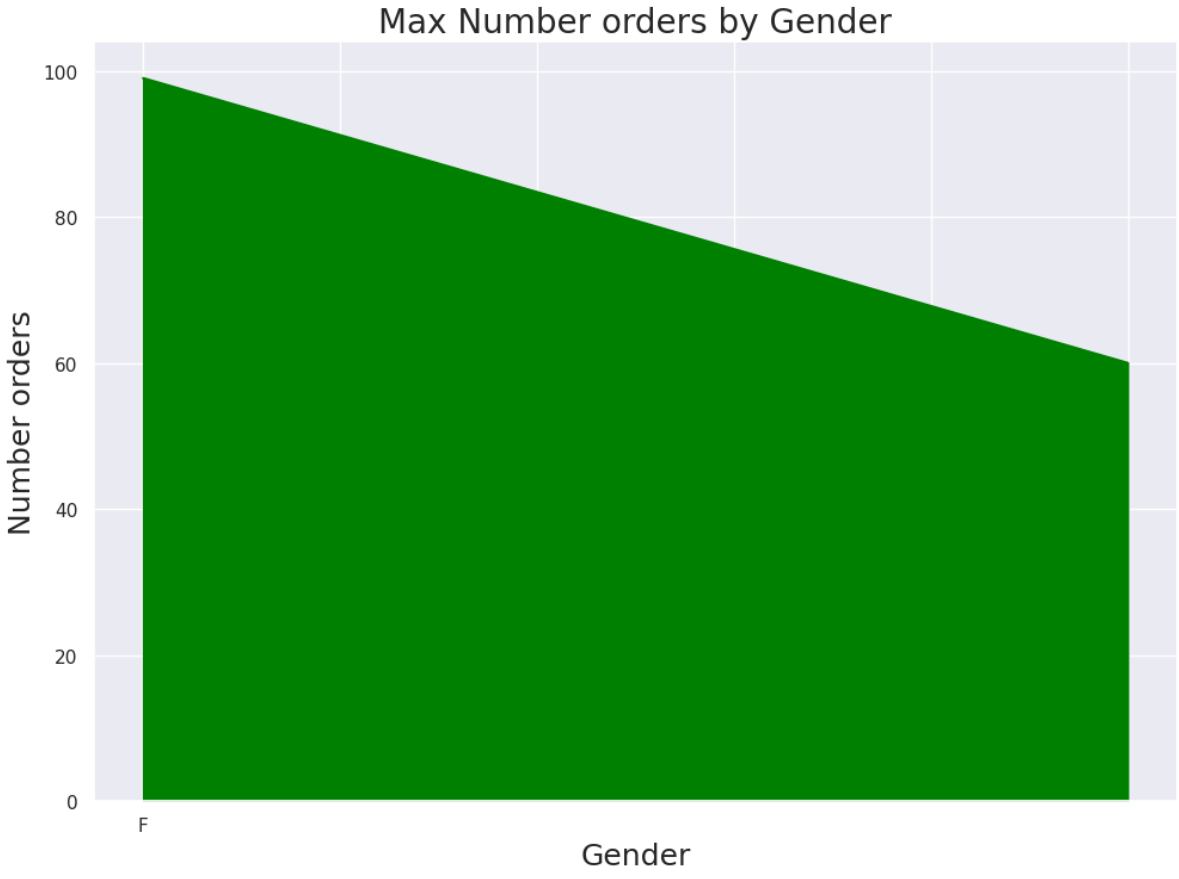

Max Diff items bought by Gender
175
150
125
100
75
50
25
0
Diff items bought
F
Gender


Min Diff items bought by Gender
1.0
0.8
0.6
0.4
0.2
0.0
Diff items bought
F
Gender

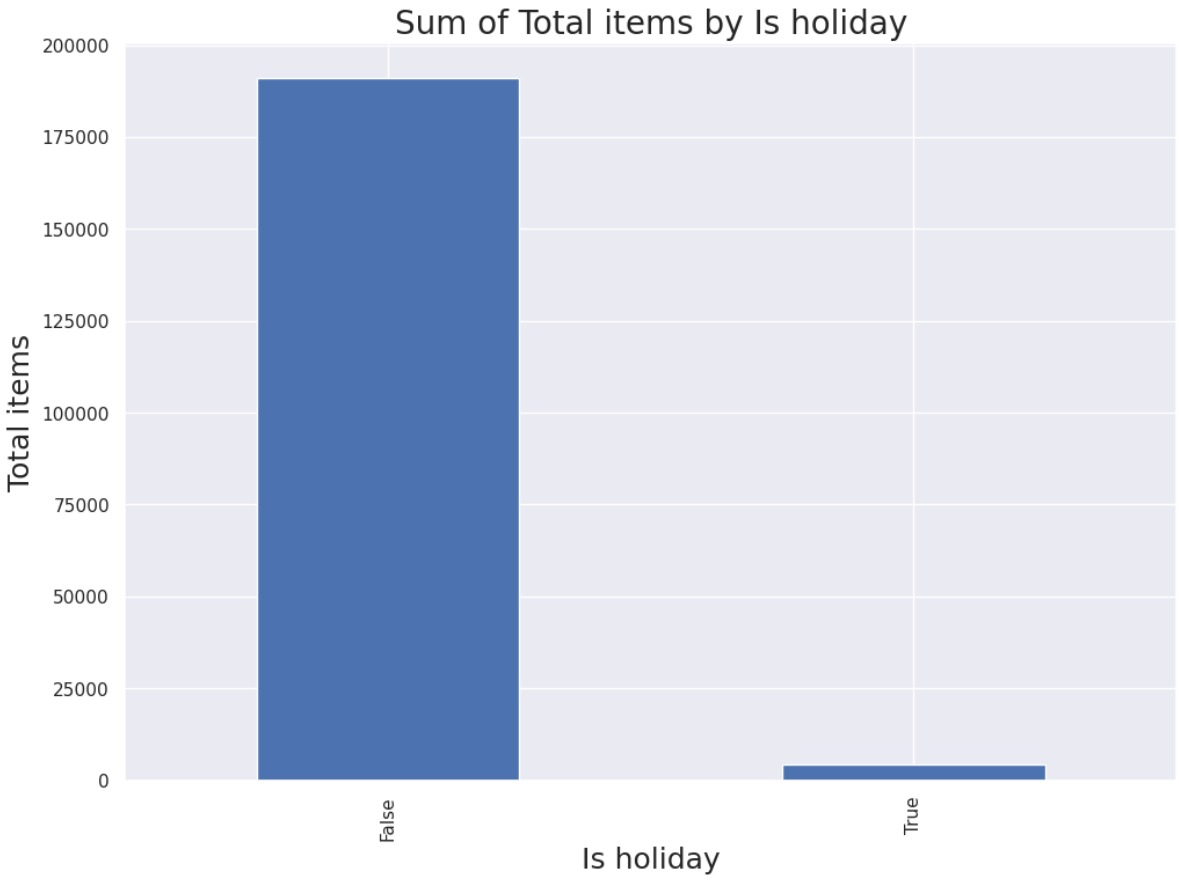

Sum of Number orders by Gender
Number orders
30000
25000
20000
15000
10000
5000
0
F
M
Gender


Average Number orders by Gender
Number orders
3.0
2.5
2.0
1.5
1.0
0.5
0.0
F
Gender


Median Number orders by Gender
1.0
0.8
0.6
0.4
0.2
0.0
Number orders
F
Gender

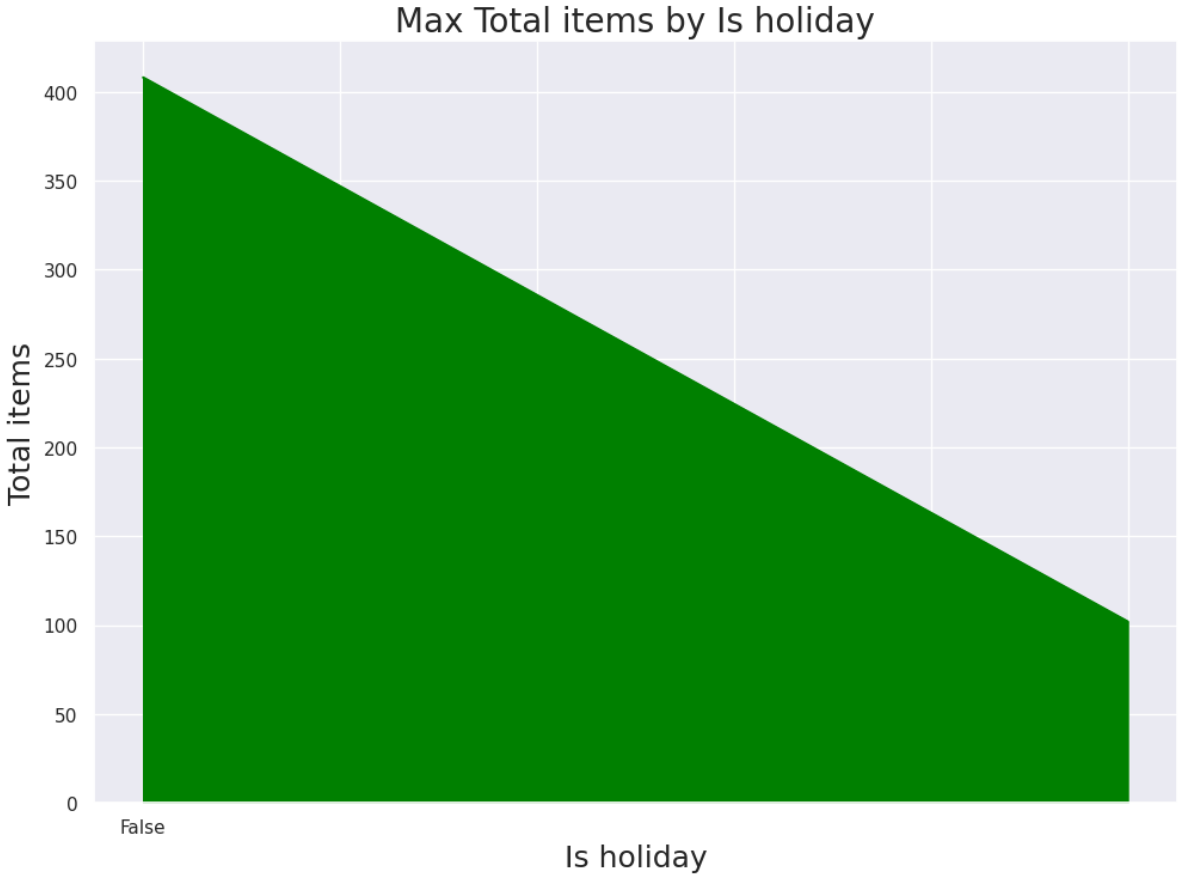

Max Number orders by Gender
100
80
60
40
20
0
Number orders
F
Gender

In [ ]:
```
plot_statistical_data(transactions, ['is_holiday'], geo_sum_cols)
```




Average Total items by Is holiday
Total items
3.0
2.5
2.0
1.5
1.0
0.5
0.0
False
Is holiday

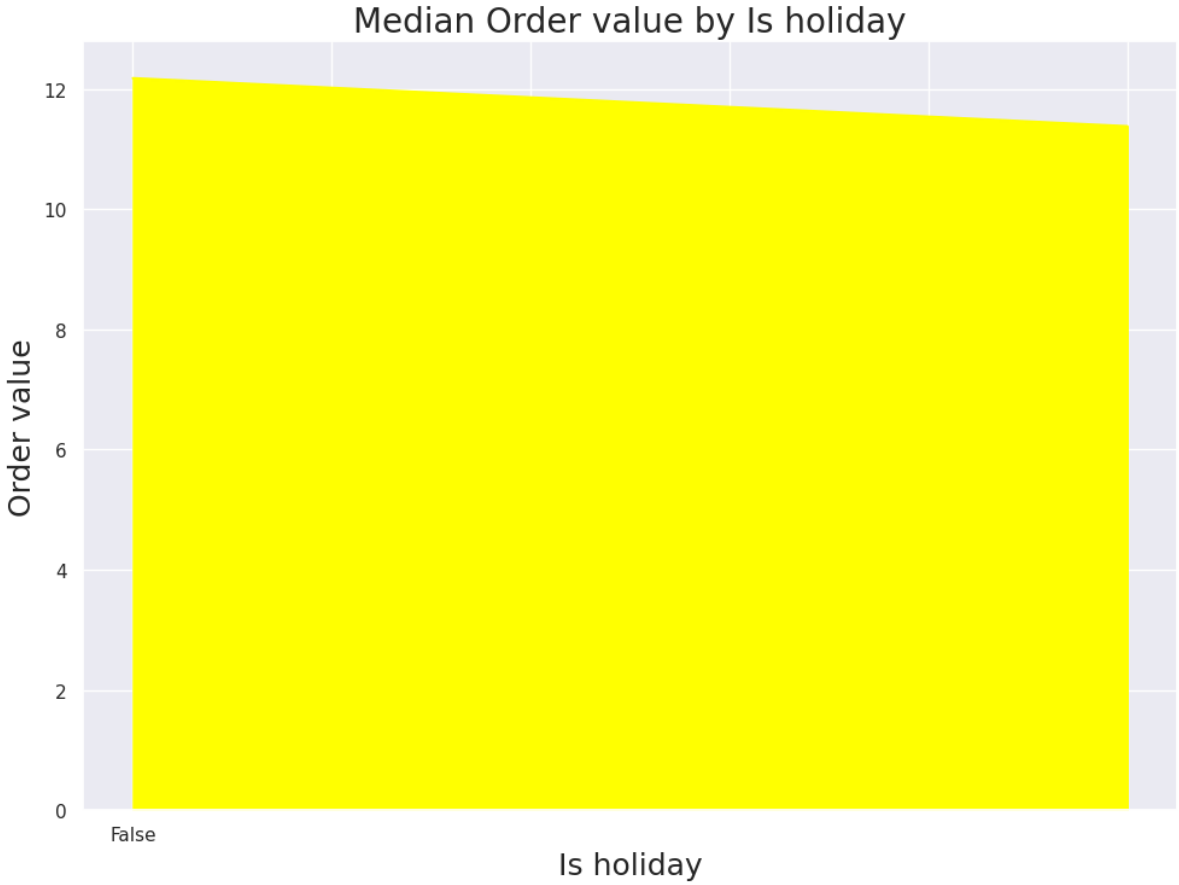

Median Total items by Is holiday
Total items
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
False
Is holiday

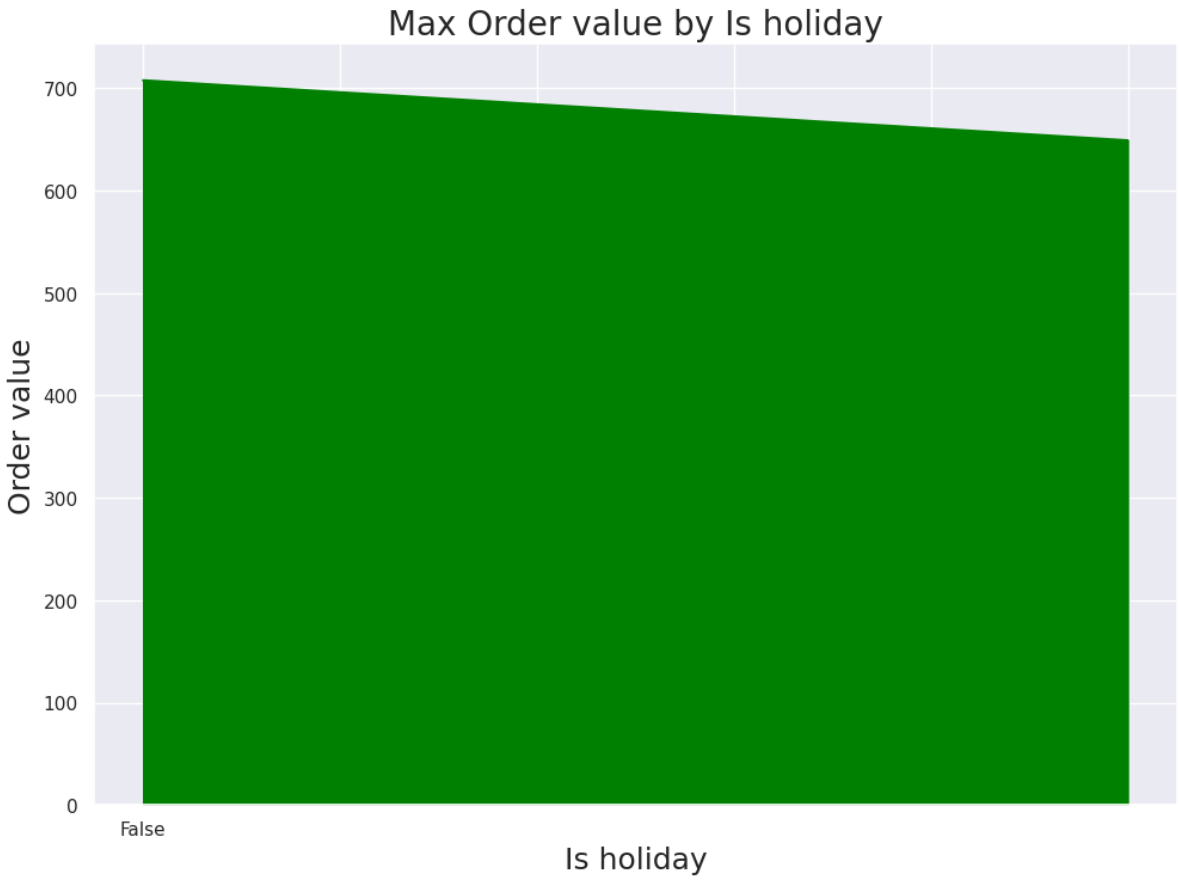

Max Total items by Is holiday
Total items
400
350
300
250
200
150
100
50
0
False
Is holiday

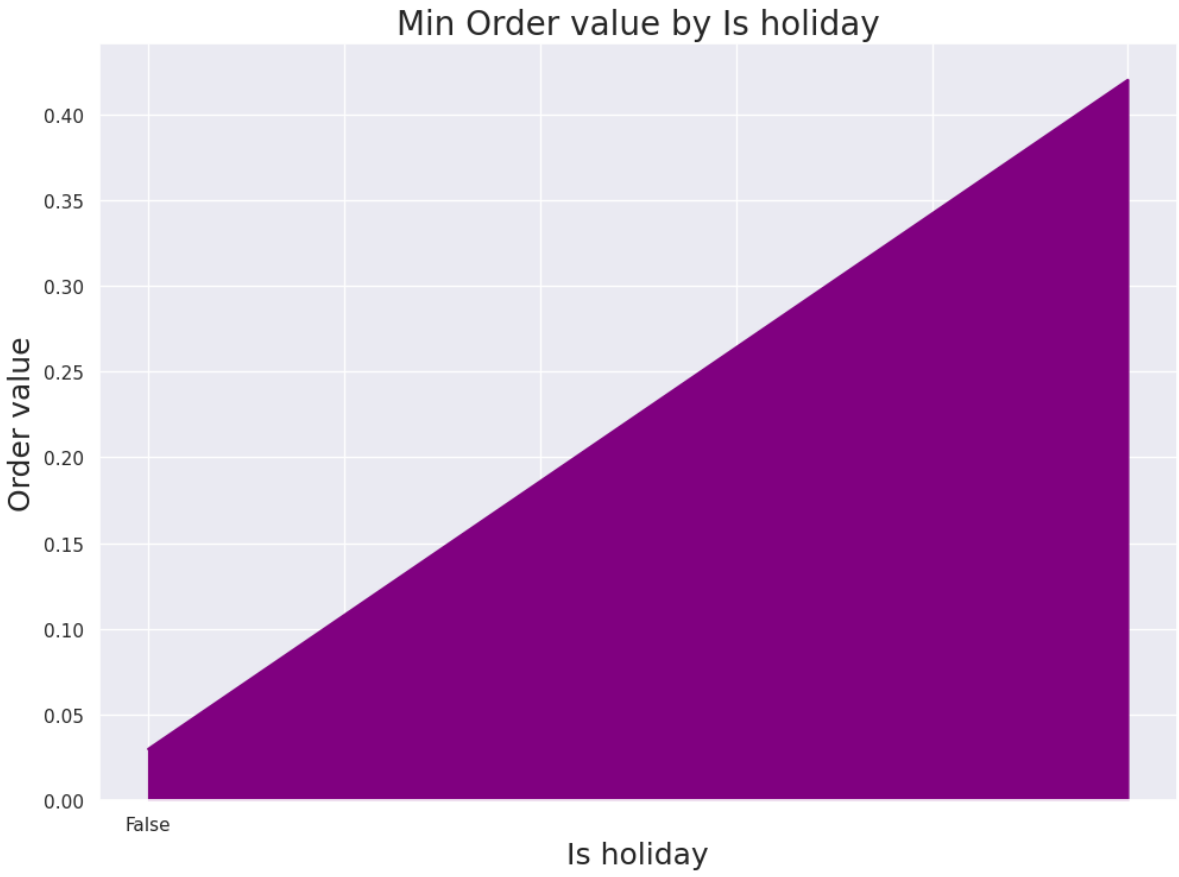

Min Total items by Is holiday
Total items
1.0
0.8
0.6
0.4
0.2
0.0
False
Is holiday


Sum of Order value by Is holiday
1e6
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Order value
False
True
Is holiday


Average Order value by Is holiday
25
20
15
10
5
0
Order value
False
Is holiday

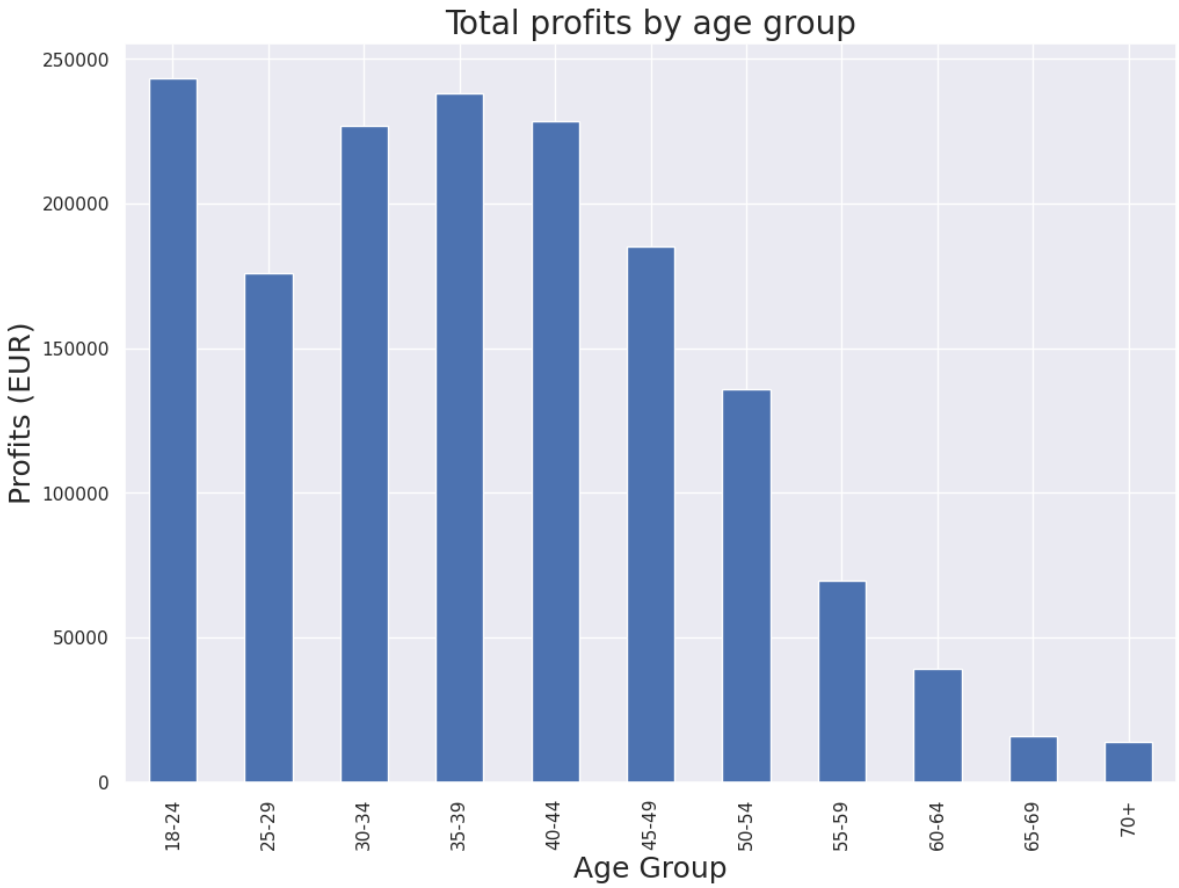

Median Order value by Is holiday
Order value
0
2
4
6
8
10
12
False
Is holiday

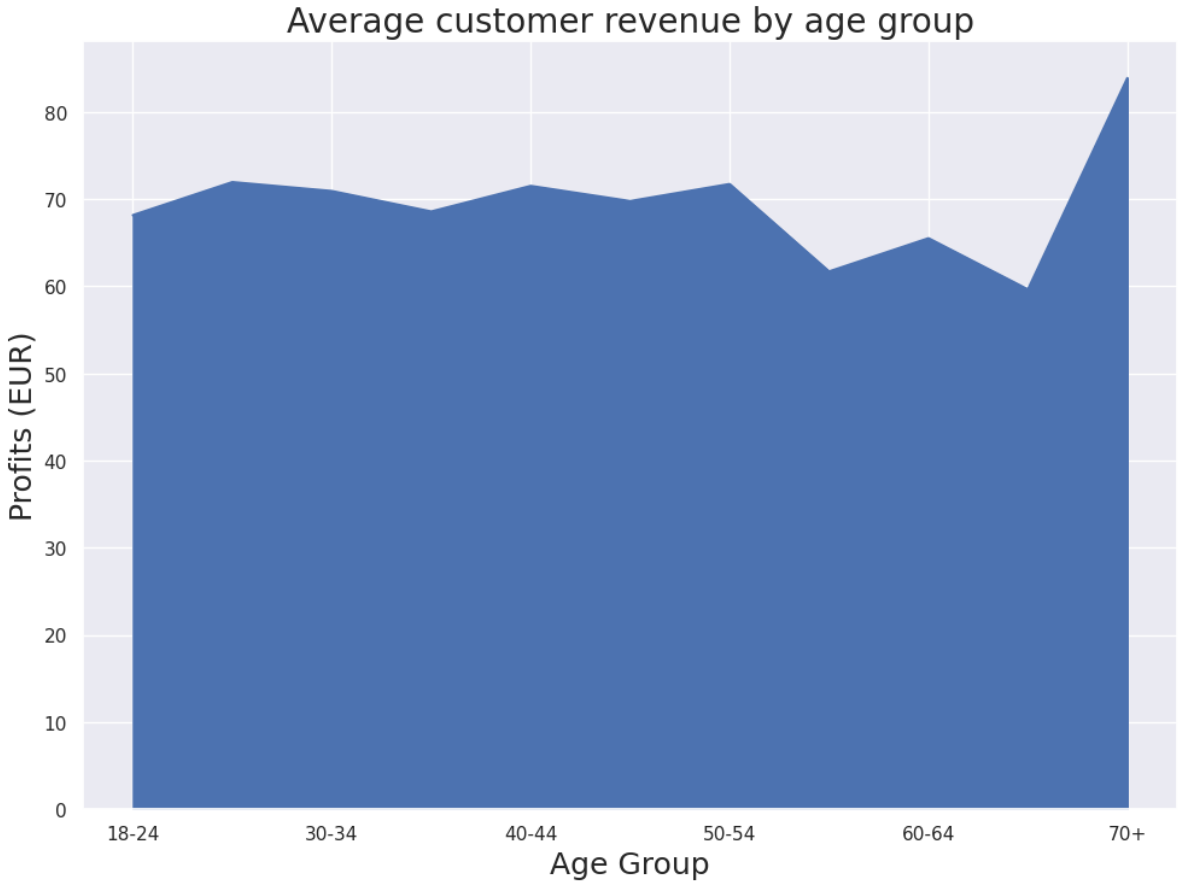

Max Order value by Is holiday
Order value
0
100
200
300
400
500
600
700
False
Is holiday

