jupyter Starbucks Last Checkpoint: 10 hours ago

File Edit View Run Kernel Settings Help Trusted

Code JupyterLab Python 3 (ipykernel)

[95]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import plotly.graph_objects as go
```

[4]:
```
df = pd.read_csv('Orders_Table.csv') #import orders table
```

[5]:
```
df #return all the rows and column
```

[5]:

	transaction_id	transaction_date	transaction_time	store_id	store_location	product_id	transaction_qty
0	1	01-01-2023	07:06:11	5	Connaught Circle	32	2
1	2	01-01-2023	07:08:56	5	Connaught Circle	57	2
2	3	01-01-2023	07:14:04	5	Connaught Circle	59	2
3	4	01-01-2023	07:20:24	5	Connaught Circle	22	1
4	5	01-01-2023	07:22:41	5	Connaught Circle	57	2
...	...	...	...	...	...	...	...
149111	149452	30-06-2023	20:18:41	8	IGI Airport	44	2
149112	149453	30-06-2023	20:25:10	8	IGI Airport	49	2
149113	149454	30-06-2023	20:31:34	8	IGI Airport	45	1
149114	149455	30-06-2023	20:57:19	8	IGI Airport	40	1
149115	149456	30-06-2023	20:57:19	8	IGI Airport	64	2

149116 rows × 7 columns

[6]:
```
df.info() #it will give information of all the columns
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 149116 entries, 0 to 149115
Data columns (total 7 columns):
 #   Column            Non-Null Count   Dtype
---  ------            --------------   -----
 0   transaction_id    149116 non-null  int64
 1   transaction_date  149116 non-null  object
 2   transaction_time  149116 non-null  object
 3   store_id          149116 non-null  int64
 4   store_location    149116 non-null  object
 5   product_id        149116 non-null  int64
 6   transaction_qty   149116 non-null  int64
dtypes: int64(4), object(3)
memory usage: 8.0+ MB
```

[7]:
```
df.describe() #it will describe the agg func
```

[7]:

	transaction_id	store_id	product_id	transaction_qty
count	149116.000000	149116.000000	149116.000000	149116.000000
mean	74737.371872	5.342063	47.918607	1.438276
std	43153.600016	2.074241	17.930020	0.