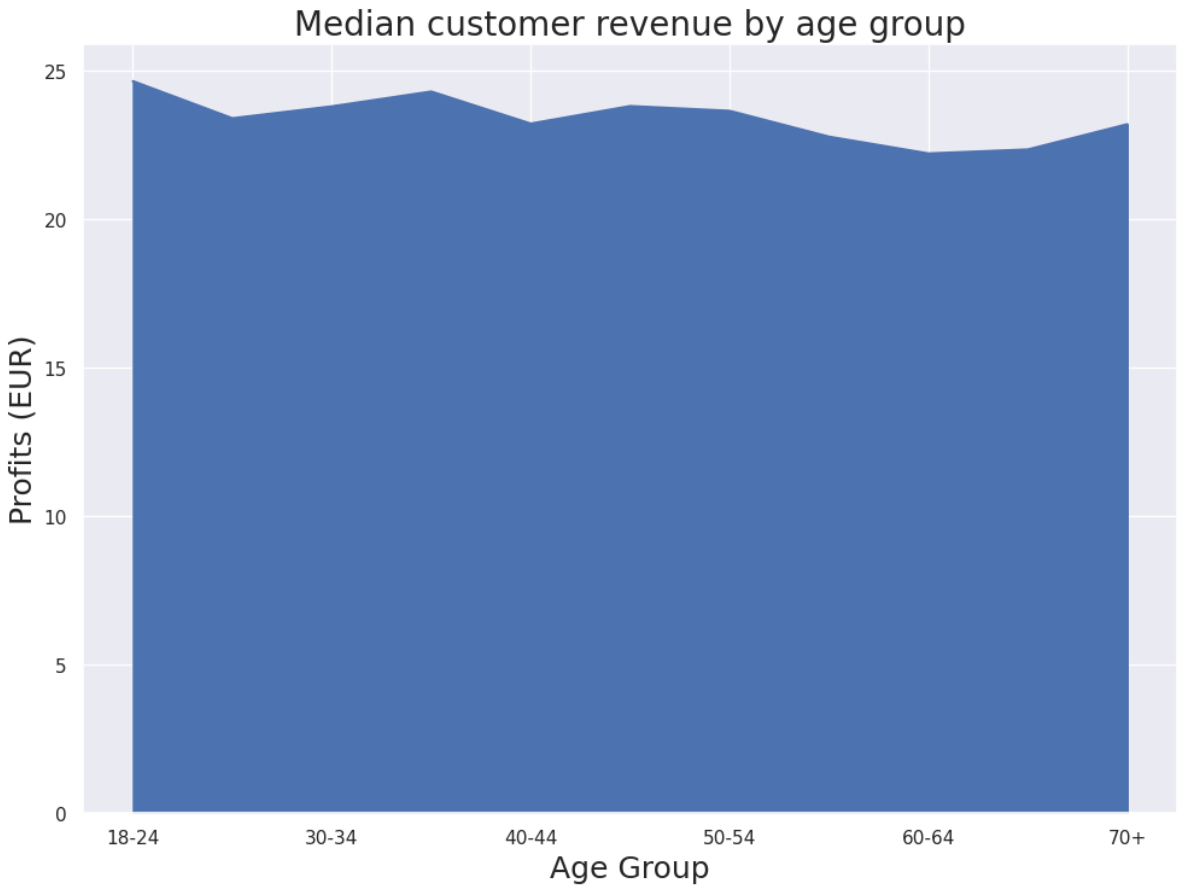


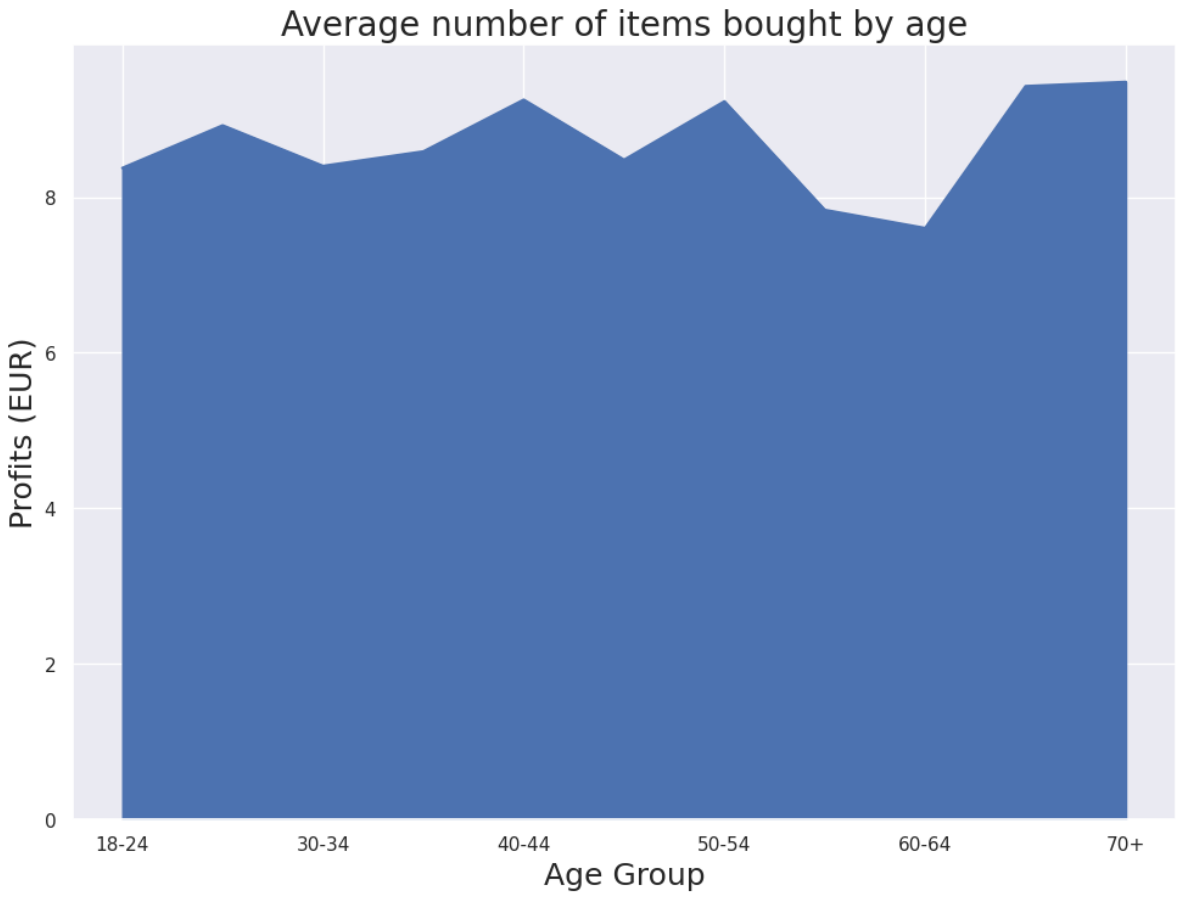
In [ ]:

```
#Plotting the profits per age group
customers_df[["age_bin","total_revenue"]].groupby("age_bin").sum().reset_index().pl
plt.title('Total profits by age group')
plt.ylabel('Profits (EUR)')
plt.xlabel('Age Group')
plt.show()

customers_df[["age_bin","total_revenue"]].groupby("age_bin").mean().reset_index().p
plt.title('Average customer revenue by age group')
plt.ylabel('Profits (EUR)')
plt.xlabel('Age Group')
plt.show()

customers_df[["age_bin","total_revenue"]].groupby("age_bin").median().reset_index(
plt.title('Median customer revenue by age group')
plt.ylabel('Profits (EUR)')
plt.xlabel('Age Group')
plt.show()

customers_df[["age_bin","number_items_bought"]].groupby("age_bin").mean().reset_in
plt.title('Average number of items bought by age')
plt.ylabel('Profits (EUR)')
plt.xlabel('Age Group')
plt.show()
```


Total profits by age group
Profits (EUR)
250000
200000
150000
100000
50000
0
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+
Age Group


Average customer revenue by age group
Profits (EUR)
80
70
60
50
40
30
20
10
0
18-24
30-34
40-44
50-54
60-64
70+
Age Group

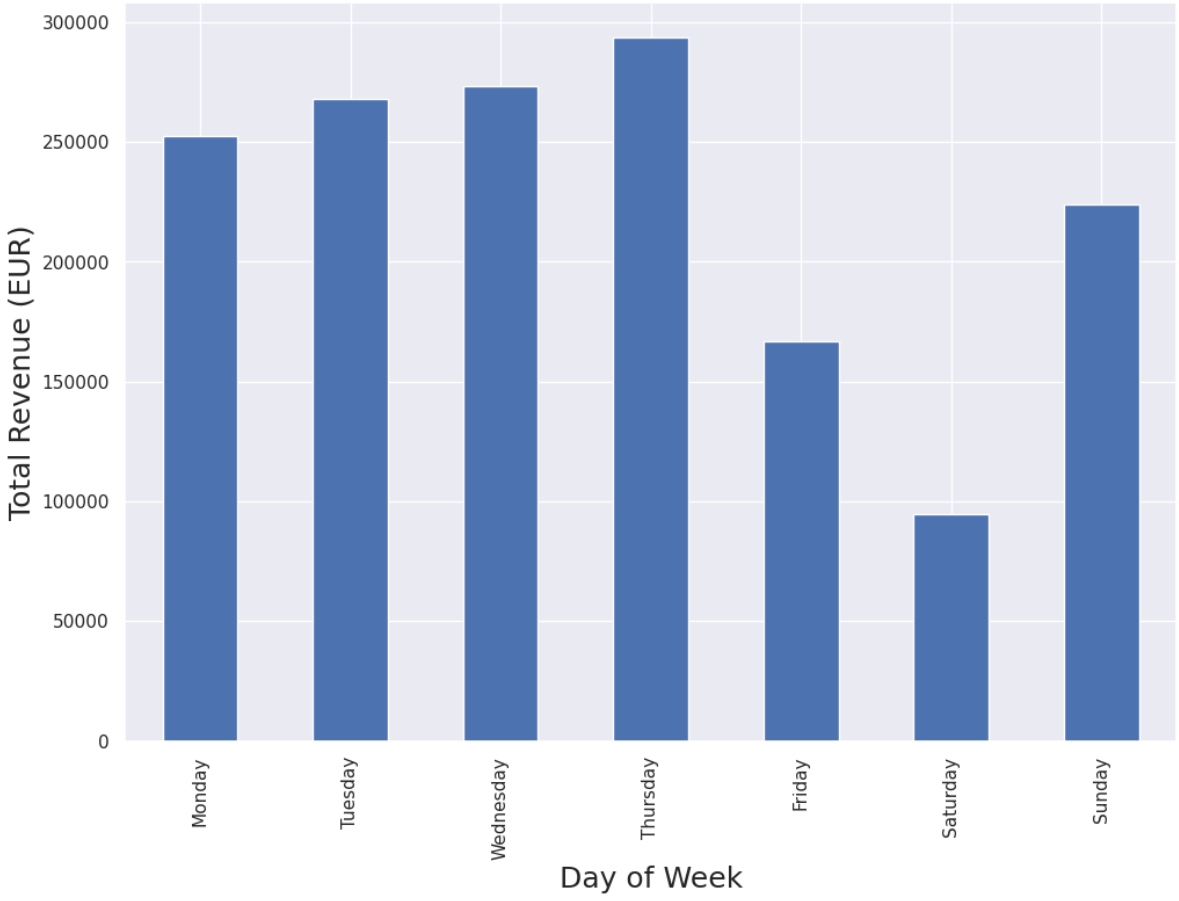


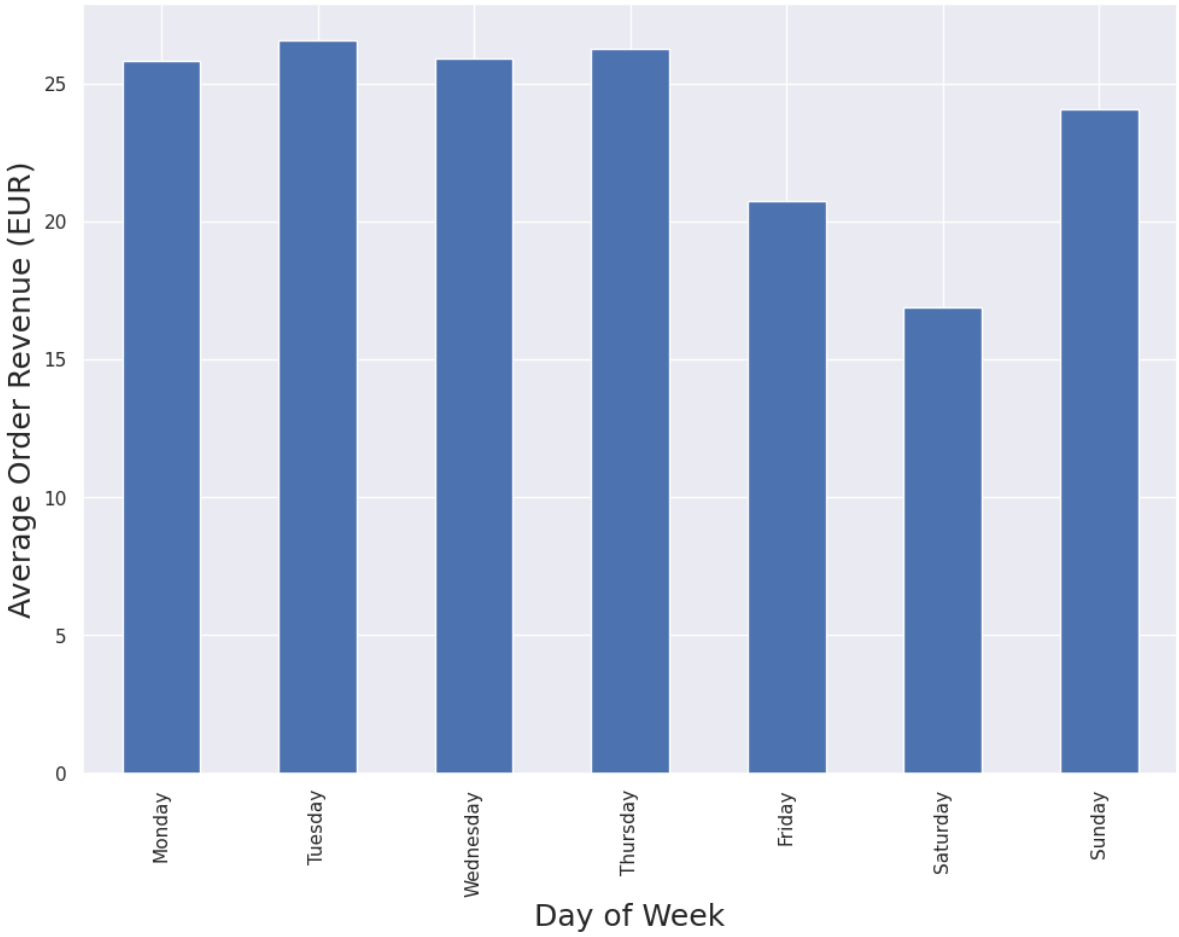


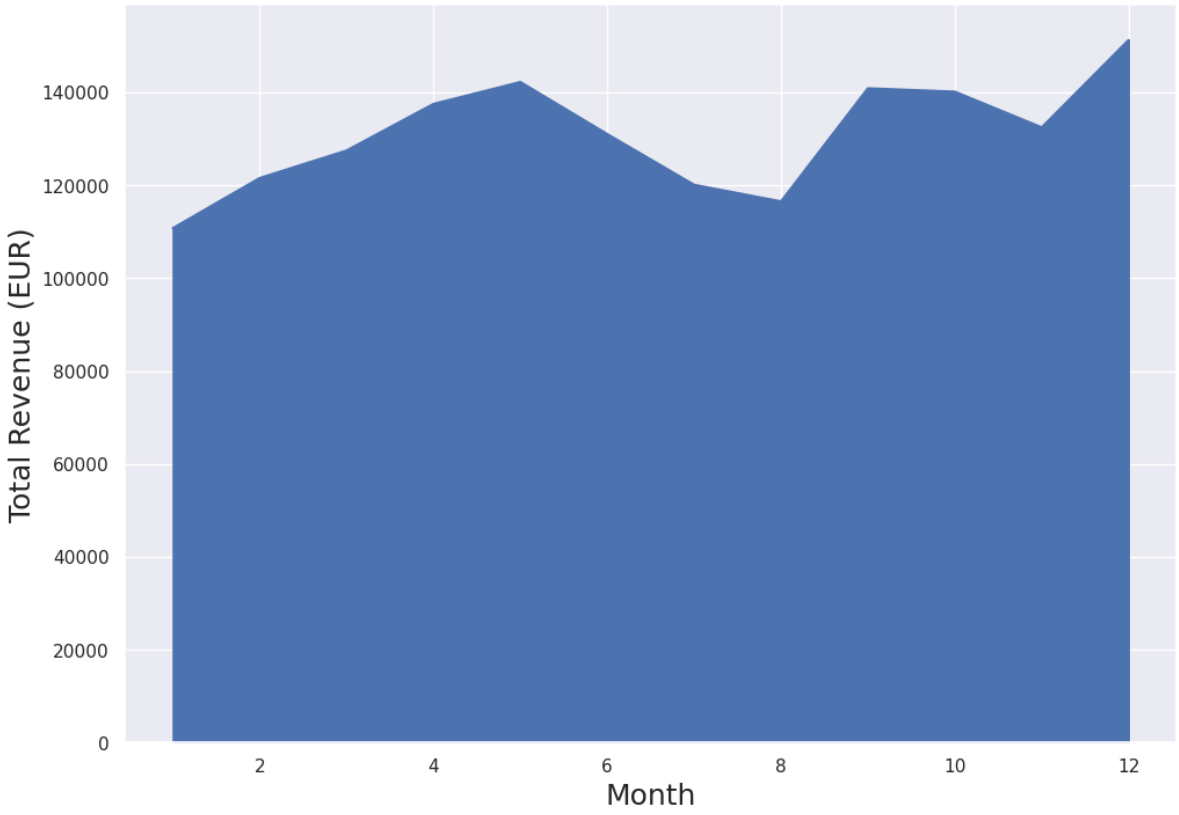
In [ ]:

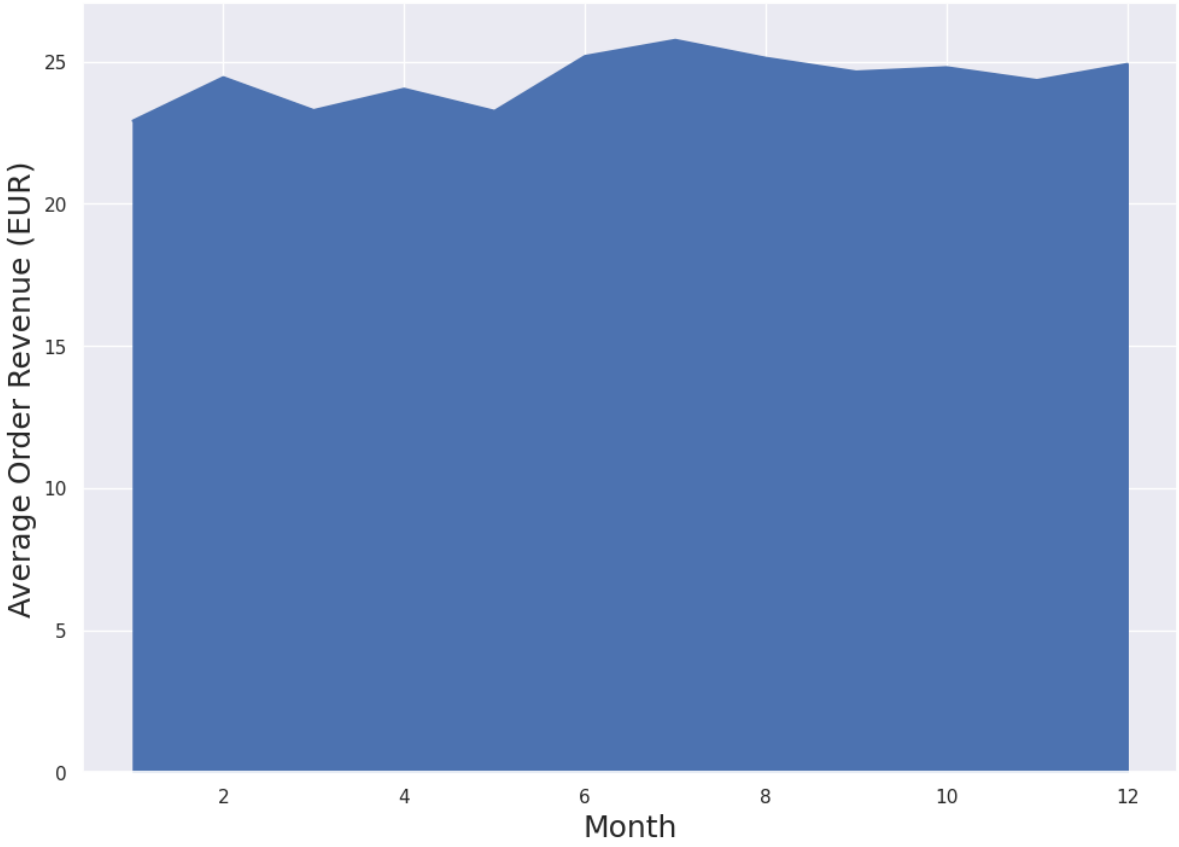
```
transactions['weekday'] = transactions.date.dt.day_name()
cats = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday', 'Sunday
transactions['weekday'] = pd.Categorical(transactions['weekday'], categories=cats,
transactions.groupby('weekday').sum(numeric_only=True).reset_index().plot(x='weekda
plt.xlabel('Day of Week')
plt.ylabel('Total Revenue (EUR)')
plt.show()
```

```
transactions.groupby('weekday').mean(numeric_only=True).reset_index().plot(x='week
plt.xlabel('Day of Week')
plt.ylabel('Average Order Revenue (EUR)')
plt.show()
transactions.groupby('month').sum(numeric_only=True).reset_index().plot(x='month',
plt.xlabel('Month')
plt.ylabel('Total Revenue (EUR)')
plt.show()
transactions.groupby('month').mean(numeric_only=True).reset_index().plot(x='month'
plt.xlabel('Month')
plt.ylabel('Average Order Revenue (EUR)')
plt.show()
```




Average Order Revenue (EUR)
0
5
10
15
20
25
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
Sunday
Day of Week


Total Revenue (EUR)
0
20000
40000
60000
80000
100000
120000
140000
2
4
6
8
10
12
Month



In [ ]:
```
def plot_stacked_graphs(cust_df: pd.DataFrame, categories: list[str]):
    for c in categories:
        subset =
```

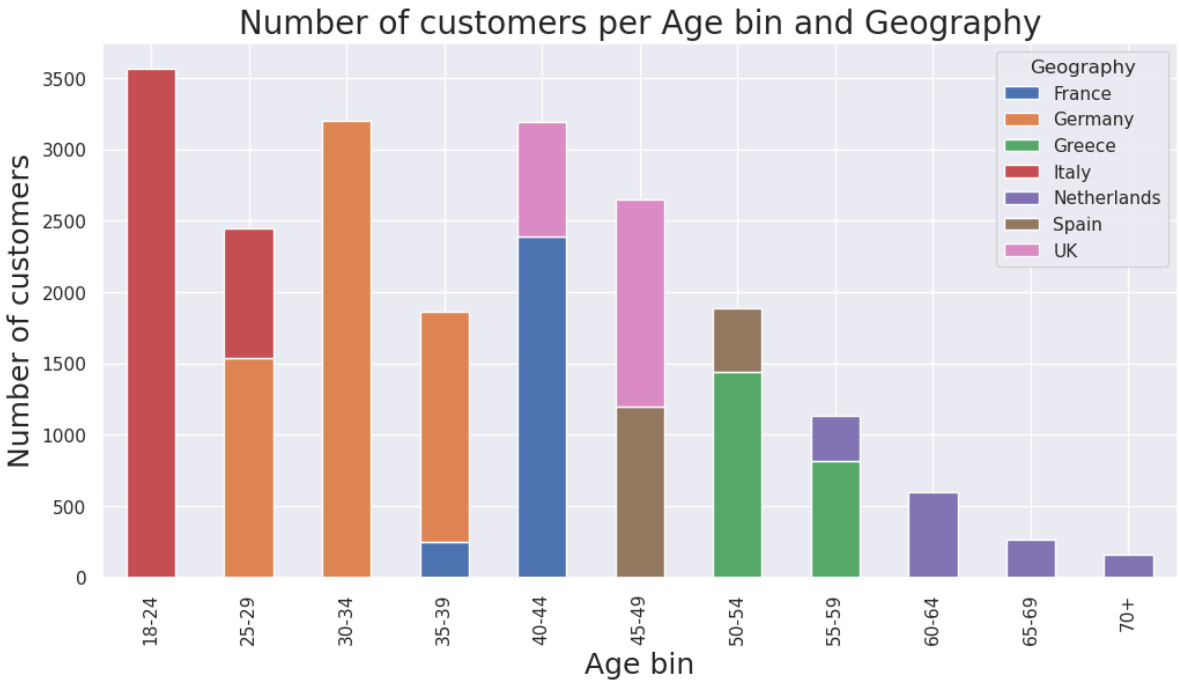
In [ ]:
```
stack_subset = customers_df[['age_bin', 'GEOGRAPHY','GENDER', 'total_revenue','numl


stack_subset = stack_subset[stack_subset.number > 0]
stack_subset
```

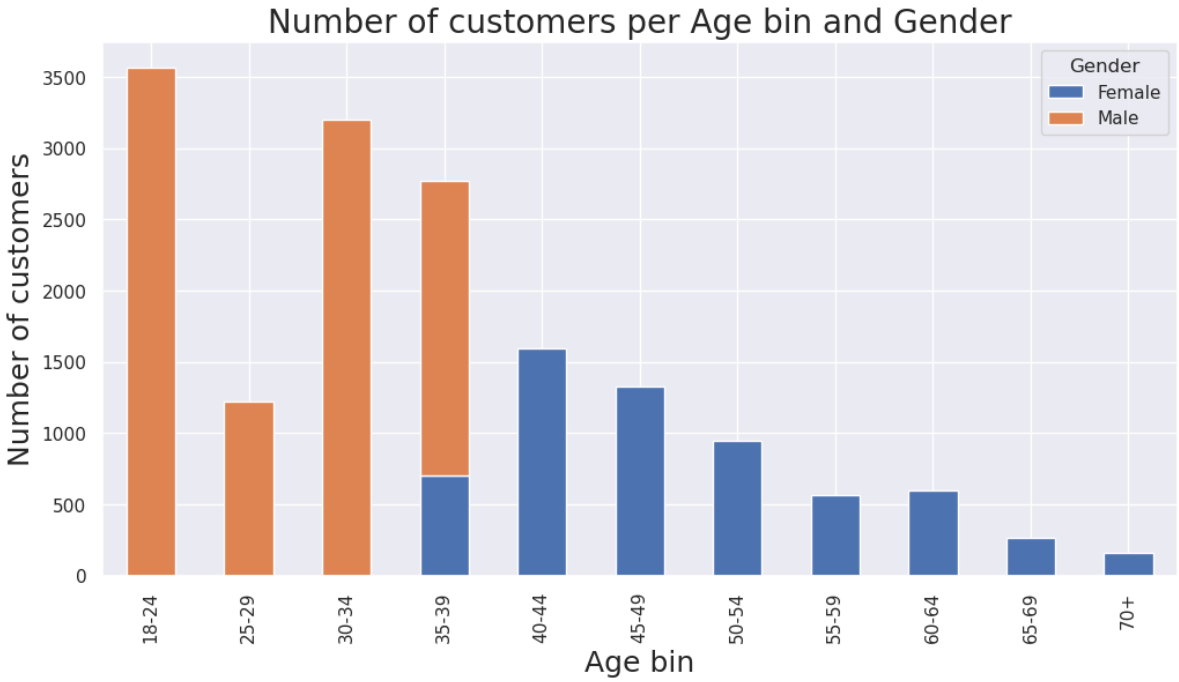
Out[ ]:

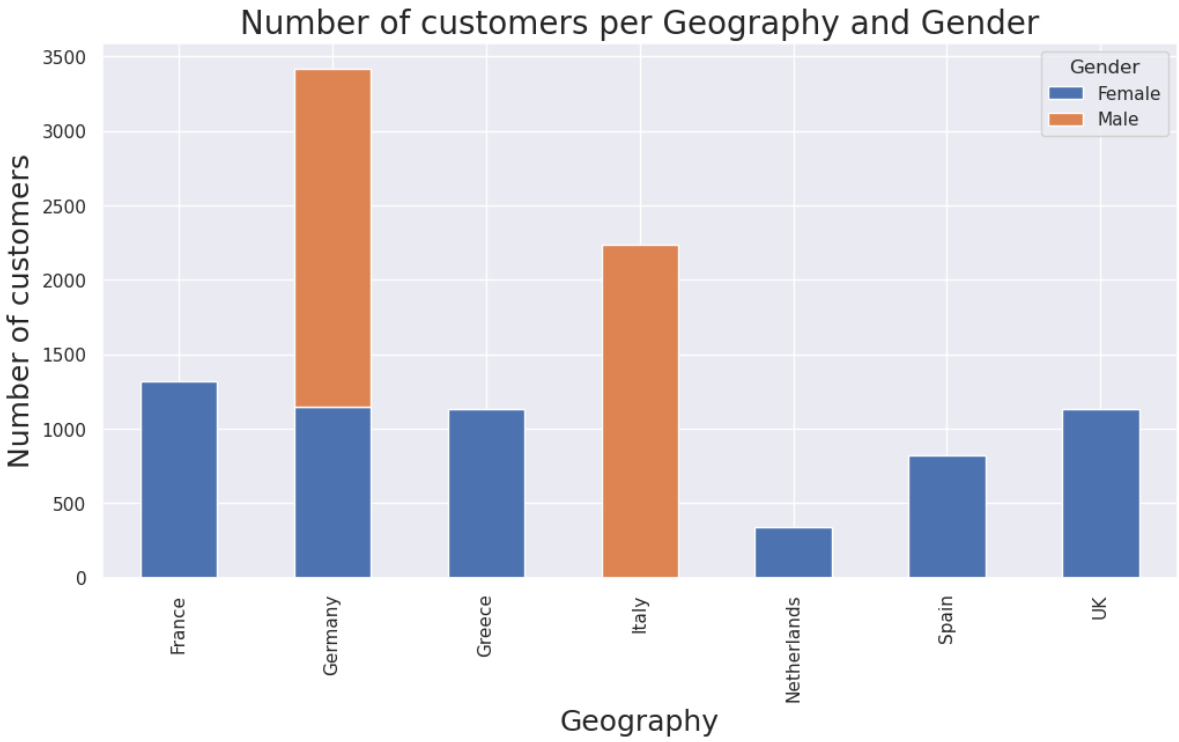
	age_bin	GEOGRAPHY	GENDER	number	total_value	total_orders
7	18-24	Italy	M	3568	243137.85	10047
17	25-29	Germany	M	1537	115777.08	4486
21	25-29	Italy	M	906	59957.49	2595
31	30-34	Germany	M	3200	226910.09	9170
42	35-39	France	F	251	16786.70	774
44	35-39	Germany	F	1151	81876.21	3331
45	35-39	Germany	M	2069	139316.99	5774
56	40-44	France	F	2390	163398.33	6681
68	40-44	UK	F	806	65101.95	2610
80	45-49	Spain	F	1200	85769.90	3465
82	45-49	UK	F	1453	99296.66	4041
88	50-54	Greece	F	1445	99247.20	4316
94	50-54	Spain	F	446	36360.35	1330
102	55-59	Greece	F	817	50745.64	2196
106	55-59	Netherlands	F	314	19018.76	841
120	60-64	Netherlands	F	600	39297.11	1622
134	65-69	Netherlands	F	269	16038.40	726
148	70+	Netherlands	F	164	13754.22	501

In [ ]:

```
def plot_stack_plots(stack_df: pd.DataFrame, cat_cols: list):
    for c in cat_cols:
        subset = stack_df.pivot_table(columns=c[0], index=c[1], values='number')
        subset.plot(kind='bar', stacked=True, figsize=(12,6))
        c_title = c[0].replace('_', ' ').capitalize()
        i_title = c[1].replace('_', ' ').capitalize()
        plt.title('Number of customers per {} and {}'.format(i_title, c_title))
        plt.xlabel(i_title)
        plt.ylabel('Number of customers')
        if c[0] == 'GENDER':
            plt.legend(title = c_title, labels=['Female', 'Male'])
        else:
            plt.legend(title=c_title)
        plt.show()

plot_stack_plots(stack_subset, [('GEOGRAPHY', 'age_bin'), ('GENDER', 'age_bin'), (
```


Number of customers per Age bin and Geography
Number of customers
Age bin
Geography
France
Germany
Greece
Italy
Netherlands
Spain
UK
0
500
1000
1500
2000
2500
3000
3500
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+


Number of customers per Age bin and Gender
Number of customers
Age bin
Gender
Female
Male
0
500
1000
1500
2000
2500
3000
3500
18-24
25-29
30-34
35-39
40-44
45-49
50-54
55-59
60-64
65-69
70+



3 RFM Analysis

In [ ]:

```
def compute_RFM(cust_df: pd.DataFrame, revenue_col: str, last_date_col: str, num_o
    # Computing Recency
    last_purchase_date = max(cust_df[last_date_col])
    cust_df["recency"] = cust_df[last_date_col].apply(lambda x: (last_purchase_dat
    # Monetary value has already been calculated, it is in the total revenue colum
    # Frequency has also been calculated and is in the number_orders column

    # Calculating Recency (R), Frequency (F) and Monetary value (M) ranks
    cust_df['R_rank'] = cust_df['recency'].rank(ascending=False)
    cust_df['F_rank'] = cust_df[num_orders_col].rank(ascending=True)
    cust_df['M_rank'] = cust_df[revenue_col].rank(ascending=True)

    cust_df['R_rank_norm'] = 100*cust_df['R_rank']/cust_df['R_rank'].max()
    cust_df['F_rank_norm'] = 100*cust_df['F_rank']/cust_df['F_rank'].max()
    cust_df['M_rank_norm'] = 100*cust_df['M_rank']/cust_df['M_rank'].max()

    cust_df.drop(columns=['R_rank', 'F_rank', 'M_rank'], inplace=True)

    # Computing RFM
    cust_df['RFM_score'] = 0.15*cust_df['R_rank_norm'] + 0.28*cust_df['F_rank_norm
    cust_df['RFM_score'] = cust_df['RFM_score'].apply(lambda x: round(x,2)*0.05)

    # Creating customer segments
    cust_df["customer_segment"] = None
    cust_df.loc[(pd.isna(cust_df['customer_segment'])) & (cust_df['RFM_score'] > 4
    cust_df.loc[(pd.isna(cust_df['customer_segment'])) & (cust_df['RFM_score'] > 4
    cust_df.loc[(pd.isna(cust_df['customer_segment'])) & (cust_df['RFM_score'] > 3
    cust_df.loc[(pd.isna(cust_df['customer_segment'])) & (cust_df['RFM_score'] > 1
    cust_df.loc[(pd.isna(cust_df['customer_segment'])) & (cust_df['RFM_score'] <= 1

    return cust_df.sort_index()

customers_df = compute_RFM(customers_df, 'total_revenue', 'date_max', 'number_orde
customers_df.head(2)
```

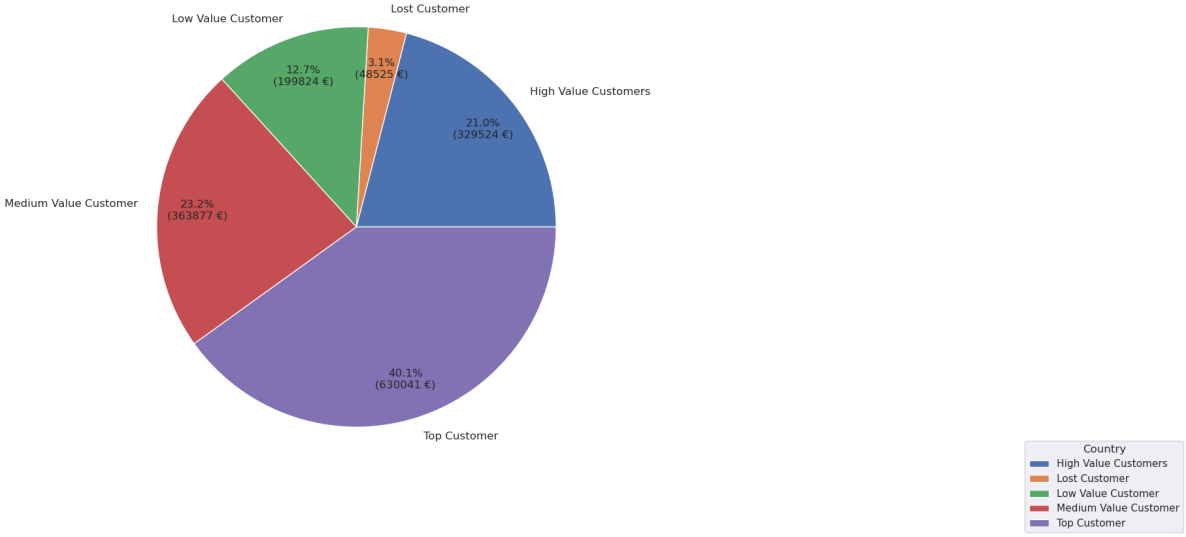
Out[ ]:

Customer_ID	GENDER	AGE	GEOGRAPHY	number_orders	order_value_min	order_value_max	tot
1	M	22	Italy	1	16.29	16.29	
2	M	24	Italy	2	7.77	15.00	

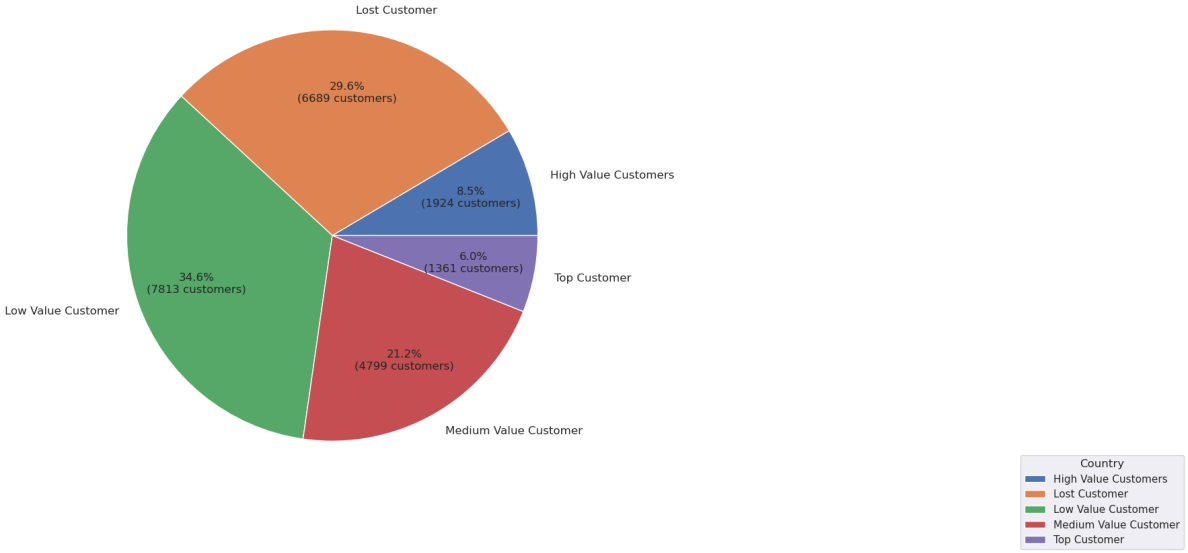
2 rows × 31 columns

In [ ]:
```
rfm_summary = customers_df.groupby('customer_segment').sum().reset_index()
plt.pie(rfm_summary['total_revenue'], labels=rfm_summary['customer_segment'],
        pctdistance = 0.8,
        labeldistance = 1.1,
        autopct=lambda x: f"{x:.1f}%\n({(x/100)*sum(rfm_summary['total_revenue']):
        textprops={"size": 12},
        radius = 1.2)
plt.legend(loc="best", bbox_to_anchor=(2.5,0), title="Country")
plt.show()
```

```
/tmp/ipykernel_912/579592388.py:1: FutureWarning: The default value of numeric_onl
y in DataFrameGroupBy.sum is deprecated. In a future version, numeric_only will de
fault to False. Either specify numeric_only or select only columns which should be
valid for the function.
  rfm_summary = customers_df.groupby('customer_segment').sum().reset_index()
```



In [ ]:
```
rfm_summary = customers_df.groupby('customer_segment').count().reset_index()
plt.pie(rfm_summary['total_revenue'], labels=rfm_summary['customer_segment'],
        pctdistance = 0.7,
        labeldistance = 1.1,
        autopct=lambda x: f"{x:.1f}%\n({(x/100)*sum(rfm_summary['total_revenue']):
        textprops={"size": 12},
        radius = 1.2)
plt.legend(loc="best", bbox_to_anchor=(2.5,0), title="Country")
plt.show()
```



4 Clustering

4.1 Test Train Split

In [ ]:
```
# First we need to select only relevant features
X = pd.get_dummies(customers_df, columns=['GENDER','GEOGRAPHY'])
X = X.select_dtypes(include=np.number)
X = X.dropna(axis=0)
print(X.shape)
X.head(2)
```

```
(22586, 34)
```

Out[ ]:

Customer_ID	AGE	number_orders	order_value_min	order_value_max	total_revenue	order_value_r
1	22	1	16.29	16.29	16.29	1
2	24	2	7.77	15.00	22.77	1

2 rows × 34 columns

In [ ]:
```
from sklearn.model_selection import train_test_split
#Creating train test split to be able to test stability
X_train, X_test = train_test_split(X, test_size=0.1, random_state=45658, shuffle=T

X_train.head()
```

Out[ ]:

Customer_ID	AGE	number_orders	order_value_min	order_value_max	total_revenue	order_value_r
21374	20	1	7.25	7.25	7.25	7.25
5399	49	1	15.65	15.65	15.65	15.65
8417	34	3	7.00	11.25	26.60	8.86
11299	47	1	19.92	19.92	19.92	19.92
20709	38	2	6.07	12.28	18.35	9.17

5 rows × 34 columns

4.2 Preprocessing Pipeline

In [ ]:

```
from sklearn.base import TransformerMixin, BaseEstimator
from sklearn.preprocessing import FunctionTransformer

#Define a function to remove highly correlated features
def correlated_features(dataset, threshold=0.8):
    correlated_columns = set()
    df = pd.DataFrame(dataset)
    correlations = df.corr()
    for i in range(len(correlations)):
        for j in range(i):
            if abs(correlations.iloc[i,j]) > threshold:
                correlated_columns.add(correlations.columns[i])
    return correlated_columns

class CorrSelector(BaseEstimator, TransformerMixin):

    def __init__(self, threshold=0.8):
        self.threshold = threshold
        self.correlated_cols = None

    def fit(self, X, y=None):
        self.correlated_cols = correlated_features(X, threshold=self.threshold)
        self.correlated_indexes = [pd.DataFrame(X).columns.get_loc(c) for c in sel
        return self

    def transform(self, X, y=None, **kwargs):
        self.X = X
        X_df = pd.DataFrame(X)
        return X_df.drop(X_df.columns[list(self.correlated_cols)], axis=1)

    def get_params(self, deep=False):
        return {"threshold": self.threshold}

    def get_feature_names_out(self, features_out):
        feature_list = list(features_out)
        for i in self.correlated_indexes:
            feature_list.pop(i)
        return feature_list
```

In [ ]:

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.feature_selection import VarianceThreshold
from sklearn.pipeline import Pipeline
#Preprocessing pipeline with scaling and feature selection
```

```
preprocess_pipe = Pipeline([
    ('var', VarianceThreshold(threshold=0.04)),
    ('corr', CorrSelector()),
    ('scaler', MinMaxScaler())

])

X_train_tr = preprocess_pipe.fit_transform(X_train)
X_test_tr = preprocess_pipe.transform(X_test)

X_train_tr.shape
```

Out[ ]:
```
(20327, 22)
```

4.3 K Means Clustering

We will perform KMeans clustering to classify our customers into different segments

In [ ]:
```
def cluster_stability(X, est, n_iter=10, random_state=None):
    labels = []
    indices = []
    for i in range(n_iter):
        # draw bootstrap samples, store indices
        sample_indices = rng.randint(0, X.shape[0], X.shape[0])
        indices.append(sample_indices)
        est = clone(est)
        if hasattr(est, "random_state"):
            # randomize estimator if possible
            est.random_state = rng.randint(1e5)
        X_bootstrap = X[sample_indices]
        est.fit(X_bootstrap)
        # store clustering outcome using original indices
        relabel = -np.ones(X.shape[0], dtype=np.int32)
        relabel[sample_indices] = est.labels_
        labels.append(relabel)
    scores = []
    for l, i in zip(labels, indices):
        for k, j in zip(labels, indices):
            in_both = np.intersect1d(i, j)
            scores.append(adjusted_rand_score(l[in_both], k[in_both]))
    return np.mean(scores)
```

4.3.1 Hyperparameter Optimization (Kmeans)

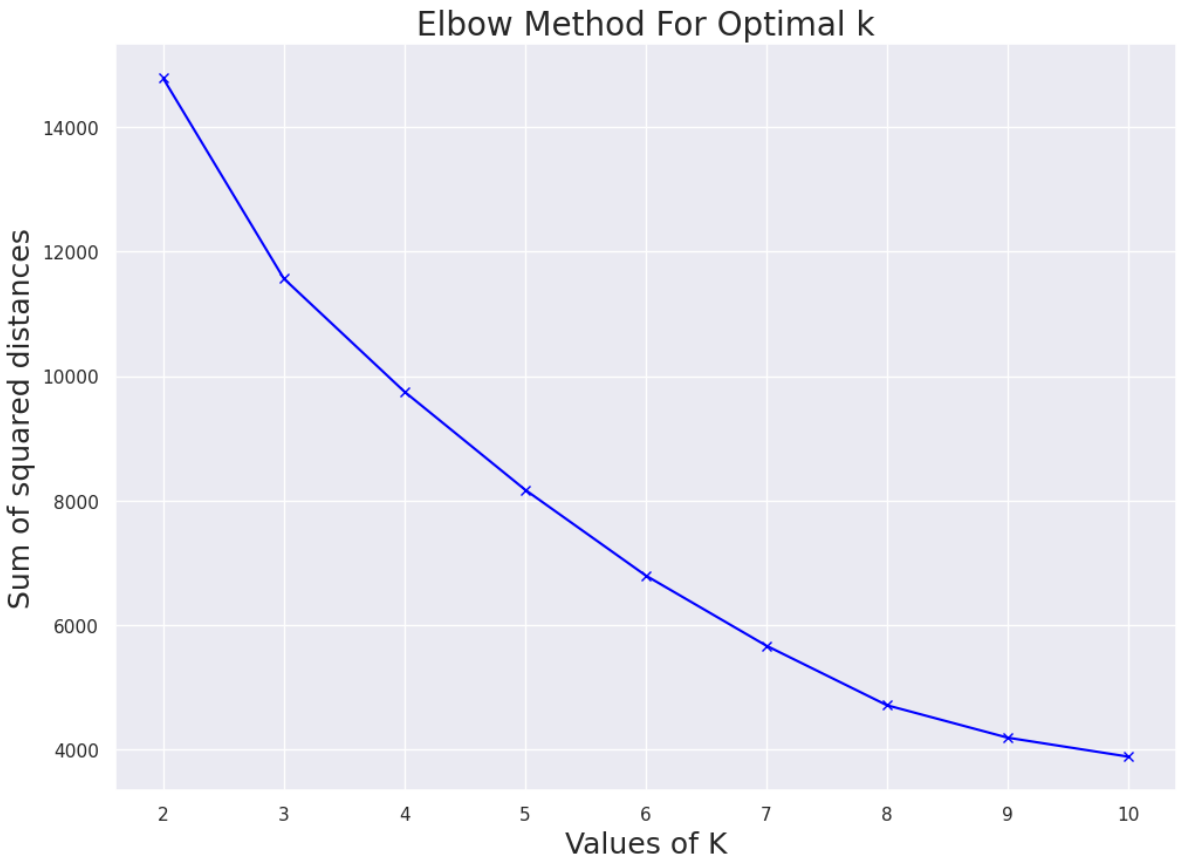
In [ ]:
```
kf = KFold(n_splits=5)
scores = []

for k in range (2,11):  #Higher values of K have been tested before full implementa
    labels = []
    indices = []
    scores_stab = []
    SSD = []
    SIL = []
    for train_index, test_index in kf.split(X_train_tr):
        X_train = X_train_tr[train_index]
        X_test = X_train_tr[test_index]

        km_tr = KMeans(n_clusters=k, random_state=10, n_init = 10).fit(X_train)
        cluster_labels = km_tr.labels_
        ssd = km_tr.inertia_
```

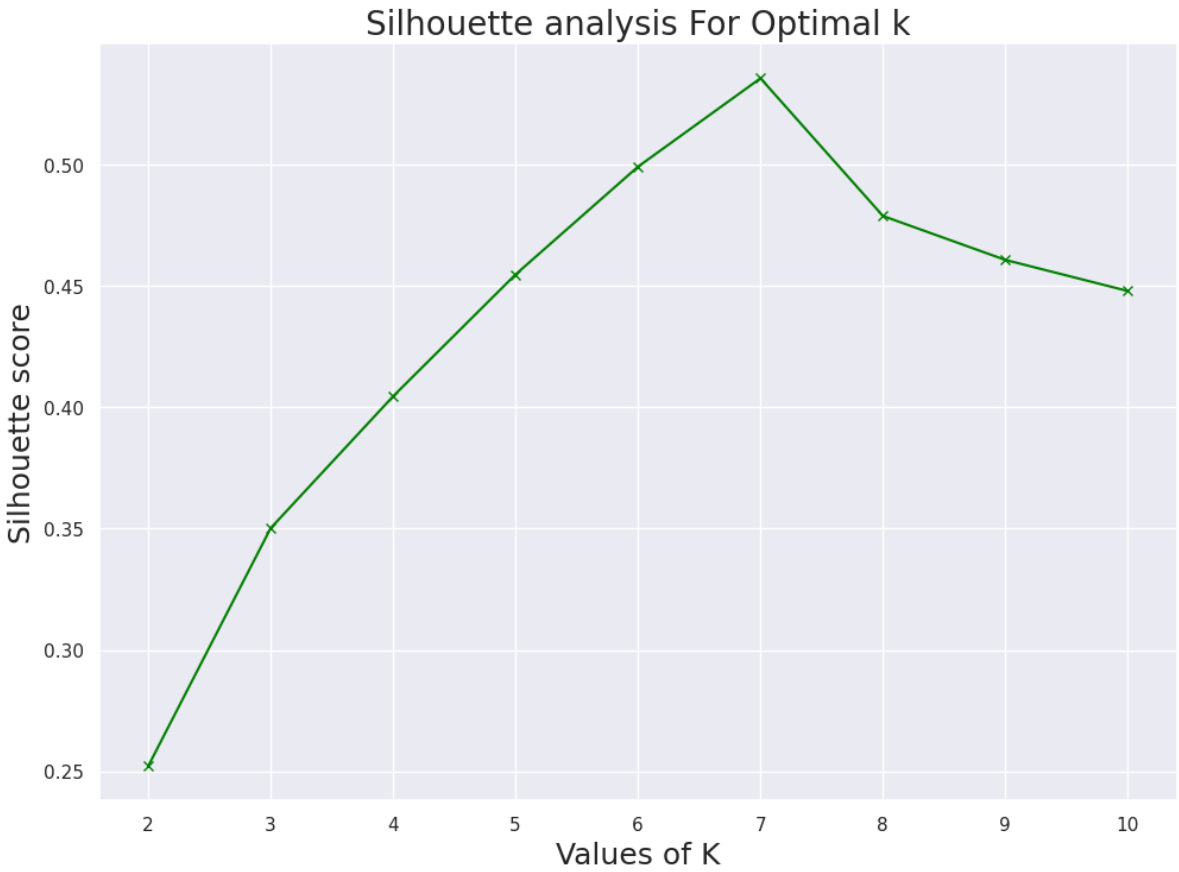
```
        sil = silhouette_score(X_train, cluster_labels)
        SSD.append(ssd)
        SIL.append(sil)

        print("K: {}, Silhouette: {}".format(k, sil))

        #Calculating stability on the train set
        indices.append(train_index)
        relabel = -np.ones(X_train_tr.shape[0], dtype=np.int32)
        relabel[train_index] = cluster_labels
        labels.append(relabel)

        gc.enable()
        del X_train, X_test
        gc.collect()

    for l, i in zip(labels, indices):
        for m, j in zip(labels, indices):
        # we also compute the diagonal
            in_both = np.intersect1d(i, j)
            scores_stab.append(adjusted_rand_score(l[in_both], m[in_both]))


    print("K: {}, SSD: {}, Silhouette: {}, Stability: {}".format(k, np.mean(SSD), r
    scores.append({'K': k, 'SSD': np.mean(SSD), 'Silhouette': np.mean(SIL), 'Stabi

scores_df = pd.DataFrame(scores)
```

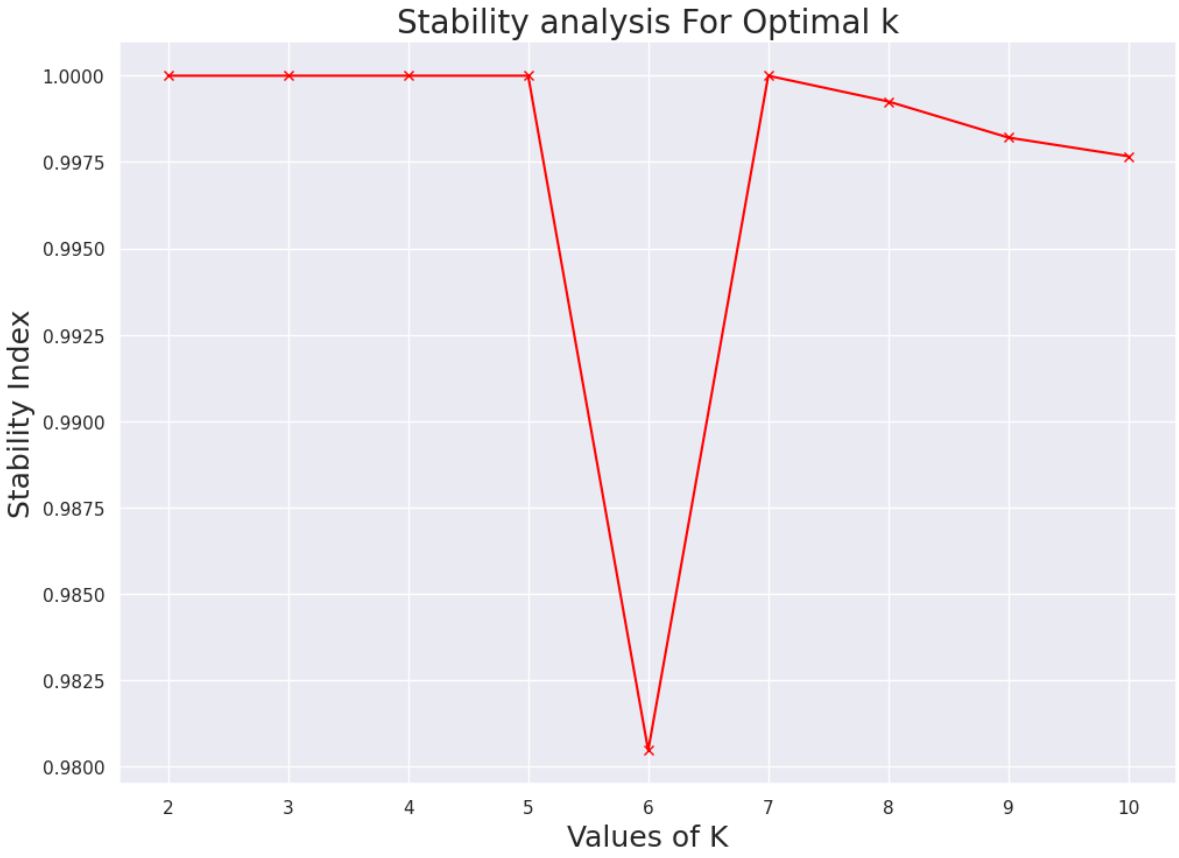
```
K: 2, Silhouette: 0.2516228097170912
K: 2, Silhouette: 0.25138709649399626
K: 2, Silhouette: 0.252910788366884
K: 2, Silhouette: 0.25223851290714133
K: 2, Silhouette: 0.25236665188865876
K: 2, SSD: 14785.527455661073, Silhouette: 0.25210517187475434, Stability: 1.0
K: 3, Silhouette: 0.350394411981345
K: 3, Silhouette: 0.34965346097513106
K: 3, Silhouette: 0.35034781202914256
K: 3, Silhouette: 0.35020485633944437
K: 3, Silhouette: 0.3503381105374748
K: 3, SSD: 11566.633558894715, Silhouette: 0.35018773037250756, Stability: 1.0
K: 4, Silhouette: 0.40477614970416337
K: 4, Silhouette: 0.4048421152892709
K: 4, Silhouette: 0.40381689640174806
K: 4, Silhouette: 0.40489833070019393
K: 4, Silhouette: 0.40433101089042717
K: 4, SSD: 9748.939701966283, Silhouette: 0.4045329005971607, Stability: 1.0
K: 5, Silhouette: 0.4549838247318755
K: 5, Silhouette: 0.45530356387546006
K: 5, Silhouette: 0.4537874464663286
K: 5, Silhouette: 0.4547051844814369
K: 5, Silhouette: 0.4549426564609264
K: 5, SSD: 8173.364556216188, Silhouette: 0.45474453520320546, Stability: 1.0
K: 6, Silhouette: 0.5003022650680633
K: 6, Silhouette: 0.5003740042443241
K: 6, Silhouette: 0.499208599014605
K: 6, Silhouette: 0.5001026756353519
K: 6, Silhouette: 0.49591836401190537
K: 6, SSD: 6795.001231634733, Silhouette: 0.4991811815948499, Stability: 0.9804651
613823048
K: 7, Silhouette: 0.5357297604739991
K: 7, Silhouette: 0.5358604728737244
K: 7, Silhouette: 0.5355126897711169
K: 7, Silhouette: 0.5352355174634308
K: 7, Silhouette: 0.5357054844263481
K: 7, SSD: 5666.994592789537, Silhouette: 0.5356087850017238, Stability: 1.0
K: 8, Silhouette: 0.4784735587251483
K: 8, Silhouette: 0.4790929924522786
K: 8, Silhouette: 0.47965252498732697
K: 8, Silhouette: 0.47836958750527075
K: 8, Silhouette: 0.478651222891815
K: 8, SSD: 4710.136590545218, Silhouette: 0.47884797731236795, Stability: 0.999253
755590957
K: 9, Silhouette: 0.4612253162152353
K: 9, Silhouette: 0.46074163157312353
K: 9, Silhouette: 0.4615304158900736
K: 9, Silhouette: 0.4600039365636909
K: 9, Silhouette: 0.46016512226765033
K: 9, SSD: 4189.253378409412, Silhouette: 0.4607332845019547, Stability: 0.9982089
625338378
K: 10, Silhouette: 0.44812749533132656
K: 10, Silhouette: 0.44795054445324645
K: 10, Silhouette: 0.449323798890992
K: 10, Silhouette: 0.4469403914148095
K: 10, Silhouette: 0.447289863364888
K: 10, SSD: 3884.028523241775, Silhouette: 0.4479264186910525, Stability: 0.997667
8385306192
```

In [ ]:
```python
#Plotting our Elbow Score
ax = plt.figure().gca()
plt.plot(scores_df.K, scores_df.SSD,'x-', color="blue", label = "SSD")
plt.xlabel('Values of K')
plt.ylabel('Sum of squared distances')
```

```
plt.title('Elbow Method For Optimal k')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()

#Displaying the Silhouette score
ax = plt.figure().gca()
plt.plot(scores_df.K,scores_df.Silhouette,'x-', color="green", label="Silhouette s
plt.xlabel('Values of K')
plt.ylabel('Silhouette score')
plt.title('Silhouette analysis For Optimal k')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()

#Displaying our Stability Index
ax = plt.figure().gca()
plt.plot(scores_df.K,scores_df.Stability,'x-', color="red", label="Stability Index
plt.xlabel('Values of K')
plt.ylabel('Stability Index')
plt.title('Stability analysis For Optimal k')
ax.xaxis.set_major_locator(MaxNLocator(integer=True))
plt.show()
```



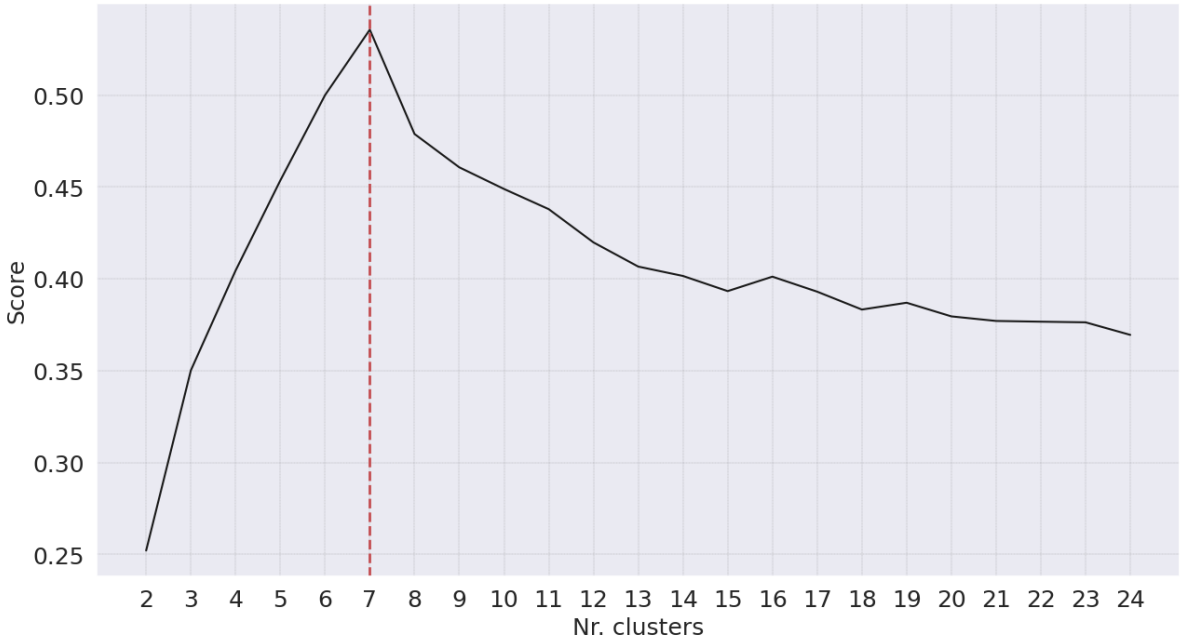




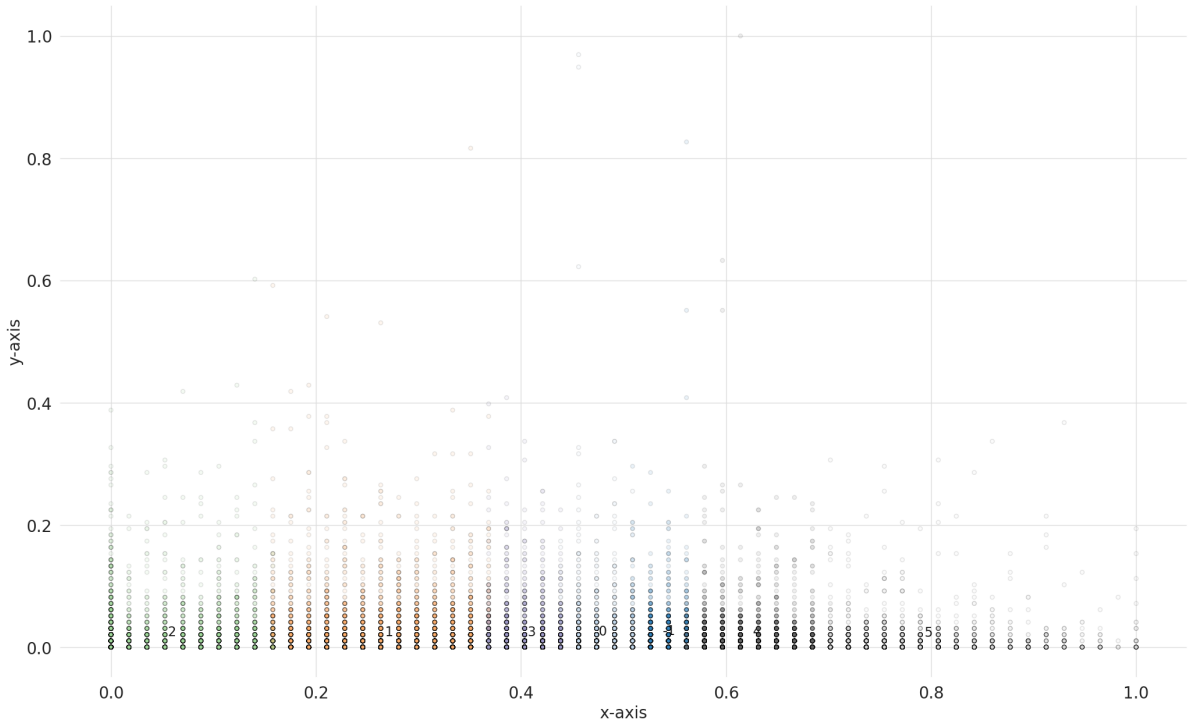
In [ ]:

```
%%capture --no-display
from clusteval import clusteval

ce = clusteval(cluster='kmeans')

results = ce.fit(X_train_tr)

print(results['labx'])
```

```
ce.plot()
ce.scatter(X_train_tr)
```



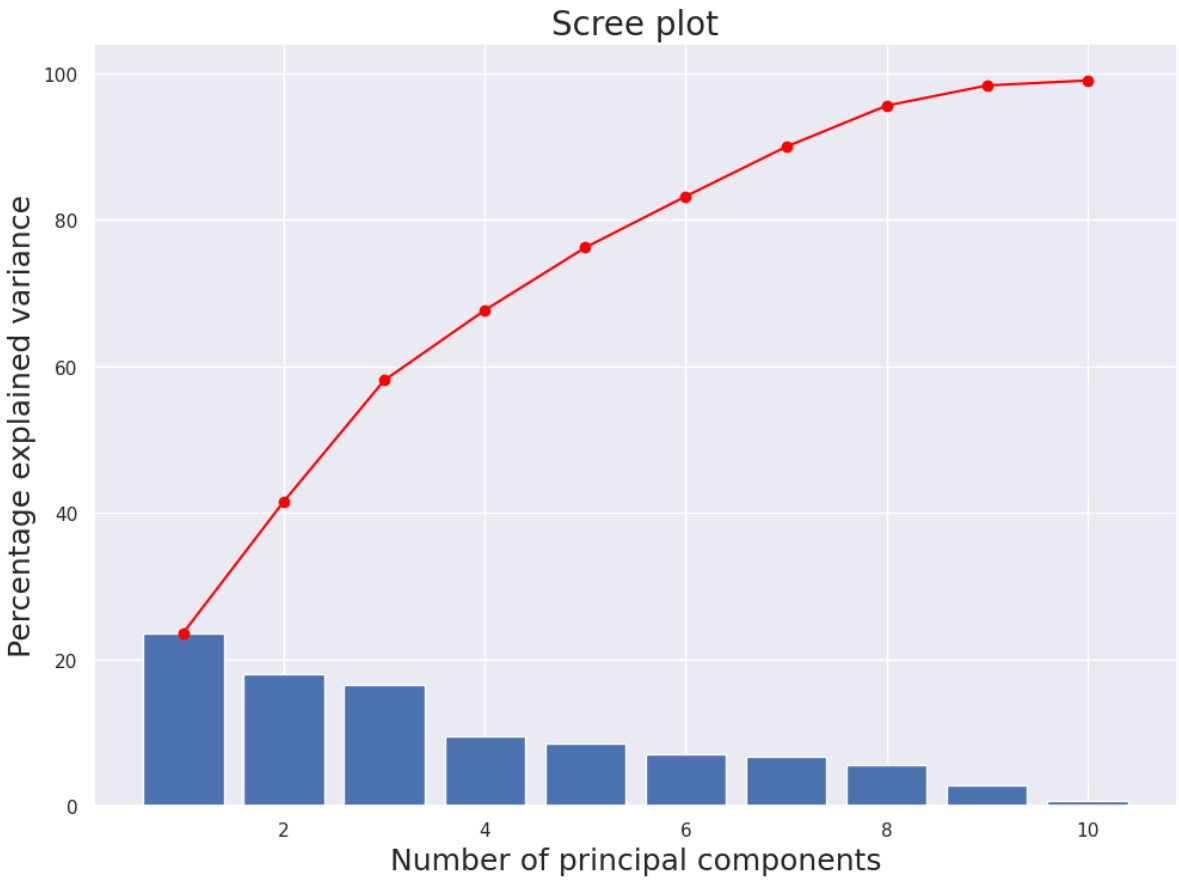
Out[ ]: (<Figure size 2500x1500 with 1 Axes>, <Axes: xlabel='x-axis', ylabel='y-axis'>)



In [ ]:
```
#Selecting 7 as our best amount of clusters
km = KMeans(n_clusters=7, n_init=10)
km_clusters = km.fit_predict(X_train_tr)
```

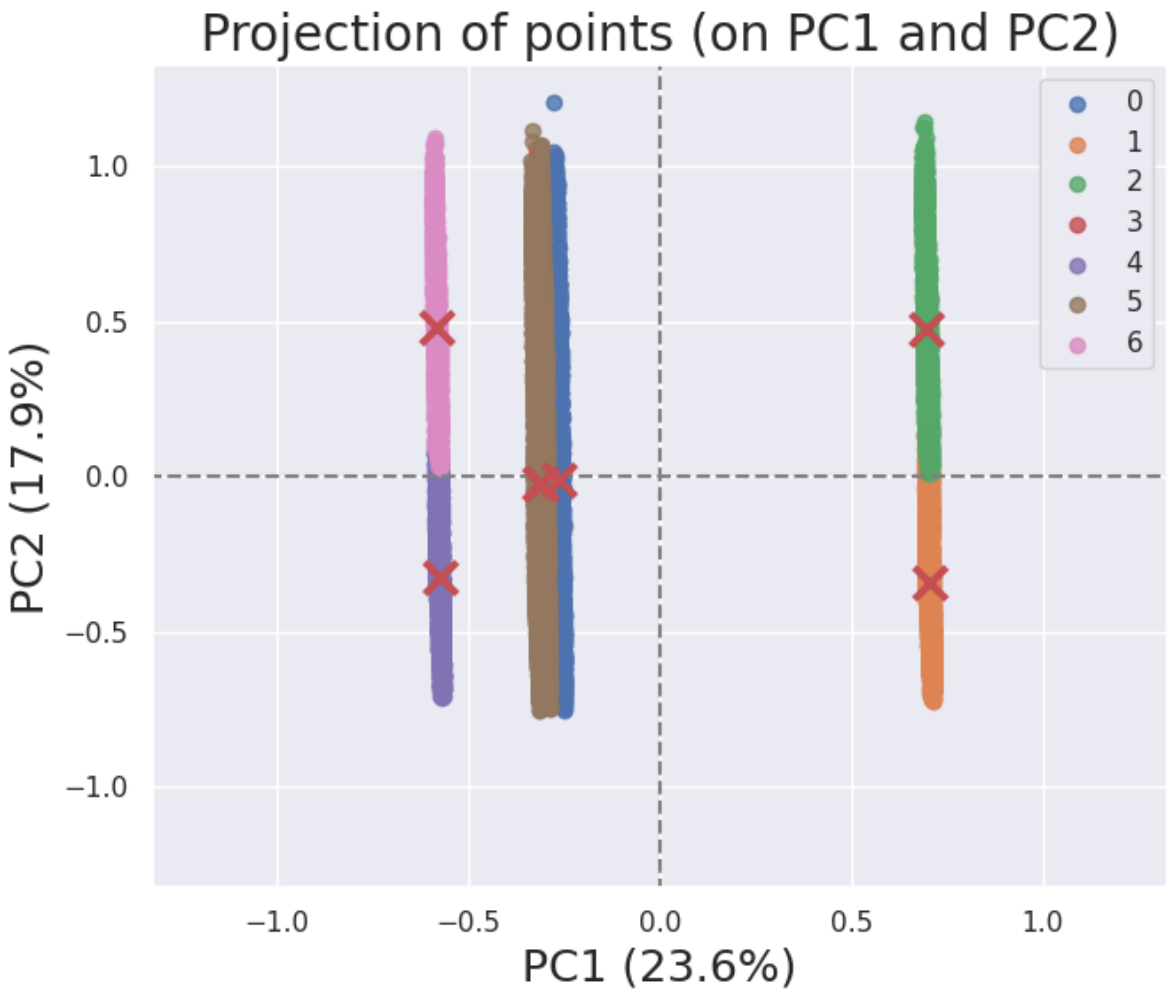
4.3.2 PCA Visualization

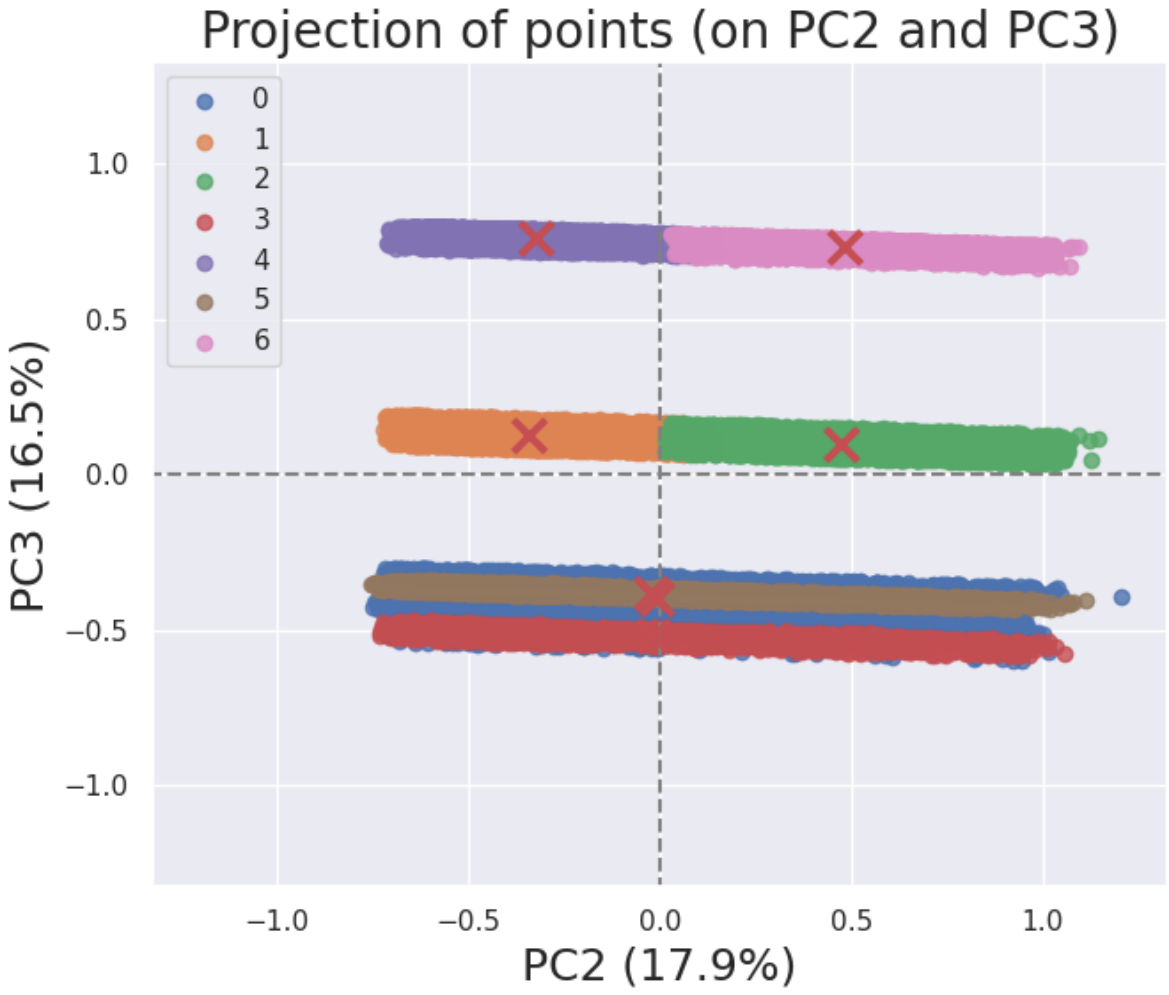
In [ ]:
```
n_components = 10
# Create a PCA model to reduce our data to 2 dimensions for visualisation
pca = PCA(n_components = n_components, n_oversamples=10)
pca.fit(X_train_tr)

display_scree_plot(pca)
```

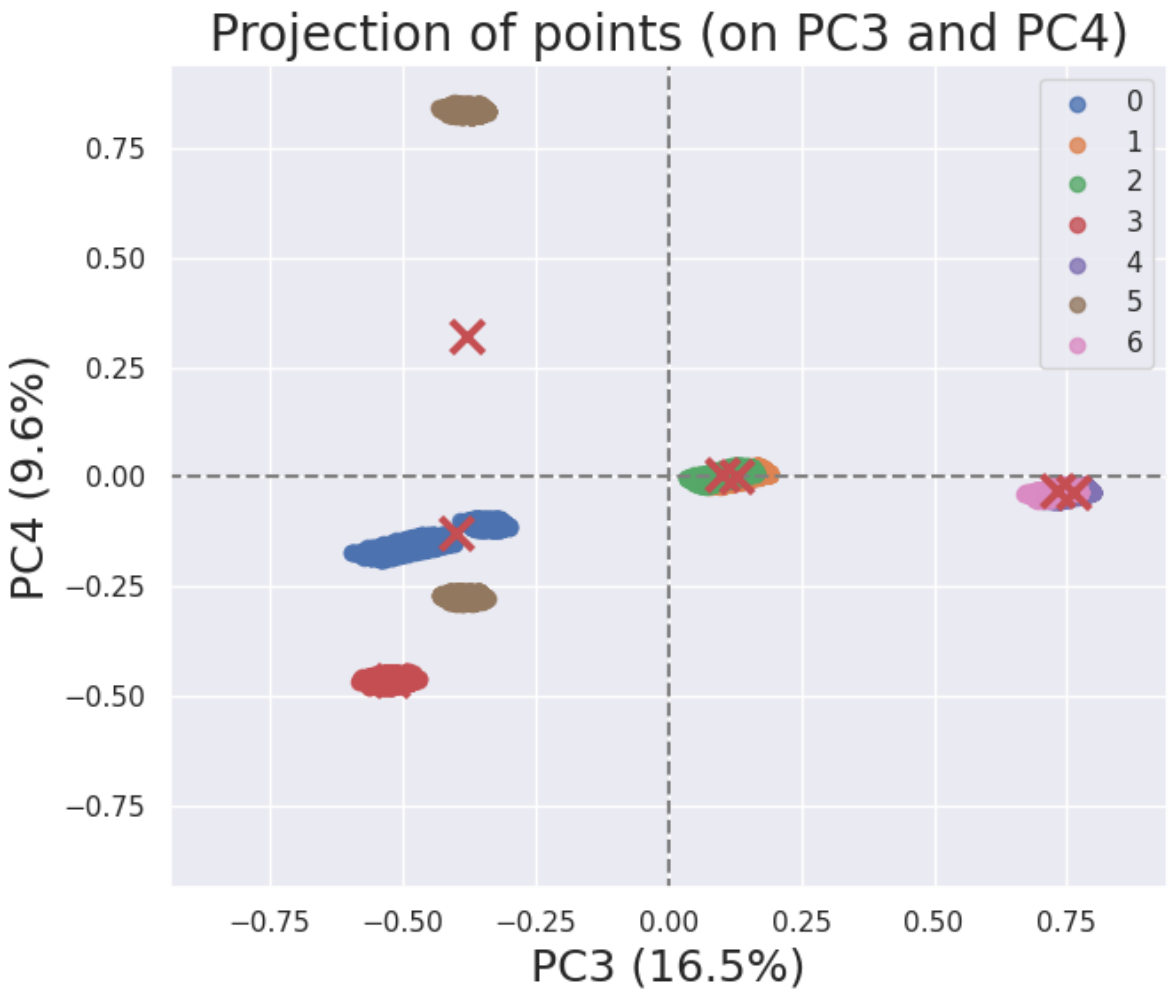


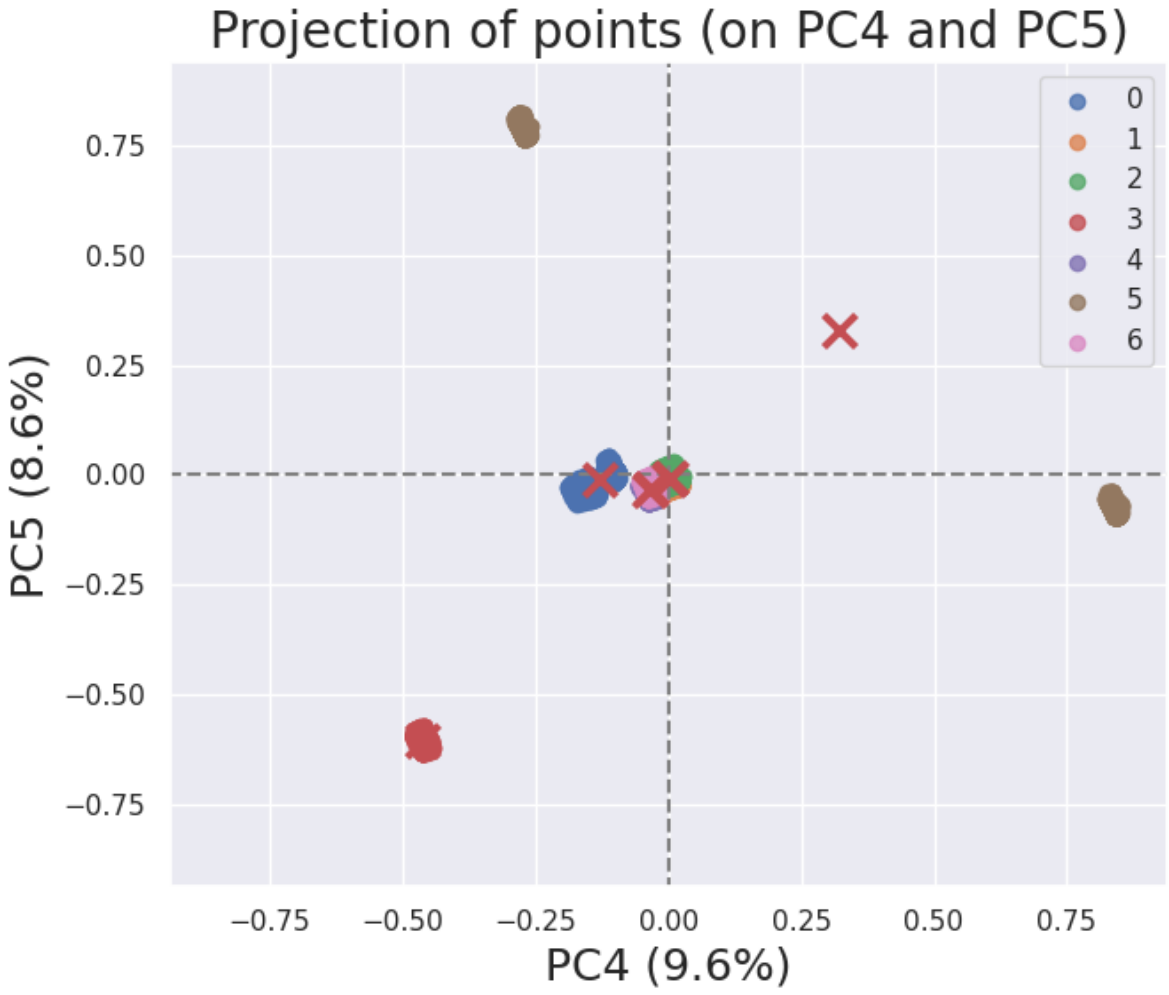
In [ ]:

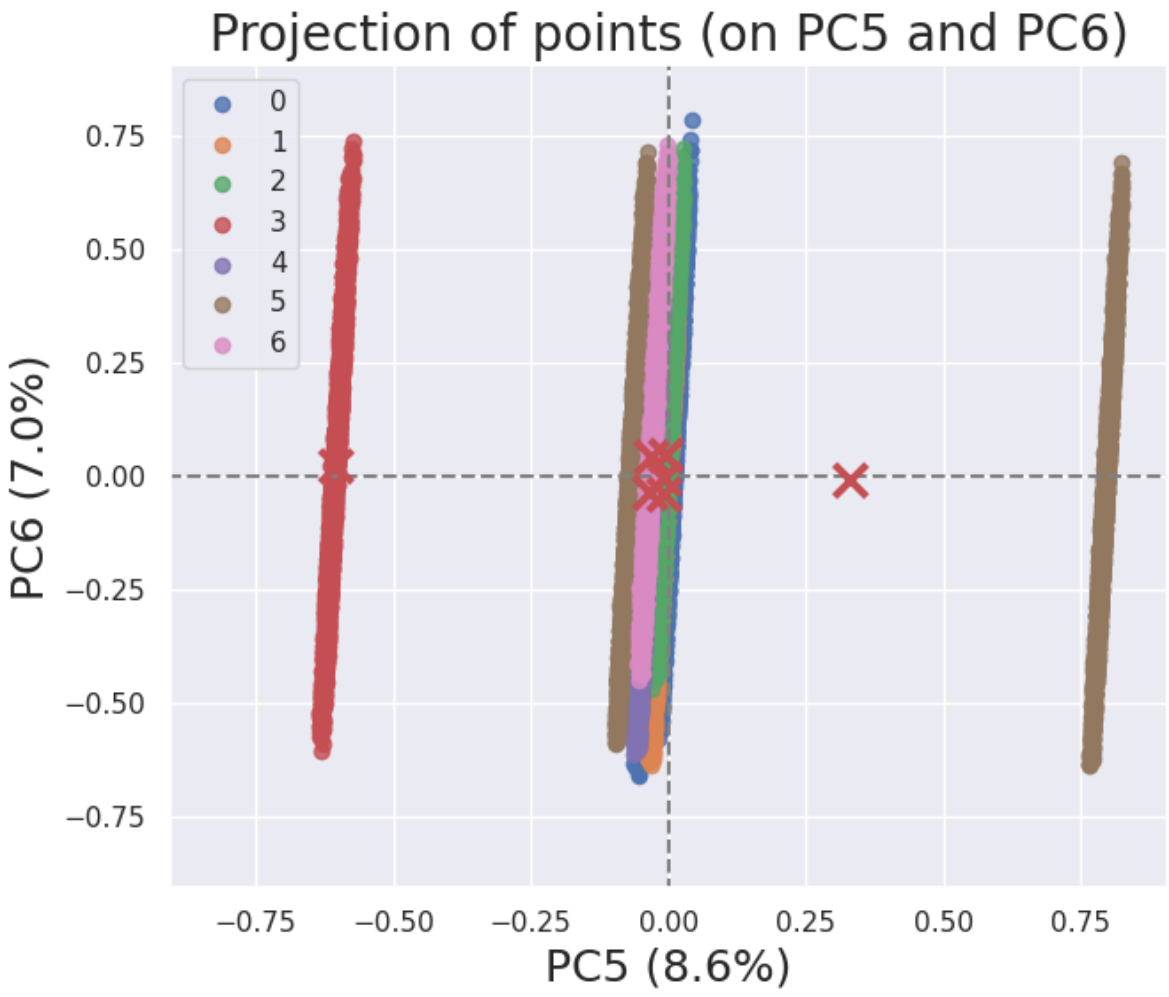
```
#Restricting to 7 components to have 90% of explained variance
n_components = 7
pca = PCA(n_components = n_components, n_oversamples=10)
pca.fit(X_train_tr)
# Transfor the scaled data to the new PCA space
X_train_red = pca.transform(X_train_tr)

centres_reduced = pca.transform(km.cluster_centers_)
for i in range(1,n_components):
    display_factorial_planes(X_train_red, n_components, pca, [(i-1,i)], illustrati
    plt.scatter(centres_reduced[:, i-1], centres_reduced[:, i],
                marker='x', s=169, linewidths=3,
                color='r', zorder=10)
    plt.tight_layout()
    plt.show()
```

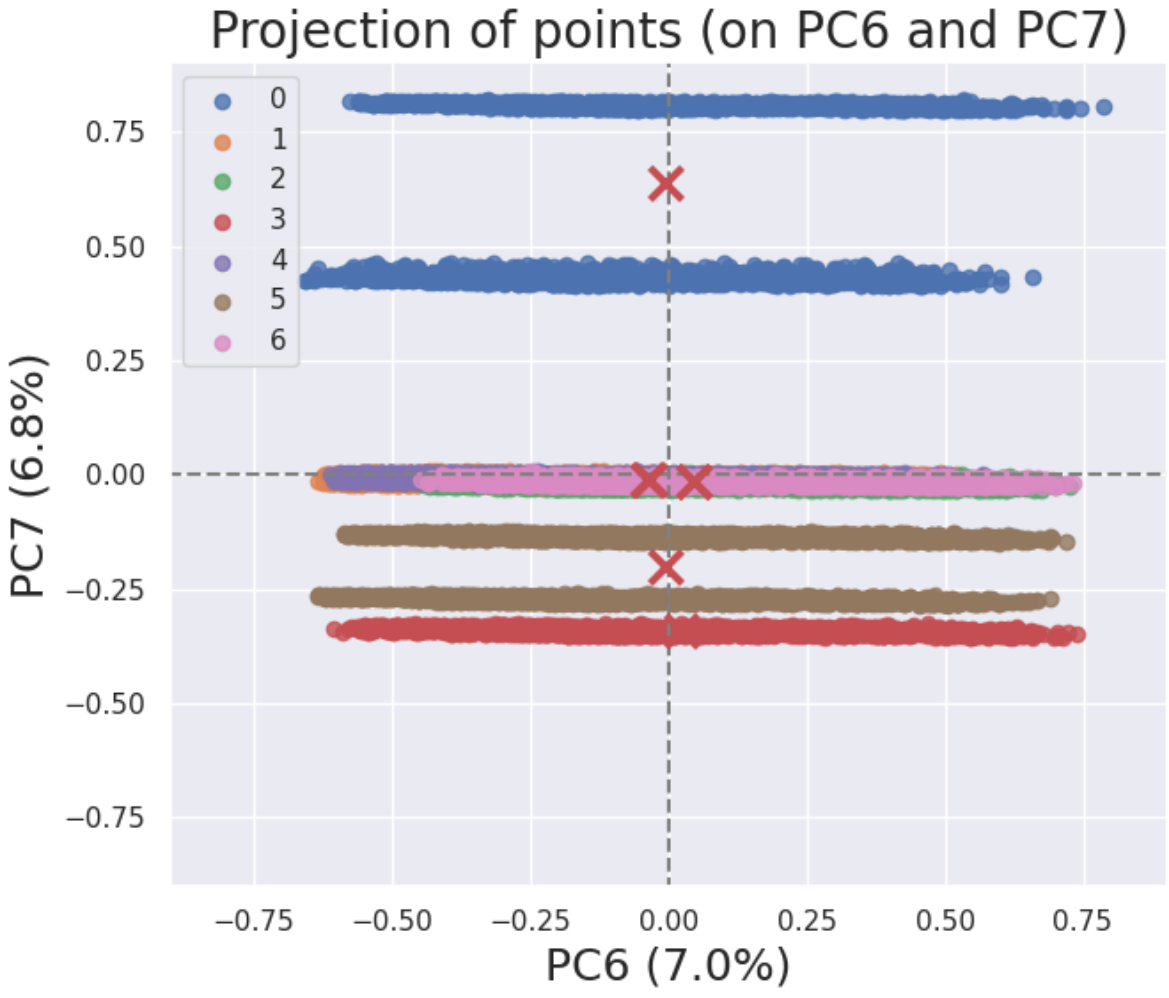

Projection of points (on PC1 and PC2)
PC2 (17.9%)
PC1 (23.6%)
0
1
2
3
4
5
6
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


Projection of points (on PC2 and PC3)
PC3 (16.5%)
PC2 (17.9%)
0
1
2
3
4
5
6
1.0
0.5
0.0
−0.5
−1.0
−1.0
−0.5
0.0
0.5
1.0


Projection of points (on PC3 and PC4)
PC4 (9.6%)
PC3 (16.5%)
0
1
2
3
4
5
6
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75


Projection of points (on PC4 and PC5)
PC5 (8.6%)
PC4 (9.6%)
0
1
2
3
4
5
6
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75


Projection of points (on PC5 and PC6)
0
1
2
3
4
5
6
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
PC6 (7.0%)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
PC5 (8.6%)

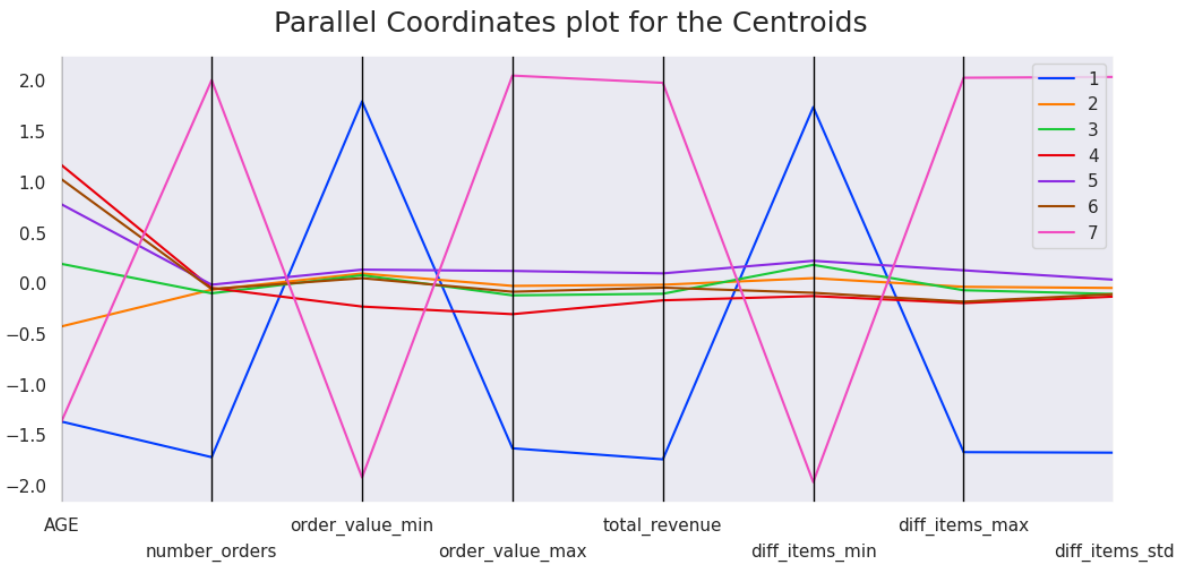

Projection of points (on PC6 and PC7)
0
1
2
3
4
5
6
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
PC7 (6.8%)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
PC6 (7.0%)

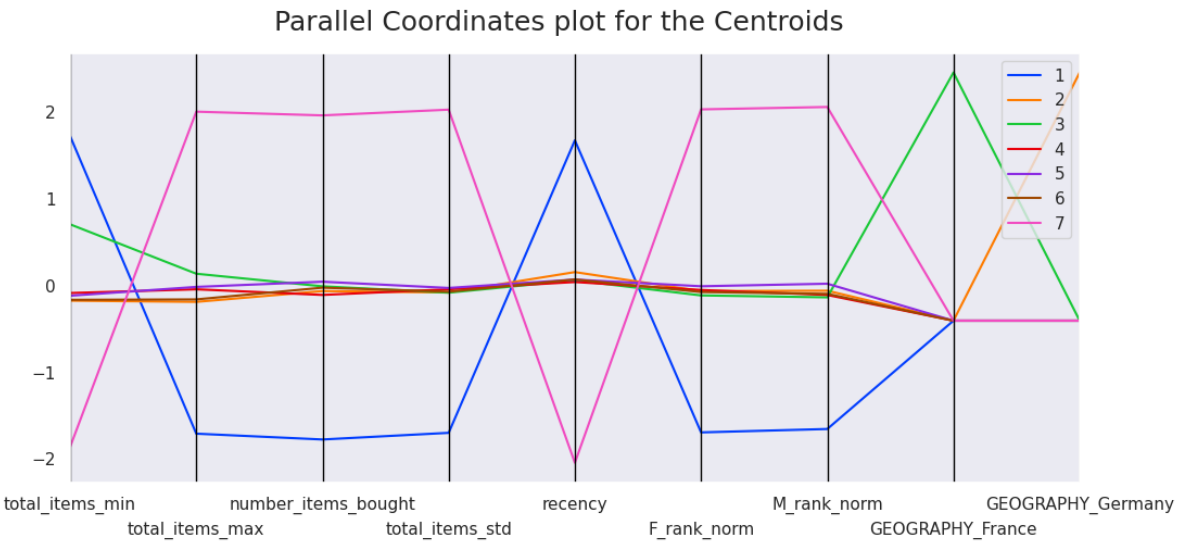
4.3.3 Parallel coordinate visualization

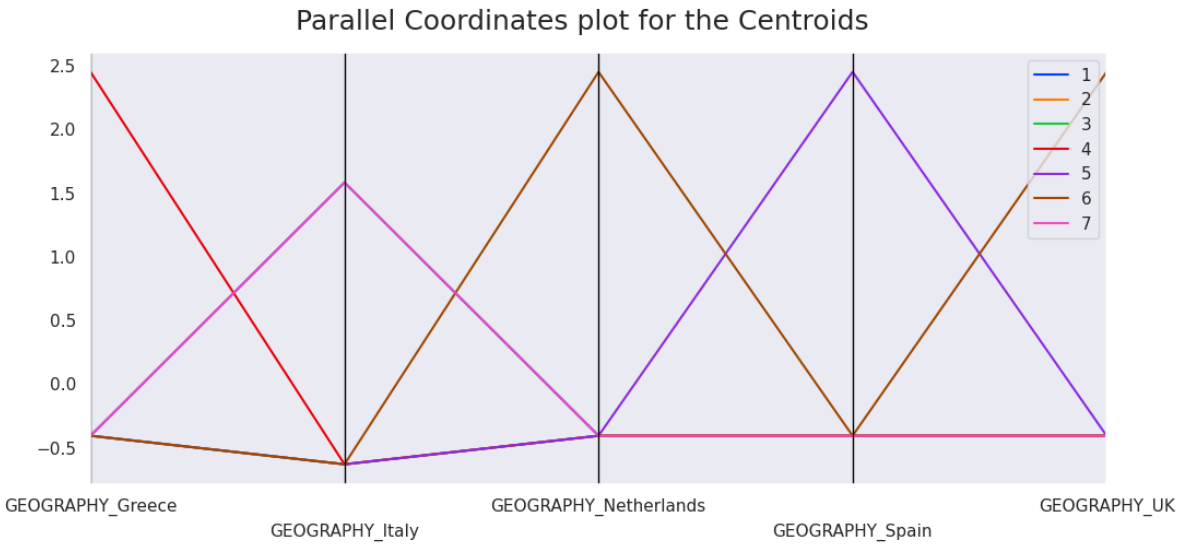
In [ ]:
```
def display_parallel_coordinates_plot(X_tr: pd.DataFrame, cluster_col: str):

    #Removing unclustered datapoints
    df_mean = X_tr.drop(X_tr[X_tr[cluster_col] == -1].index)
    #Aggregating by cluster and calculating the means of each fields
    df_mean = X_tr.groupby(cluster_col).mean()

    #Rescaling data for better visualization
    df_mean_scaled = StandardScaler().fit_transform(df_mean)

    #Transforming our data back into a dataframe
    df_mean = pd.DataFrame(df_mean_scaled, columns=df_mean.columns.to_list()).rese
    df_mean.rename(columns={"index": "cluster"}, inplace=True)

    #Adding 1 to the number of cluster just to have numbers from 1 to 7
    df_mean["cluster"] = df_mean["cluster"] + 1
    clusters = df_mean["cluster"]
    for k in range(1, len(X_tr.columns)//8+2):
        i = k*9
        if i > len(X_tr):
            subset = df_mean.iloc[:, -9:]
            subset["cluster"] = clusters
        elif k==1:
            subset = df_mean.iloc[:,(i-9):i]
        else:
            subset = df_mean.iloc[:,(i-9):i]
            subset["cluster"] = clusters
        display_parallel_coordinates_centroids(subset, 7)
        plt.show()
```

In [ ]:
```
X_tr = preprocess_pipe.fit_transform(X)

km = KMeans(n_clusters=7, n_init='auto')
km_clusters= km.fit_predict(X_tr)


cols = list(X.drop(X.columns[list(preprocess_pipe[-2].correlated_cols)], axis=1).c
X_tr = pd.DataFrame(X_tr, columns = cols)
X_tr['km_cluster'] = km_clusters
```

In [ ]:
```
display_parallel_coordinates_plot(X_tr, 'km_cluster')
```







4.4 DBSCAN

4.4.1 Hyperparameter tuning

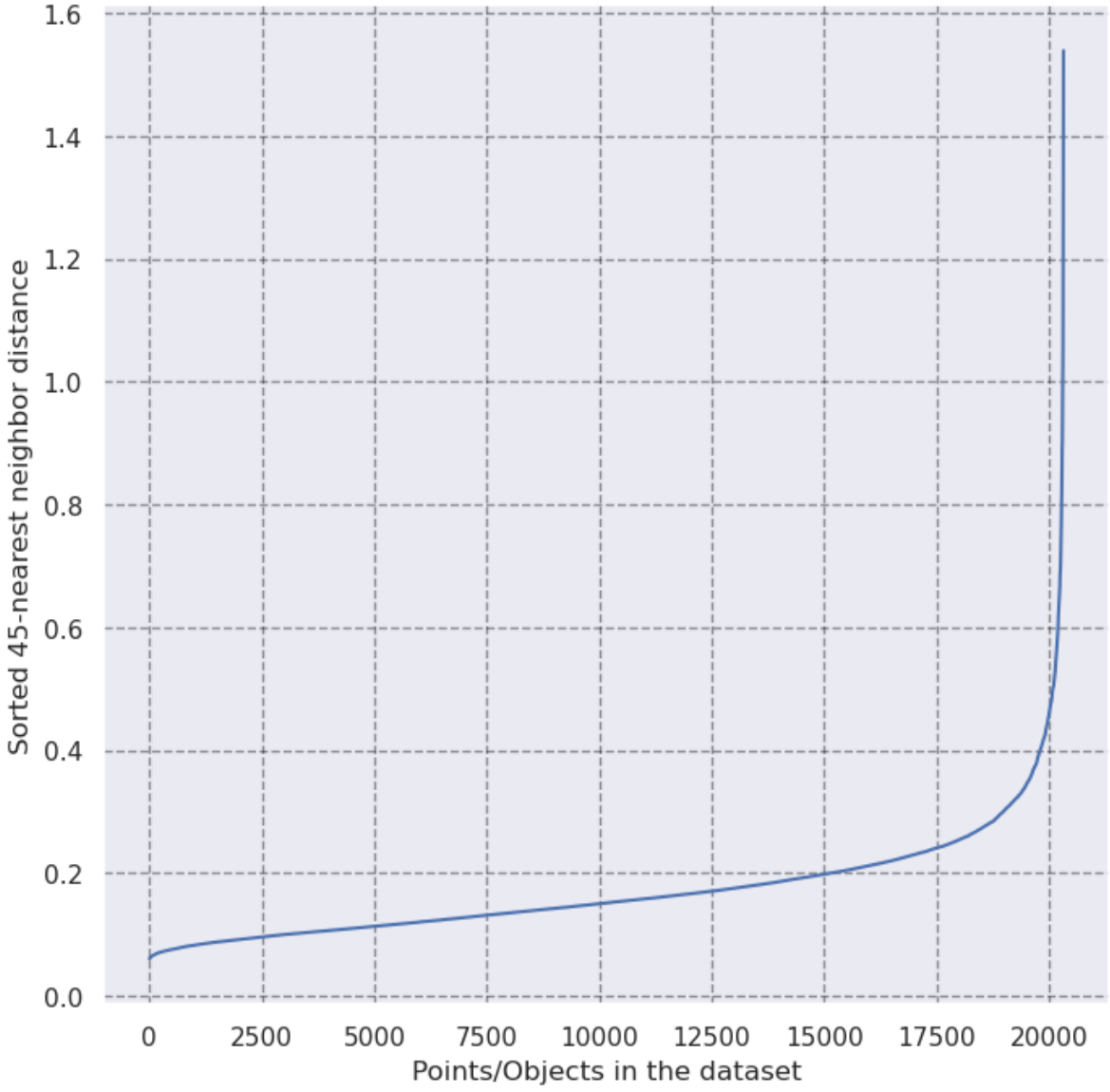
To tune the hyperparameter of the DBSCAN algorithm, we will use a starting point the heuristics defined in the latest DBSCAN paper (https://www.ccs.neu.edu/home/vip/teach/DMcourse/2_cluster_EM_mixt/notes_slides/revisitofre
For setting eps, it is advised to perform kdist plotting using the NearestNeighbors algorithm and to set :

k = 2*number of features -1

For min_samples, the starting point should generally be twice the number of features, or 46 in that case.

In [ ]:
```
#Defining kdist plot
from sklearn.neighbors import NearestNeighbors

def get_kdist_plot(X=None, k=None, radius_nbrs=1.0):

    nbrs = NearestNeighbors(n_neighbors=k, radius=radius_nbrs).fit(X)

    # For each point, compute distances to its k-nearest neighbors
    distances, indices = nbrs.kneighbors(X)
```

```
    distances = np.sort(distances, axis=0)
    distances = distances[:, k-1]

    # Plot the sorted K-nearest neighbor distance for each point in the dataset
    plt.figure(figsize=(8,8))
    plt.plot(distances)
    plt.xlabel('Points/Objects in the dataset', fontsize=12)
    plt.ylabel('Sorted {}-nearest neighbor distance'.format(k), fontsize=12)
    plt.grid(True, linestyle="--", color='black', alpha=0.4)
    plt.show()
    plt.close()

#Defining K as mentioned in the article
k = 2 * X_tr.shape[-1] - 1 # k=2*{dim(dataset)} - 1
get_kdist_plot(X=X_train_tr, k=k)
```



We see in the kdist plot above that the elbow is at around 0.2, so we will look for an eps at around 0.2

In [ ]:
```
#Specify parameters to sample from
param_grid = list(ParameterGrid(
    {
    'min_samples': [10,50,100,200,300,400,500,700],
    'eps' : [0.2,0.25,0.3,0.37,0.45,0.55,0.6, 0.65, 0.75, 0.85, 0.95],
    'metric': ['euclidean','manhattan']
```

```
}))

scores = []
kf = KFold(n_splits=3)
SIL = []
i = 0
#Performing manual gridsearch
for params in param_grid:

    labels = []
    indices = []
    scores_stab = []
    SIL = []

    for train_index, test_index in kf.split(X_train_tr):
        i+=1
        if i % 10 == 0:
            print("{}th iteration".format(i))
        X_train = X_train_tr[train_index]
        X_test = X_train_tr[test_index]

        dbs = DBSCAN(n_jobs=-1,
                     min_samples=params['min_samples'],
                     eps=params['eps'],
                     metric=params['metric']).fit(X_train)
        clusters = dbs.labels_
        if len(np.unique(clusters)) > 1:
            sil = silhouette_score(X_train, clusters)
        else:
            sil = 0
        SIL.append(sil)

        #Calculating stability on the train set
        indices.append(train_index)
        relabel = -np.ones(X_train_tr.shape[0], dtype=np.int32)
        relabel[train_index] = clusters
        labels.append(relabel)

        min_samples_pct = params['min_samples'] / len(X_train)

        gc.enable()
        del X_train, X_test
        gc.collect()

    for k, l in zip(labels, indices):
        for m, j in zip(labels, indices):
        # we also compute the diagonal
            in_both = np.intersect1d(l, j)
            scores_stab.append(adjusted_rand_score(k[in_both], m[in_both]))
    print("min_samples: {}, eps: {}, metric: {} ==>  Silhouette: {}, Stability {}"
    params['min_samples'], params['eps'], params['metric'], np.mean(SIL), np.mean(
    scores.append({'min_samples': params['min_samples'], 'min_samples_pct': min_sa
                   'metric': params['metric'],'Silhouette': np.mean(SIL), 'Stabili

scores_dbs = pd.DataFrame(scores)
```

```
min_samples: 10, eps: 0.2, metric: euclidean ==>  Silhouette: 0.38134778158869814,
Stability 0.9920014439208704
min_samples: 50, eps: 0.2, metric: euclidean ==>  Silhouette: 0.24648134671033828,
Stability 0.95949091407407
min_samples: 100, eps: 0.2, metric: euclidean ==>  Silhouette: 0.1052210380075511
7, Stability 0.926277917355204
10th iteration
min_samples: 200, eps: 0.2, metric: euclidean ==>  Silhouette: 0.02790234793647711
6, Stability 0.9608240732452269
min_samples: 300, eps: 0.2, metric: euclidean ==>  Silhouette: 0.00059506678178019
21, Stability 0.9082881513443863
min_samples: 400, eps: 0.2, metric: euclidean ==>  Silhouette: -0.0149778676247780
5, Stability 0.9793575445717396
20th iteration
min_samples: 500, eps: 0.2, metric: euclidean ==>  Silhouette: -0.0207830289900127
06, Stability 0.9719443450585928
min_samples: 700, eps: 0.2, metric: euclidean ==>  Silhouette: 0.0, Stability 1.0
min_samples: 10, eps: 0.2, metric: manhattan ==>  Silhouette: -0.1504799392340982
2, Stability 0.8959281181543565
30th iteration
min_samples: 50, eps: 0.2, metric: manhattan ==>  Silhouette: -0.0938303165353088
1, Stability 0.9522280398796654
min_samples: 100, eps: 0.2, metric: manhattan ==>  Silhouette: -0.0626388879057037
2, Stability 0.9647951285145102
min_samples: 200, eps: 0.2, metric: manhattan ==>  Silhouette: -0.0233461561520774
13, Stability 0.5555555555555556
min_samples: 300, eps: 0.2, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
40th iteration
min_samples: 400, eps: 0.2, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 500, eps: 0.2, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 700, eps: 0.2, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
50th iteration
min_samples: 10, eps: 0.25, metric: euclidean ==>  Silhouette: 0.3982836263677331,
Stability 0.9970550543555705
min_samples: 50, eps: 0.25, metric: euclidean ==>  Silhouette: 0.3650043831948260
3, Stability 0.9887607675109934
min_samples: 100, eps: 0.25, metric: euclidean ==>  Silhouette: 0.2915979739601985
4, Stability 0.9705232485064607
60th iteration
min_samples: 200, eps: 0.25, metric: euclidean ==>  Silhouette: 0.1409301975520576
3, Stability 0.9727441564529086
min_samples: 300, eps: 0.25, metric: euclidean ==>  Silhouette: 0.085933661959556
2, Stability 0.9768741399291151
min_samples: 400, eps: 0.25, metric: euclidean ==>  Silhouette: 0.0233044574742522
74, Stability 0.9713794069135733
min_samples: 500, eps: 0.25, metric: euclidean ==>  Silhouette: 0.073988241045794
2, Stability 0.9925693951187488
70th iteration
min_samples: 700, eps: 0.25, metric: euclidean ==>  Silhouette: 0.0041334791395695
155, Stability 0.9939271488993567
min_samples: 10, eps: 0.25, metric: manhattan ==>  Silhouette: 0.04970824244888897
4, Stability 0.9171695297672338
min_samples: 50, eps: 0.25, metric: manhattan ==>  Silhouette: -0.0629518815782062
9, Stability 0.9442693060596241
80th iteration
min_samples: 100, eps: 0.25, metric: manhattan ==>  Silhouette: -0.084460231826476
59, Stability 0.974585096037371
min_samples: 200, eps: 0.25, metric: manhattan ==>  Silhouette: -0.002407012074410
4956, Stability 0.9706368767157814
min_samples: 300, eps: 0.25, metric: manhattan ==>  Silhouette: -0.036999360775915
47, Stability 0.5262985291216826
90th iteration
min_samples: 400, eps: 0.25, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 500, eps: 0.25, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
```

```
min_samples: 700, eps: 0.25, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 10, eps: 0.3, metric: euclidean ==>  Silhouette: 0.4039953369441629,
Stability 0.998112421136881
100th iteration
min_samples: 50, eps: 0.3, metric: euclidean ==>  Silhouette: 0.3964786993772072,
Stability 0.9967032699612643
min_samples: 100, eps: 0.3, metric: euclidean ==>  Silhouette: 0.3674653419032725,
Stability 0.9904813371697103
min_samples: 200, eps: 0.3, metric: euclidean ==>  Silhouette: 0.2552507566808017,
Stability 0.9223478170454708
110th iteration
min_samples: 300, eps: 0.3, metric: euclidean ==>  Silhouette: 0.1533044667173298
6, Stability 0.9608065122715735
min_samples: 400, eps: 0.3, metric: euclidean ==>  Silhouette: 0.1310258422197556
3, Stability 0.9588476810263656
min_samples: 500, eps: 0.3, metric: euclidean ==>  Silhouette: 0.0786189020048599,
Stability 0.9881572684396135
120th iteration
min_samples: 700, eps: 0.3, metric: euclidean ==>  Silhouette: 0.0815704094073396,
Stability 0.9956280861830132
min_samples: 10, eps: 0.3, metric: manhattan ==>  Silhouette: 0.2608630914338302,
Stability 0.9567454654275068
min_samples: 50, eps: 0.3, metric: manhattan ==>  Silhouette: 0.05514733587112183,
Stability 0.9385612642003941
min_samples: 100, eps: 0.3, metric: manhattan ==>  Silhouette: 0.00571953351802549
2, Stability 0.9527084938884379
130th iteration
min_samples: 200, eps: 0.3, metric: manhattan ==>  Silhouette: -0.0028118712770695
124, Stability 0.9800405435342948
min_samples: 300, eps: 0.3, metric: manhattan ==>  Silhouette: 0.0293414216525442
9, Stability 0.9834642798036506
min_samples: 400, eps: 0.3, metric: manhattan ==>  Silhouette: -0.0321526011395107
95, Stability 0.9731302136594228
140th iteration
min_samples: 500, eps: 0.3, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 700, eps: 0.3, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 10, eps: 0.37, metric: euclidean ==>  Silhouette: 0.4080305701103566,
Stability 0.9989935765486844
150th iteration
min_samples: 50, eps: 0.37, metric: euclidean ==>  Silhouette: 0.4062204484128355
6, Stability 0.9986085034676354
min_samples: 100, eps: 0.37, metric: euclidean ==>  Silhouette: 0.4011968714920570
7, Stability 0.9979363456665461
min_samples: 200, eps: 0.37, metric: euclidean ==>  Silhouette: 0.3573391553829447
5, Stability 0.9823173575900791
min_samples: 300, eps: 0.37, metric: euclidean ==>  Silhouette: 0.2775053867149538
4, Stability 0.9891442636868477
160th iteration
min_samples: 400, eps: 0.37, metric: euclidean ==>  Silhouette: 0.2242366558505567
7, Stability 0.9810788048562669
min_samples: 500, eps: 0.37, metric: euclidean ==>  Silhouette: 0.2101430032520746
6, Stability 0.9899072662601848
min_samples: 700, eps: 0.37, metric: euclidean ==>  Silhouette: 0.1176632121117721
8, Stability 0.9248159519478875
170th iteration
min_samples: 10, eps: 0.37, metric: manhattan ==>  Silhouette: 0.3403495639576404,
Stability 0.9810157970902303
min_samples: 50, eps: 0.37, metric: manhattan ==>  Silhouette: 0.1836162989361372,
Stability 0.9378423333503874
min_samples: 100, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0914291391128570
5, Stability 0.9457166969691184
180th iteration
min_samples: 200, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0099520206579670
84, Stability 0.9670071042542049
```

```
min_samples: 300, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0201487791329777
63, Stability 0.9838758295403649
min_samples: 400, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0547647186250924
6, Stability 0.989068049829227
min_samples: 500, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0138477657798703
07, Stability 0.910904441032574
190th iteration
min_samples: 700, eps: 0.37, metric: manhattan ==>  Silhouette: 0.0, Stability 1.0
min_samples: 10, eps: 0.45, metric: euclidean ==>  Silhouette: 0.4092583587805538,
Stability 0.9995098529963923
min_samples: 50, eps: 0.45, metric: euclidean ==>  Silhouette: 0.4088902453709659,
Stability 0.9996998365123666
200th iteration
min_samples: 100, eps: 0.45, metric: euclidean ==>  Silhouette: 0.4079561313860804
5, Stability 0.9996007222399506
min_samples: 200, eps: 0.45, metric: euclidean ==>  Silhouette: 0.4003570198947214
4, Stability 0.9983077659415414
min_samples: 300, eps: 0.45, metric: euclidean ==>  Silhouette: 0.367147127312830
2, Stability 0.9947660679433254
210th iteration
min_samples: 400, eps: 0.45, metric: euclidean ==>  Silhouette: 0.2952540305746748
6, Stability 0.9874497259990606
min_samples: 500, eps: 0.45, metric: euclidean ==>  Silhouette: 0.252607541514754
9, Stability 0.926408744486261
min_samples: 700, eps: 0.45, metric: euclidean ==>  Silhouette: 0.228151219074352
8, Stability 0.9910743124552636
min_samples: 10, eps: 0.45, metric: manhattan ==>  Silhouette: 0.3794294542043699
6, Stability 0.9908235911942376
220th iteration
min_samples: 50, eps: 0.45, metric: manhattan ==>  Silhouette: 0.3061447435862596
5, Stability 0.9734313871793563
min_samples: 100, eps: 0.45, metric: manhattan ==>  Silhouette: 0.2086731886736888
6, Stability 0.9299612774131449
min_samples: 200, eps: 0.45, metric: manhattan ==>  Silhouette: 0.1214989733645591
8, Stability 0.9767484490042904
230th iteration
min_samples: 300, eps: 0.45, metric: manhattan ==>  Silhouette: 0.0611988950005060
5, Stability 0.979432701855139
min_samples: 400, eps: 0.45, metric: manhattan ==>  Silhouette: 0.0519194728019928
3, Stability 0.9729744848146381
min_samples: 500, eps: 0.45, metric: manhattan ==>  Silhouette: 0.0744057842309806
8, Stability 0.9969264931516675
240th iteration
min_samples: 700, eps: 0.45, metric: manhattan ==>  Silhouette: 0.0381330481712395
74, Stability 0.96653321995092
min_samples: 10, eps: 0.55, metric: euclidean ==>  Silhouette: 0.535016340762694,
Stability 0.9995590127462335
min_samples: 50, eps: 0.55, metric: euclidean ==>  Silhouette: 0.5346635110292332,
Stability 0.9996599122142378
min_samples: 100, eps: 0.55, metric: euclidean ==>  Silhouette: 0.534302046610914
4, Stability 0.9996861022151032
250th iteration
min_samples: 200, eps: 0.55, metric: euclidean ==>  Silhouette: 0.533372243080253
1, Stability 0.9997199515705728
min_samples: 300, eps: 0.55, metric: euclidean ==>  Silhouette: 0.518951353492529
9, Stability 0.996982375891034
min_samples: 400, eps: 0.55, metric: euclidean ==>  Silhouette: 0.4856527627183499
5, Stability 0.9986415309431236
260th iteration
min_samples: 500, eps: 0.55, metric: euclidean ==>  Silhouette: 0.4880768969774884
6, Stability 0.9971645968483516
min_samples: 700, eps: 0.55, metric: euclidean ==>  Silhouette: 0.355198259251008
1, Stability 0.9278486168742273
min_samples: 10, eps: 0.55, metric: manhattan ==>  Silhouette: 0.5093191267007352,
```

```
Stability 0.9964140283170259
270th iteration
min_samples: 50, eps: 0.55, metric: manhattan ==>  Silhouette: 0.4729877817889865
4, Stability 0.9918778654924105
min_samples: 100, eps: 0.55, metric: manhattan ==>  Silhouette: 0.414945897340459
1, Stability 0.9871435604018988
min_samples: 200, eps: 0.55, metric: manhattan ==>  Silhouette: 0.283610979901065
5, Stability 0.9791031025327493
min_samples: 300, eps: 0.55, metric: manhattan ==>  Silhouette: 0.2052542020304497
5, Stability 0.9754836324543016
280th iteration
min_samples: 400, eps: 0.55, metric: manhattan ==>  Silhouette: 0.0907781390575760
5, Stability 0.9674628672279554
min_samples: 500, eps: 0.55, metric: manhattan ==>  Silhouette: 0.1006579710631088
8, Stability 0.9836884571383833
min_samples: 700, eps: 0.55, metric: manhattan ==>  Silhouette: 0.090138878962276
2, Stability 0.9894757295309423
290th iteration
min_samples: 10, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5351642020694914,
Stability 0.9996578811111547
min_samples: 50, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5350759624219149,
Stability 0.9998303121534298
min_samples: 100, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5348752950229257,
Stability 0.9996877823952839
300th iteration
min_samples: 200, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5343718796770794,
Stability 0.9997555147967686
min_samples: 300, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5326087529604485,
Stability 0.9995890552872938
min_samples: 400, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5228783271874636,
Stability 0.9991879450997396
min_samples: 500, eps: 0.6, metric: euclidean ==>  Silhouette: 0.5045159999120745,
Stability 0.9941656610843642
310th iteration
min_samples: 700, eps: 0.6, metric: euclidean ==>  Silhouette: 0.4568151712742767
3, Stability 0.9944203231951396
min_samples: 10, eps: 0.6, metric: manhattan ==>  Silhouette: 0.5147547382998995,
Stability 0.9969842288880295
min_samples: 50, eps: 0.6, metric: manhattan ==>  Silhouette: 0.4958494134807528,
Stability 0.9939680407043981
320th iteration
min_samples: 100, eps: 0.6, metric: manhattan ==>  Silhouette: 0.447447303365633,
Stability 0.9888118724598836
min_samples: 200, eps: 0.6, metric: manhattan ==>  Silhouette: 0.3180873423741817
4, Stability 0.9760447048617146
min_samples: 300, eps: 0.6, metric: manhattan ==>  Silhouette: 0.2420701171534222
2, Stability 0.9746973653751895
330th iteration
min_samples: 400, eps: 0.6, metric: manhattan ==>  Silhouette: 0.1879248550761035
4, Stability 0.9541878600041355
min_samples: 500, eps: 0.6, metric: manhattan ==>  Silhouette: 0.0966271524951710
5, Stability 0.9723329299623037
min_samples: 700, eps: 0.6, metric: manhattan ==>  Silhouette: 0.0880404679596884
1, Stability 0.8918901754250427
min_samples: 10, eps: 0.65, metric: euclidean ==>  Silhouette: 0.5352247063017351,
Stability 0.9997512802905989
340th iteration
min_samples: 50, eps: 0.65, metric: euclidean ==>  Silhouette: 0.5352138451166903,
Stability 0.9998349112471742
min_samples: 100, eps: 0.65, metric: euclidean ==>  Silhouette: 0.535163848865060
4, Stability 0.9997966682716792
min_samples: 200, eps: 0.65, metric: euclidean ==>  Silhouette: 0.534915225454514
5, Stability 0.9999878438328828
350th iteration
```

```
min_samples: 300, eps: 0.65, metric: euclidean ==>  Silhouette: 0.534346711856222
5, Stability 0.9998999145017399
min_samples: 400, eps: 0.65, metric: euclidean ==>  Silhouette: 0.529807505700298
1, Stability 0.9995314406663661
min_samples: 500, eps: 0.65, metric: euclidean ==>  Silhouette: 0.521725540526029
8, Stability 0.9993764235100898
360th iteration
min_samples: 700, eps: 0.65, metric: euclidean ==>  Silhouette: 0.486667452625391
4, Stability 0.9975524544853879
min_samples: 10, eps: 0.65, metric: manhattan ==>  Silhouette: 0.5189026245950861,
Stability 0.9972289782566714
min_samples: 50, eps: 0.65, metric: manhattan ==>  Silhouette: 0.5064670017148066,
Stability 0.9961425234549094
min_samples: 100, eps: 0.65, metric: manhattan ==>  Silhouette: 0.4767168849023613
6, Stability 0.9914521047745127
370th iteration
min_samples: 200, eps: 0.65, metric: manhattan ==>  Silhouette: 0.358948943123570
1, Stability 0.975596082029796
min_samples: 300, eps: 0.65, metric: manhattan ==>  Silhouette: 0.3113293296338026
7, Stability 0.981007068525204
min_samples: 400, eps: 0.65, metric: manhattan ==>  Silhouette: 0.240602726994639
1, Stability 0.9849451653340751
380th iteration
min_samples: 500, eps: 0.65, metric: manhattan ==>  Silhouette: 0.199088092485942
6, Stability 0.9824009775901901
min_samples: 700, eps: 0.65, metric: manhattan ==>  Silhouette: 0.1180218579395195
8, Stability 0.9871430315250751
min_samples: 10, eps: 0.75, metric: euclidean ==>  Silhouette: 0.5353896713890319,
Stability 0.9999569952009715
390th iteration
min_samples: 50, eps: 0.75, metric: euclidean ==>  Silhouette: 0.5353960081663772,
Stability 0.9998764709064033
min_samples: 100, eps: 0.75, metric: euclidean ==>  Silhouette: 0.535361972607085
4, Stability 0.9998257042725318
min_samples: 200, eps: 0.75, metric: euclidean ==>  Silhouette: 0.535377834725872
4, Stability 0.9999089720858392
min_samples: 300, eps: 0.75, metric: euclidean ==>  Silhouette: 0.535206455719795
7, Stability 0.9998641967988833
400th iteration
min_samples: 400, eps: 0.75, metric: euclidean ==>  Silhouette: 0.534950657363472
7, Stability 0.9999003426432691
min_samples: 500, eps: 0.75, metric: euclidean ==>  Silhouette: 0.533242921372956
9, Stability 0.999663125202294
min_samples: 700, eps: 0.75, metric: euclidean ==>  Silhouette: 0.527870965248789
5, Stability 0.9980164376840983
410th iteration
min_samples: 10, eps: 0.75, metric: manhattan ==>  Silhouette: 0.5249449267988474,
Stability 0.9980093219613799
min_samples: 50, eps: 0.75, metric: manhattan ==>  Silhouette: 0.5180263187574876,
Stability 0.9972332394748366
min_samples: 100, eps: 0.75, metric: manhattan ==>  Silhouette: 0.508465061804627
9, Stability 0.9965679949459118
420th iteration
min_samples: 200, eps: 0.75, metric: manhattan ==>  Silhouette: 0.454919011636668
3, Stability 0.9933610776268288
min_samples: 300, eps: 0.75, metric: manhattan ==>  Silhouette: 0.368336051610334,
Stability 0.9752084608913221
min_samples: 400, eps: 0.75, metric: manhattan ==>  Silhouette: 0.3375370104105825
4, Stability 0.9876837454479452
min_samples: 500, eps: 0.75, metric: manhattan ==>  Silhouette: 0.2568052369133569
4, Stability 0.9561564120838209
430th iteration
min_samples: 700, eps: 0.75, metric: manhattan ==>  Silhouette: 0.227537223274369,
Stability 0.9894573256698672
```

```
min_samples: 10, eps: 0.85, metric: euclidean ==>  Silhouette: 0.5354934218660132,
Stability 1.0
min_samples: 50, eps: 0.85, metric: euclidean ==>  Silhouette: 0.5354934218660132,
Stability 1.0
440th iteration
min_samples: 100, eps: 0.85, metric: euclidean ==>  Silhouette: 0.535493421866013
2, Stability 1.0
min_samples: 200, eps: 0.85, metric: euclidean ==>  Silhouette: 0.535493421866013
2, Stability 1.0
min_samples: 300, eps: 0.85, metric: euclidean ==>  Silhouette: 0.535493421866013
2, Stability 1.0
450th iteration
min_samples: 400, eps: 0.85, metric: euclidean ==>  Silhouette: 0.535309489556056
8, Stability 0.9999359935437886
min_samples: 500, eps: 0.85, metric: euclidean ==>  Silhouette: 0.535073872764790
2, Stability 0.9999714015864972
min_samples: 700, eps: 0.85, metric: euclidean ==>  Silhouette: 0.532875699339082
7, Stability 0.9998424895324131
min_samples: 10, eps: 0.85, metric: manhattan ==>  Silhouette: 0.5287676132770411,
Stability 0.9992124123349583
460th iteration
min_samples: 50, eps: 0.85, metric: manhattan ==>  Silhouette: 0.5248655096055622,
Stability 0.9987467476420152
min_samples: 100, eps: 0.85, metric: manhattan ==>  Silhouette: 0.520228601994507,
Stability 0.9977703965132173
min_samples: 200, eps: 0.85, metric: manhattan ==>  Silhouette: 0.4951045312756368
6, Stability 0.9973006656701112
470th iteration
min_samples: 300, eps: 0.85, metric: manhattan ==>  Silhouette: 0.4568341087209264
5, Stability 0.9921624085211354
min_samples: 400, eps: 0.85, metric: manhattan ==>  Silhouette: 0.3876037848248083
4, Stability 0.9909384518208244
min_samples: 500, eps: 0.85, metric: manhattan ==>  Silhouette: 0.3584991548604918
3, Stability 0.9896293717689573
480th iteration
min_samples: 700, eps: 0.85, metric: manhattan ==>  Silhouette: 0.259177569281299
7, Stability 0.9372652744727988
min_samples: 10, eps: 0.95, metric: euclidean ==>  Silhouette: 0.5354923621896042,
Stability 0.9999658055924486
min_samples: 50, eps: 0.95, metric: euclidean ==>  Silhouette: 0.5354923621896042,
Stability 0.9999658055924486
min_samples: 100, eps: 0.95, metric: euclidean ==>  Silhouette: 0.535492362189604
2, Stability 0.9999658055924486
490th iteration
min_samples: 200, eps: 0.95, metric: euclidean ==>  Silhouette: 0.535497026046217
4, Stability 0.999979827350568
min_samples: 300, eps: 0.95, metric: euclidean ==>  Silhouette: 0.535514680020167,
Stability 1.0
min_samples: 400, eps: 0.95, metric: euclidean ==>  Silhouette: 0.535514680020167,
Stability 1.0
500th iteration
min_samples: 500, eps: 0.95, metric: euclidean ==>  Silhouette: 0.535415855003264
3, Stability 1.0
min_samples: 700, eps: 0.95, metric: euclidean ==>  Silhouette: 0.534866443732872
8, Stability 0.9999508088061108
min_samples: 10, eps: 0.95, metric: manhattan ==>  Silhouette: 0.5319269663806926,
Stability 0.9992169838852352
510th iteration
min_samples: 50, eps: 0.95, metric: manhattan ==>  Silhouette: 0.5302351986804559,
Stability 0.9989099108721852
min_samples: 100, eps: 0.95, metric: manhattan ==>  Silhouette: 0.527056499954278
4, Stability 0.9987657616044502
min_samples: 200, eps: 0.95, metric: manhattan ==>  Silhouette: 0.517847889174220
3, Stability 0.9984069584791099
```

```
min_samples: 300, eps: 0.95, metric: manhattan ==>  Silhouette: 0.491260961012112
2, Stability 0.9964193919611996
520th iteration
min_samples: 400, eps: 0.95, metric: manhattan ==>  Silhouette: 0.457523265137432
8, Stability 0.981771826545186
min_samples: 500, eps: 0.95, metric: manhattan ==>  Silhouette: 0.4027627798504216
4, Stability 0.9899110785207247
min_samples: 700, eps: 0.95, metric: manhattan ==>  Silhouette: 0.335634408203273
1, Stability 0.9210848341182987
```

In [ ]:
```
scores_dbs = scores_dbs.sort_values(by="Silhouette", ascending=False)

with open('data/dbscan_scores_v3.pkl','wb') as file:
    dill.dump(scores_dbs, file)

scores_dbs.head(10)
```

Out[ ]:

	min_samples	min_samples_pct	eps	metric	Silhouette	Stability
165	400	0.029516	0.95	euclidean	0.535515	1.000000
164	300	0.022137	0.95	euclidean	0.535515	1.000000
163	200	0.014758	0.95	euclidean	0.535497	0.999980
144	10	0.000738	0.85	euclidean	0.535493	1.000000
145	50	0.003689	0.85	euclidean	0.535493	1.000000
146	100	0.007379	0.85	euclidean	0.535493	1.000000
147	200	0.014758	0.85	euclidean	0.535493	1.000000
148	300	0.022137	0.85	euclidean	0.535493	1.000000
161	50	0.003689	0.95	euclidean	0.535492	0.999966
162	100	0.007379	0.95	euclidean	0.535492	0.999966

In [ ]:
```
with open('data/dbscan_scores_v3.pkl', 'rb') as file:
    scores_dbs = dill.load(file)
scores_dbs.head(5)
```

Out[ ]:

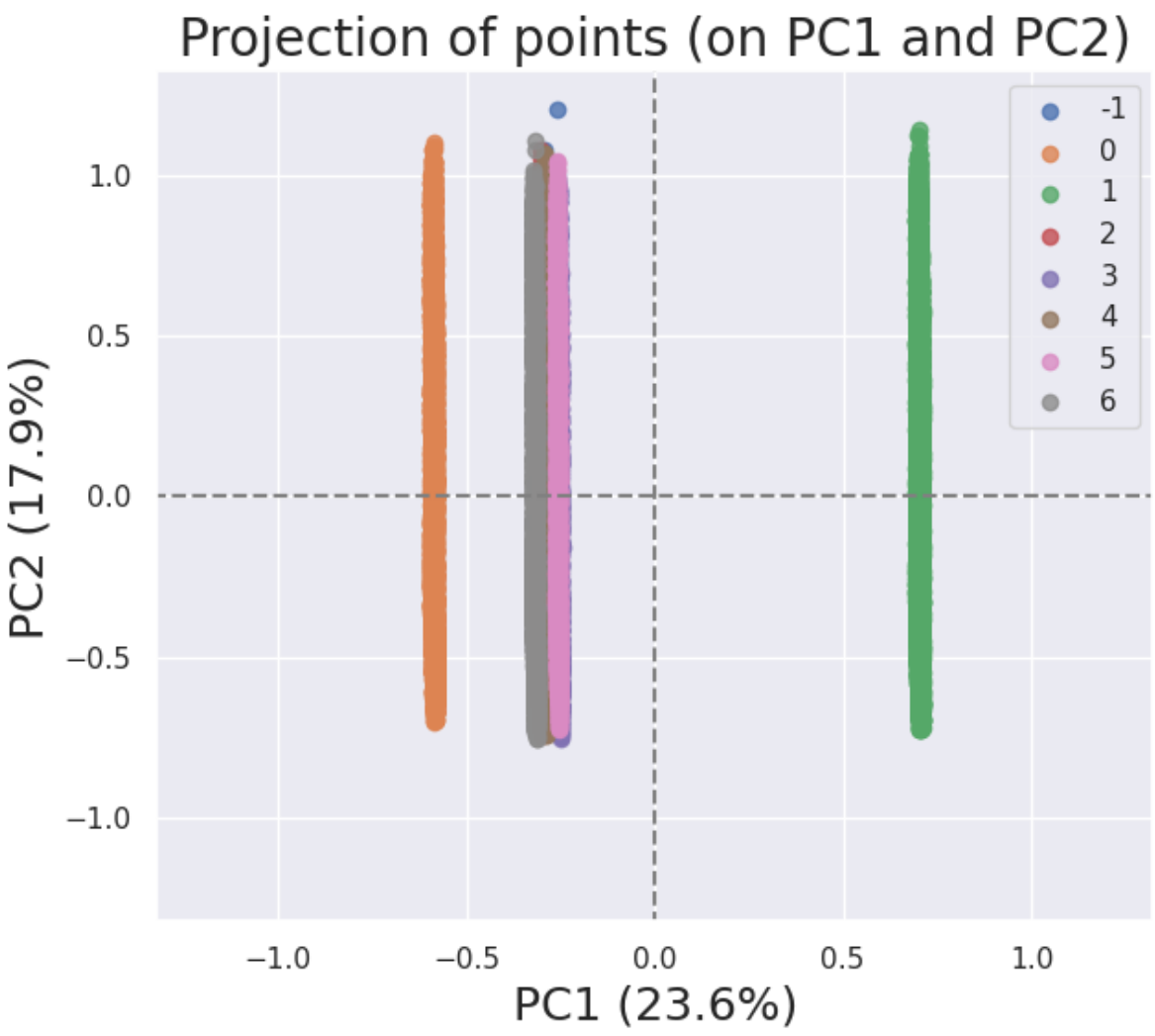
	min_samples	min_samples_pct	eps	metric	Silhouette	Stability
165	400	0.029516	0.95	euclidean	0.535515	1.00000
164	300	0.022137	0.95	euclidean	0.535515	1.00000
163	200	0.014758	0.95	euclidean	0.535497	0.99998
144	10	0.000738	0.85	euclidean	0.535493	1.00000
145	50	0.003689	0.85	euclidean	0.535493	1.00000

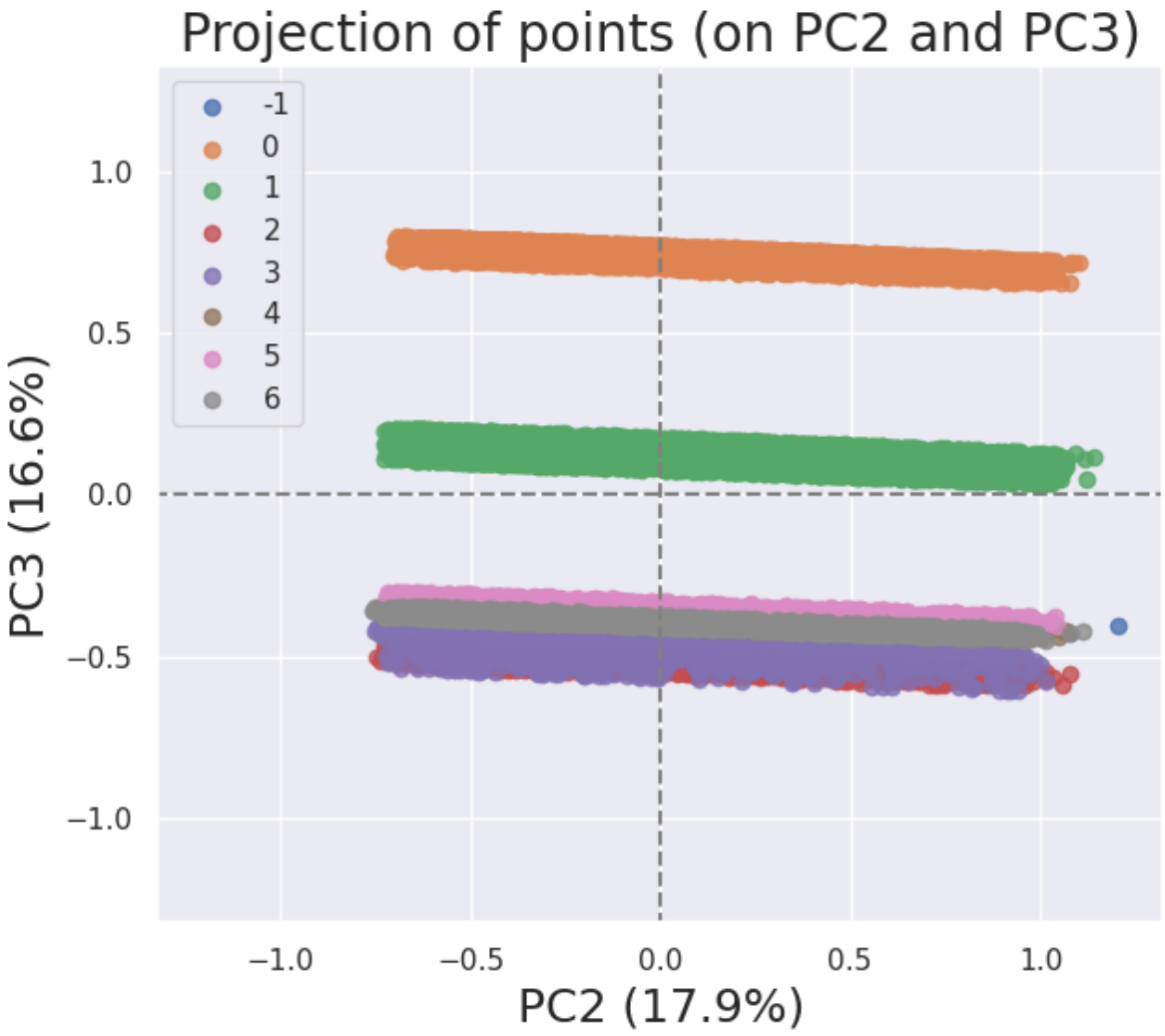
In [ ]:
```
#Defining our final DBSCAN model with optimized hyperparameters
X_tr = preprocess_pipe.fit_transform(X)
best_results = scores_dbs.head(1)
dbs_best = DBSCAN(n_jobs=-1,
                  min_samples=int(best_results.min_samples_pct.values[0]*len(X_tr)
                  eps=best_results.eps.values[0],
                  metric=best_results.metric.values[0]).fit(X_tr)
```

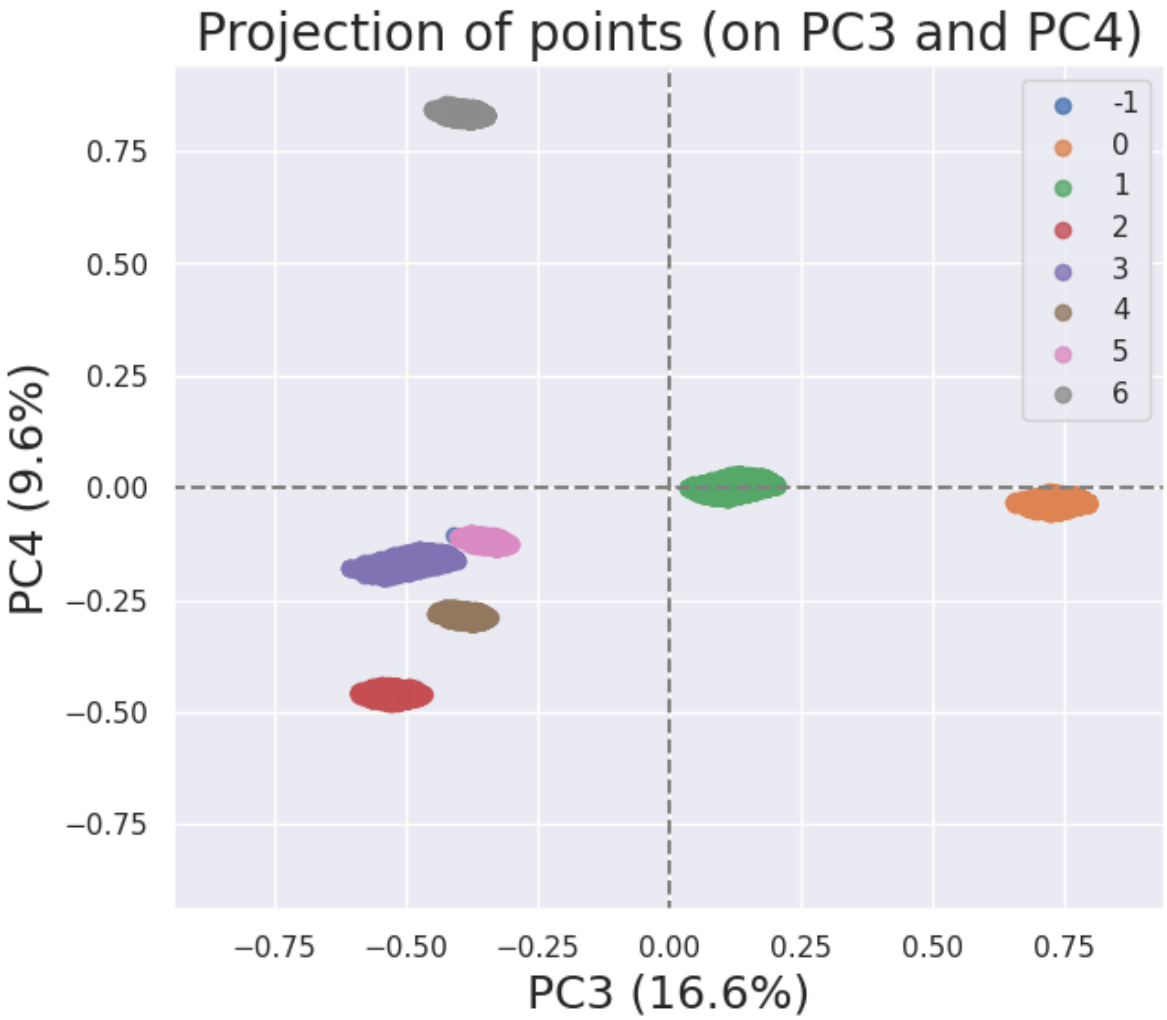
4.4.2 PCA Visualization

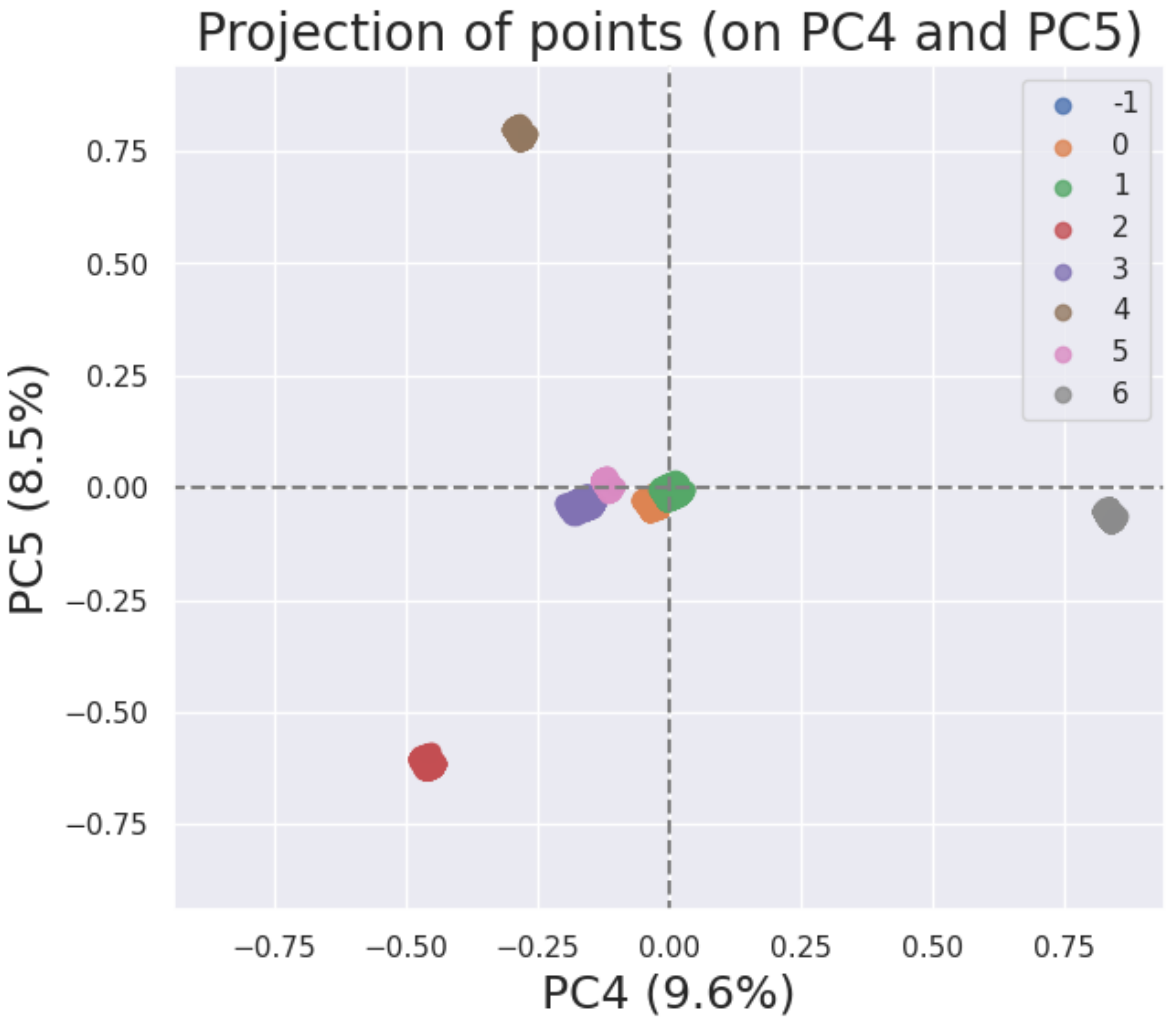
In [ ]:

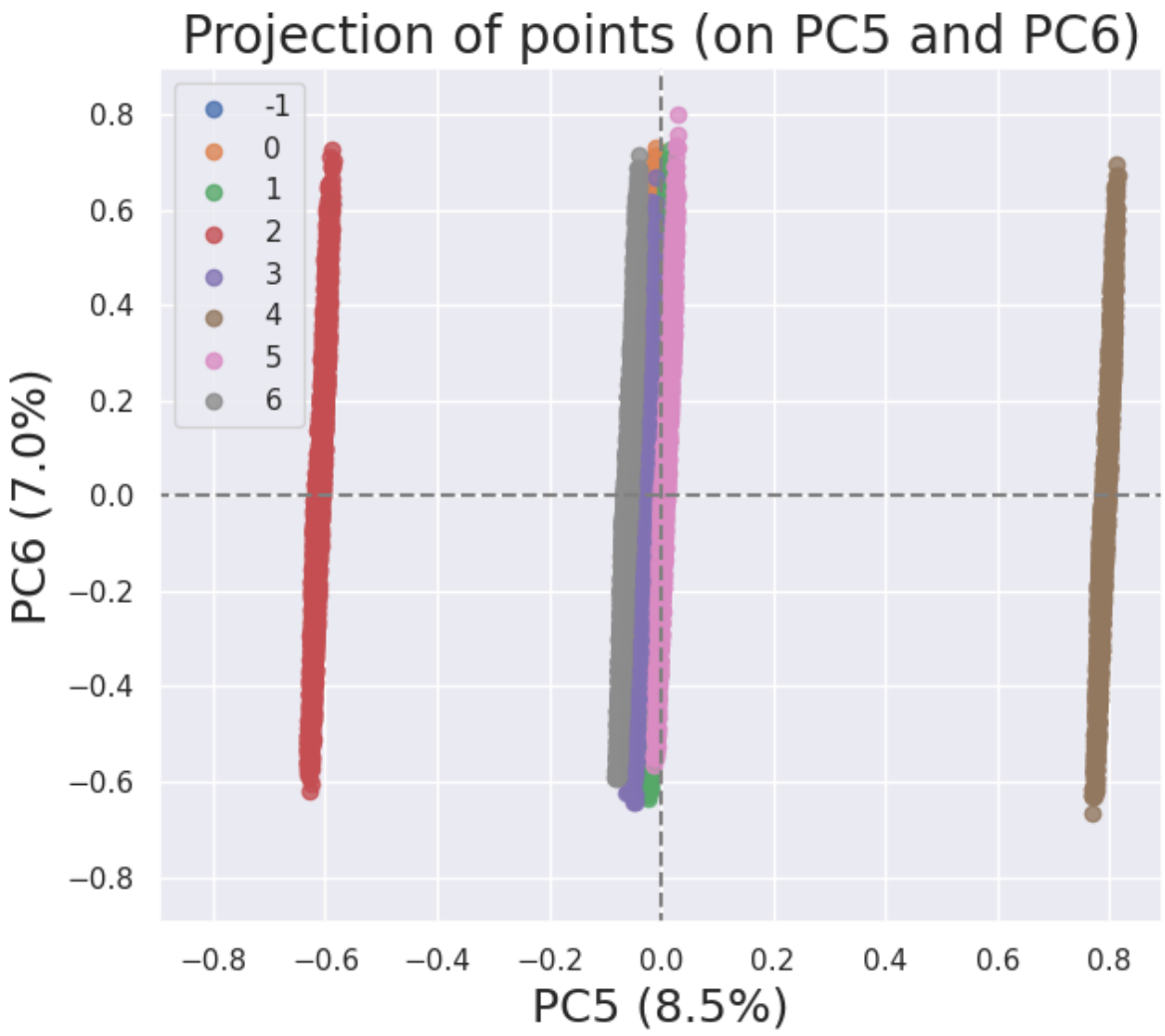
```
#Visualization with PCA
# Creating a PCA model to reduce our data for visualisation
n_components = 7
pca = PCA(n_components=n_components, n_oversamples=10)
pca.fit(X_tr)

#Transforming our initial dataset
X_red = pca.transform(X_tr)

clusters = dbs_best.labels_

for i in range(1,n_components):
    display_factorial_planes(X_red, n_components, pca, [(i-1,i)], illustrative_var

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

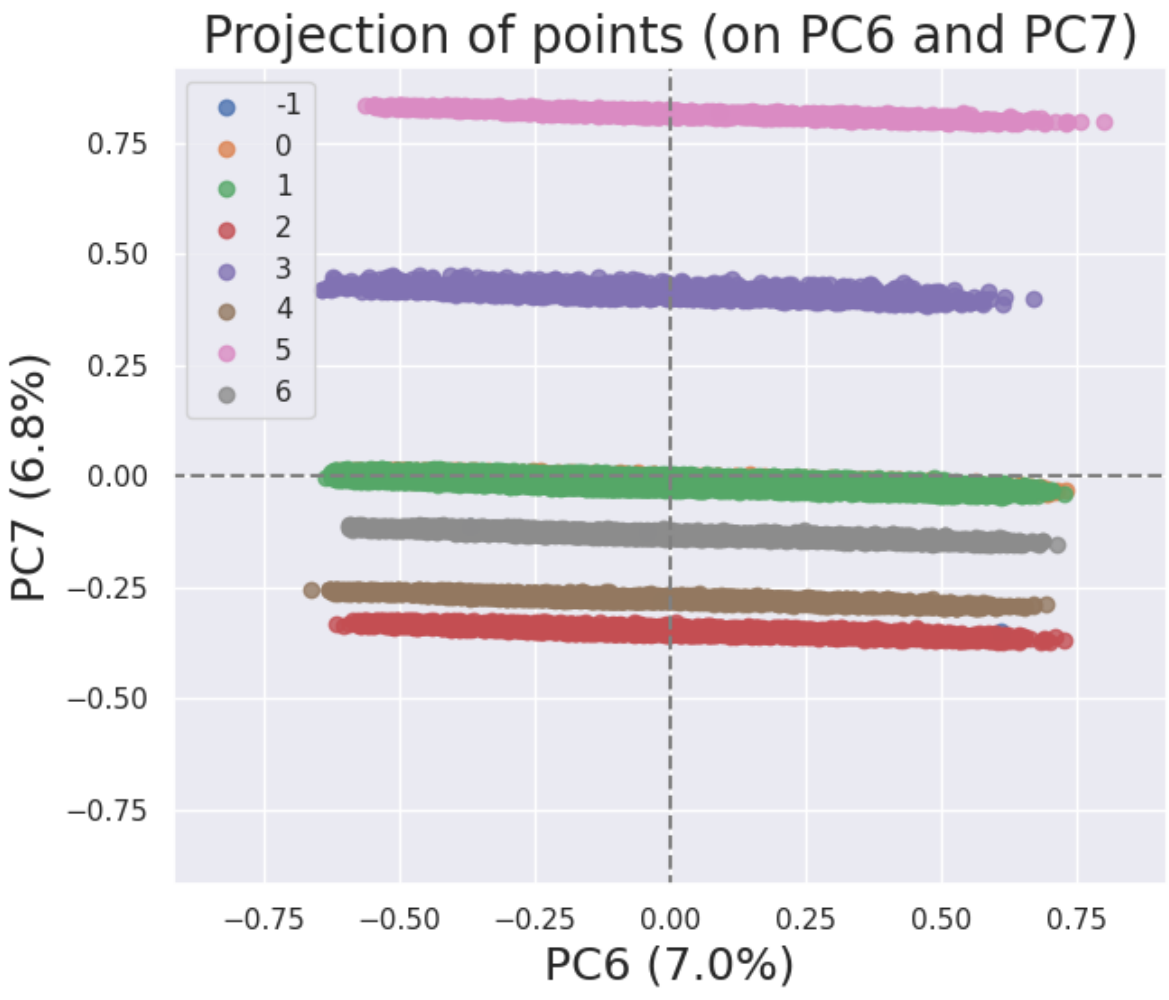



Projection of points (on PC2 and PC3)
-1
0
1
2
3
4
5
6
1.0
0.5
0.0
−0.5
−1.0
PC3 (16.6%)
−1.0
−0.5
0.0
0.5
1.0
PC2 (17.9%)


Projection of points (on PC3 and PC4)
PC4 (9.6%)
PC3 (16.6%)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
-1
0
1
2
3
4
5
6


Projection of points (on PC4 and PC5)
PC5 (8.5%)
PC4 (9.6%)
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
-1
0
1
2
3
4
5
6

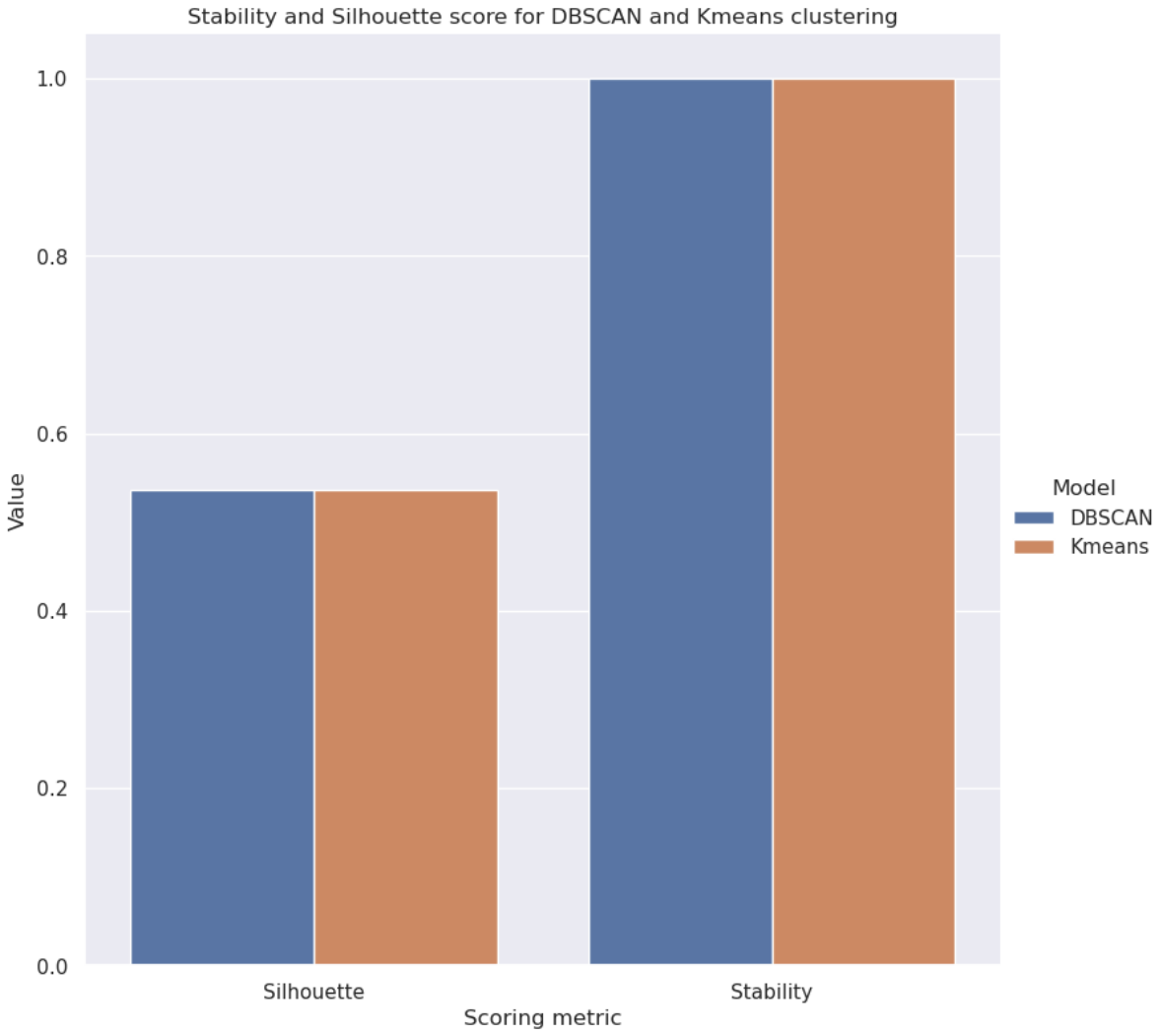

Projection of points (on PC5 and PC6)
-1
0
1
2
3
4
5
6
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
PC6 (7.0%)
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8
PC5 (8.5%)


Projection of points (on PC6 and PC7)
-1
0
1
2
3
4
5
6
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
PC7 (6.8%)
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
PC6 (7.0%)

In [ ]:
```
silhouette_score(X_tr, clusters)
```

Out[ ]:
```
0.5359952083171928
```

In [ ]:
```
results = pd.DataFrame([{"Model" : "DBSCAN", "Silhouette": 0.5359952083171928, "St
                        {"Model" : "Kmeans", "Silhouette": 0.5356087850017238, "Stab

results = pd.melt(results, id_vars='Model')

sns.set(rc={'figure.figsize':(20,15)})
sns.catplot(x='variable', y='value', hue='Model', data=results, kind='bar', height
plt.title("Stability and Silhouette score for DBSCAN and Kmeans clustering")
plt.xlabel("Scoring metric")
plt.ylabel("Value")
plt.show()
```



4.5 Final Model Performance

Let's compare both our models on the complete dataset:

In [ ]:
```
X_tr = preprocess_pipe.transform(X)

clusters = dbs_best.labels_

silhouette = silhouette_score(X_tr, clusters)
stability = cluster_stability(X_tr, dbs_best)
print("Silhouette score for DBSCAN on dataset: {:.4f}, Stability index : {:.3f}".f
```

```
km = KMeans(n_clusters=7, n_init=10)
km_clusters= km.fit_predict(X_tr)

silhouette = silhouette_score(X_tr, km_clusters)
stability = cluster_stability(X_tr, km)
print("Silhouette score for Kmeans on dataset: {:.4f}, Stability index : {:.3f}".f
```

```
Silhouette score for DBSCAN on dataset: 0.5360, Stability index : 1.000
Silhouette score for Kmeans on dataset: 0.5360, Stability index : 0.957
```

In [ ]:

```
X_tr['cluster'] = clusters

#Studying the repartition of clusters

sub = X_tr.cluster.value_counts()

fig = go.Figure(data=[go.Pie(values=sub, labels=sub.index)])
fig.update_layout(
title={
    'text': "Pie chart of the repartition of clusters",
},
title_x = 0.5,
font=dict(
    size=18,
    color="Black"
))
fig.show()
```

5 Describing our customer segments

In [ ]:
```
customers_df['cluster'] = clusters

km = KMeans(n_clusters=7, n_init=10)
km_clusters = km.fit_predict(X_tr)

customers_df['km_cluster'] = km_clusters
customers_df["customer_segment_num"] = 0
customers_df.loc[(customers_df.customer_segment_num ==0) & (customers_df['RFM_scor
customers_df.loc[(customers_df.customer_segment_num ==0) & (customers_df['RFM_scor
customers_df.loc[(customers_df.customer_segment_num ==0) & (customers_df['RFM_scor
customers_df.loc[(customers_df.customer_segment_num ==0) & (customers_df['RFM_scor
customers_df.loc[(customers_df.customer_segment_num ==0) & (customers_df['RFM_scor
customers_df
```

Out[ ]:

Customer_ID	GENDER	AGE	GEOGRAPHY	number_orders	order_value_min	order_value_max	tot
1	M	22	Italy	1	16.29	16.29	
2	M	24	Italy	2	7.77	15.00	
3	M	23	Italy	1	10.92	10.92	
4	M	27	Germany	2	6.69	26.60	
5	M	36	Germany	5	7.06	20.13	
...	...	...	...	...	...	...	
22621	F	50	Spain	1	9.69	9.69	
22622	F	40	France	1	6.07	6.07	
22623	F	41	France	1	128.01	128.01	
22624	M	26	Italy	1	19.60	19.60	
22625	M	36	Germany	4	1.11	59.50	

22586 rows × 34 columns

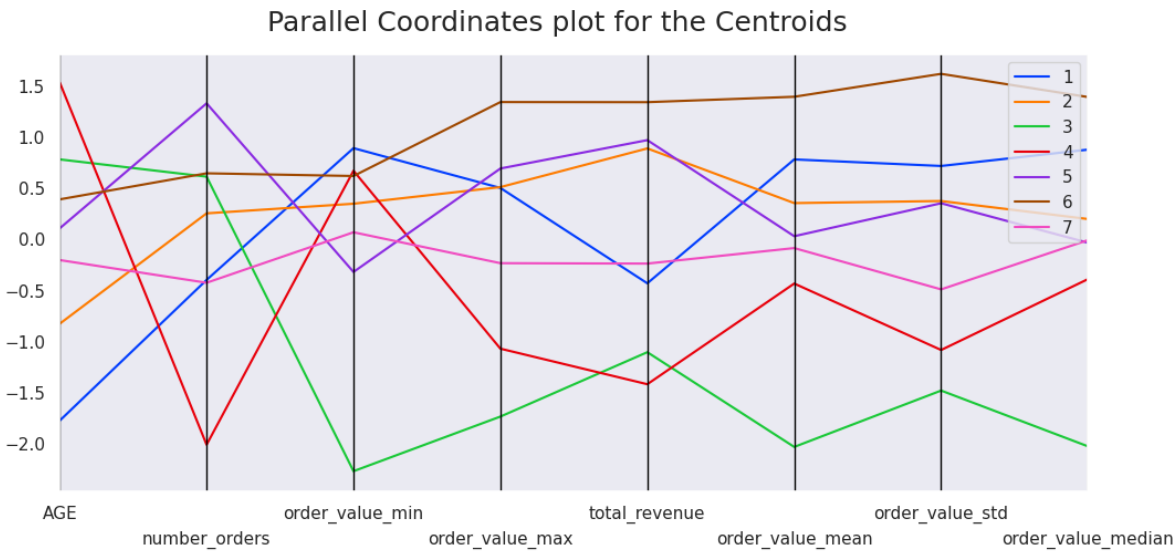
In [ ]:
```
customers_num = pd.get_dummies(customers_df, columns=['GENDER','GEOGRAPHY'])
customers_num = customers_num.select_dtypes(include=np.number)
```

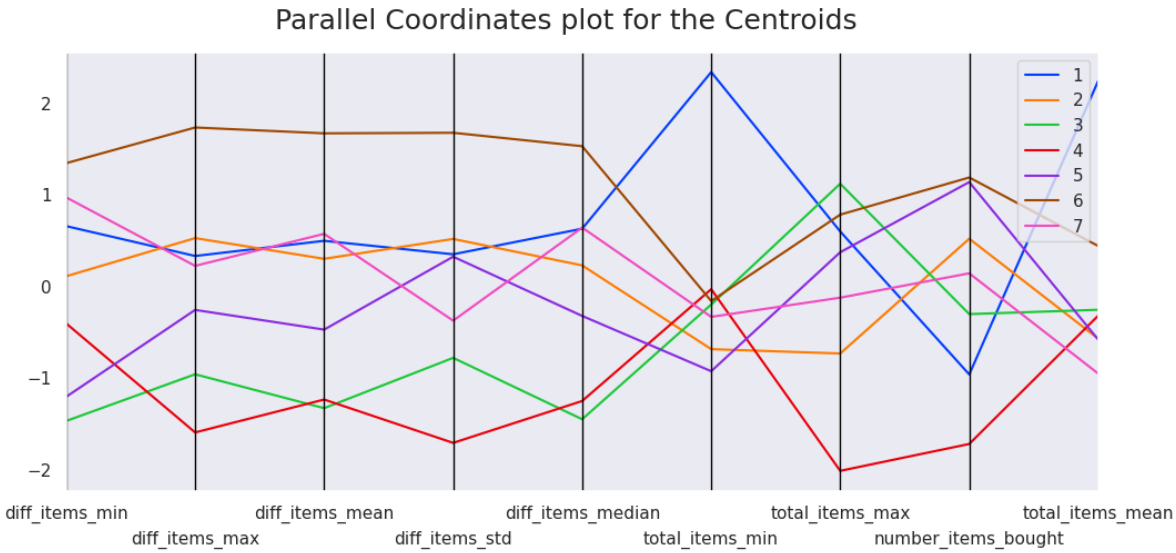
In [ ]:
```
def display_parallel_coordinates_plot(X_tr: pd.DataFrame, cluster_col: str):

    #Removing unclustered datapoints
    df_mean = X_tr.drop(X_tr[X_tr[cluster_col] == -1].index)

    #Aggregating by cluster and calculating the means of each fields
    df_mean = df_mean.groupby(cluster_col).mean(numeric_only=True)
```

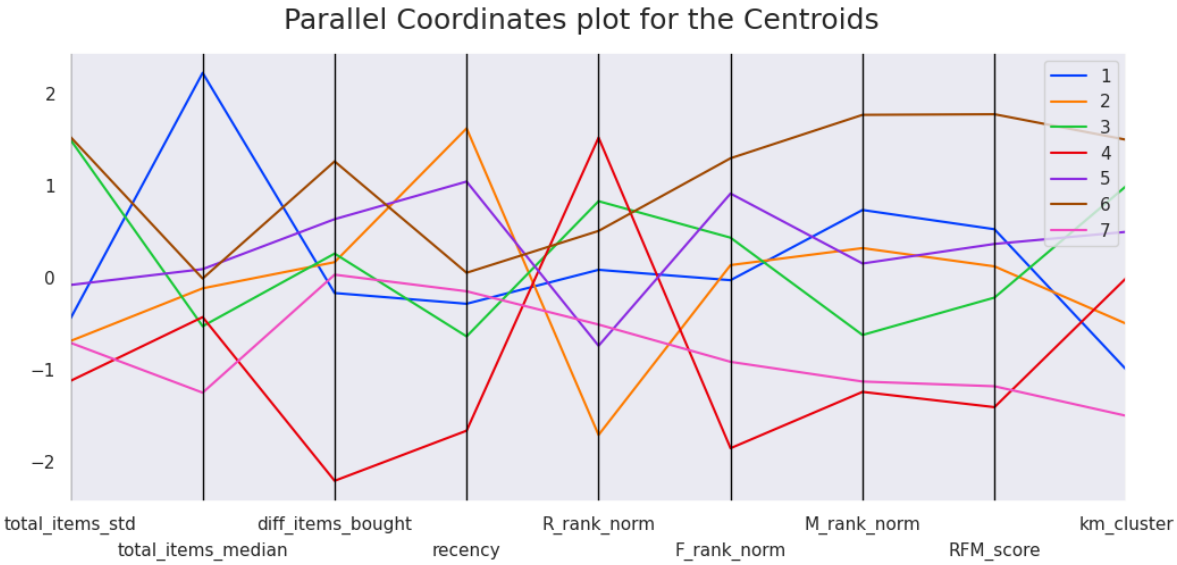
```
    #Rescaling data for better visualization
    df_mean_scaled = StandardScaler().fit_transform(df_mean)

    #Transforming our data back into a dataframe
    df_mean = pd.DataFrame(df_mean_scaled, columns=df_mean.columns.to_list()).rese
    if 'cluster' in list(df_mean.columns):
        df_mean.drop(columns='cluster', inplace=True)
    df_mean.rename(columns={"index": "cluster"}, inplace=True)

    #Adding 1 to the number of cluster just to have numbers from 1 to 7
    if min(df_mean["cluster"]) == 0:
        df_mean["cluster"] = df_mean["cluster"] + 1
    clusters = df_mean["cluster"]
    for k in range(1,len(X_tr.columns)//9+1):
        i = k*9
        if i > len(X_tr):
            subset = df_mean.iloc[:, -9:]
            subset["cluster"] = clusters
        elif k==1:
            subset = df_mean.iloc[:,(i-9):i]
        else:
            subset = df_mean.iloc[:,(i-9):i]
            subset["cluster"] = clusters
        display_parallel_coordinates_centroids(subset, 6)
        plt.show()
```

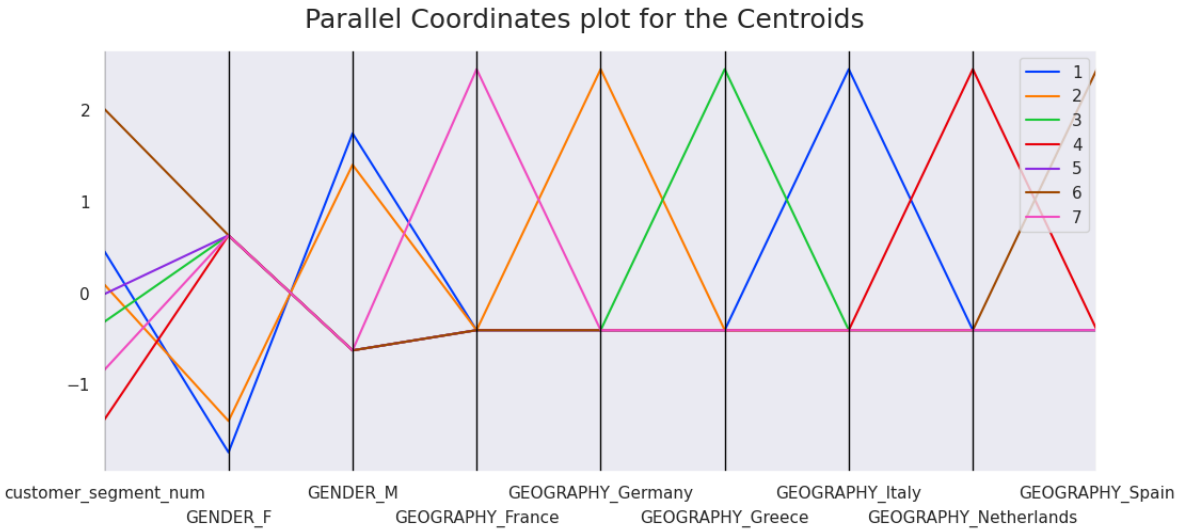
5.1 Customers Segments as described by DBSCAN clustering

In [ ]:
```
display_parallel_coordinates_plot(customers_num, 'cluster')
```









In [ ]:
```
customers_df[customers_df.cluster==1].GENDER.value_counts()
```

Out[ ]:
```
M    6806
F    1151
Name: GENDER, dtype: int64
```

7 customer segments have been identified :

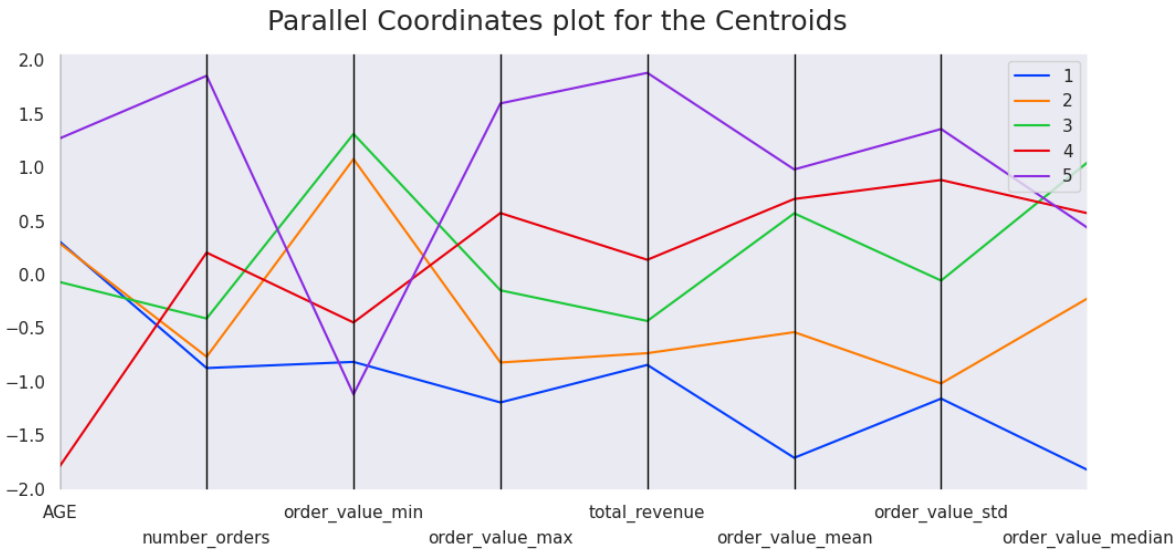
- Segment 1 : Male customers, young adults (18-26), very high number of items bought but average order value, number of different items bought and number of orders, living

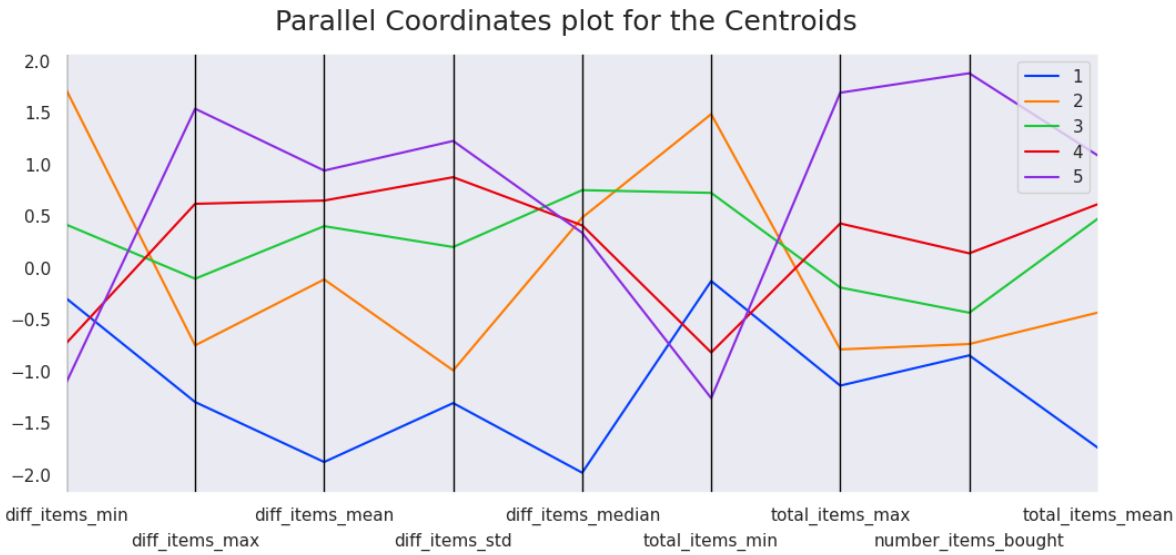
in Italy.

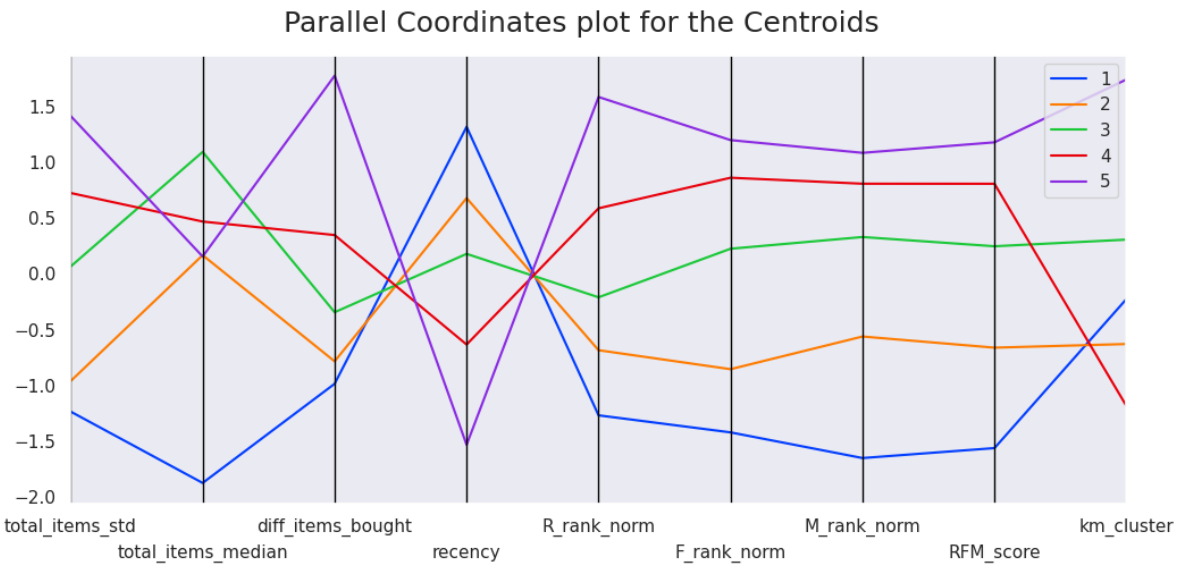
- Segment 2 : Mostly male customers, Middle aged (27-39), High order value but low recency, living in Germany.
- Segment 3 : Female customers, in their fifties (50-59), low value orders, number of items bought and low frequency but high recency and number of orders, living in Greece.
- Segment 4 : Female customers, age higher than 50, low revenue ,number of orders and number of items bought, but high recency, living in the Netherlands.
- Segment 5 : Female customers, late forties (43-47), very high number of orders, high revenue but low recency and number of items bought, living in the UK.
- Segment 6 : Female customers, Late forties (47-50), High revenue, number of orders and number of different items bought, living in Spain.
- Segment 7 : Female customers, in their forties (39-43), average revenue and order metrics, living in France.

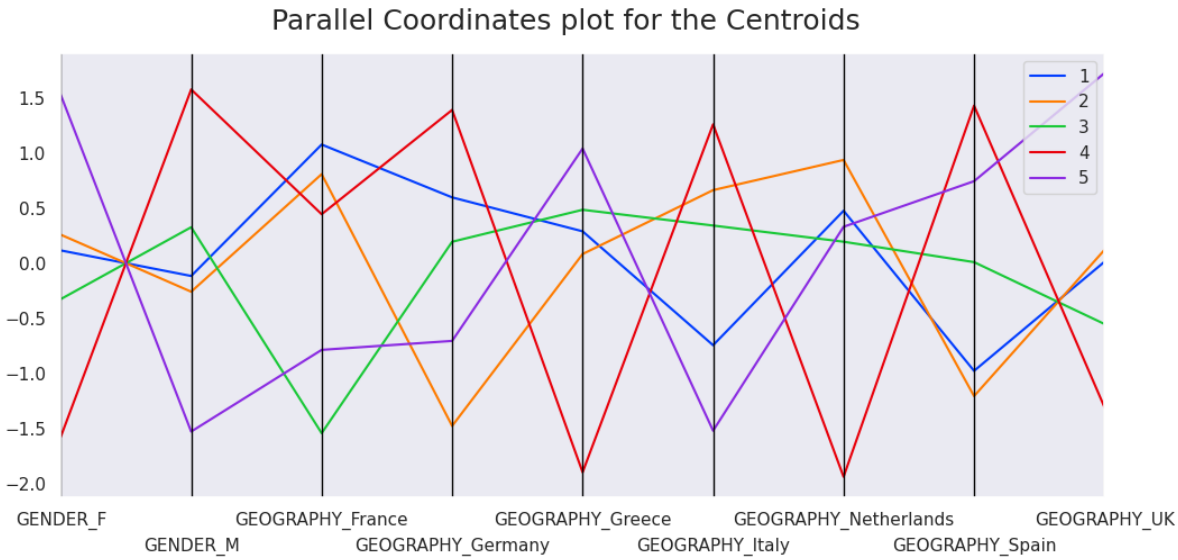
5.2 Customer segments as described by RFM clustering

In [ ]:
```
display_parallel_coordinates_plot(customers_num, 'customer_segment_num')
```









5 segments have been identified :

- Top customers (Segment 5): Likely to be Female, elderly, likely to live in Greece, the Netherlands, Spain and the UK. High revenue, number of items bought and recency.
- High value customers (Segment 4) : Likely to be Male, to live in Spain, Italy, Germany or France, younger customers.
- Medium value customers (Segment 3) : Average customers, no specific gender or age. Unlikely to live in France and likely to live in Germany.
- Low value customers (Segment 2) : Low revenue, number of items bought and recency. Unlikely to live in Spain and Germany.
- Lost customers (Segment 1) : Very low revenue, number of items bought and recency. Likely to live in France and Germany.

2 different marketing campaigns can be set in place to target highest revenue customers :

- 1st marketing strategy aimed at elderly women, mostly aimed at the Greek, Dutch, Spanish and UK markets.
- 2nd marketing strategy aimed at young men, mostly aimed at the Spanish, Italian, German and French markets.

In [ ]:
```
customers_df[customers_df.customer_segment_num == 4].GEOGRAPHY.value_counts()
```

Out[ ]:
```
Germany        694
Italy          399
France         226
UK             181
Greece         172
Spain          155
Netherlands     97
Name: GEOGRAPHY, dtype: int64
```

In [ ]:
```
customers_df.to_csv('data/customers_df_final.csv')
transactions.to_csv('data/transactions_final.csv')
```

6 CLV calculation

Churn rate

In [ ]:
```
num_customers = len(orders.Customer_ID.unique())
churn_df = orders.groupby("Customer_ID").nunique()[["Transaction_ID"]]
churn_rate = 1- len(churn_df[churn_df.Transaction_ID > 1])/num_customers
churn_rate
```

Out[ ]:
```
0.5101390241742673
```

Average Purchase Frequency

In [ ]:
```
apf = len(orders.Transaction_ID.unique())/num_customers
apf
```

Out[ ]:
```
2.856017001682458
```

Average Purchase Value

In [ ]:
```
apv = orders.Sales_Amount.sum()/num_customers
apv
```

Out[ ]:
```
69.59138094394757
```

Customer Lifetime Value (CLV)

In [ ]:
```
clv = apv*apf/churn_rate
clv
```

Out[ ]:
```
389.60784752389185
```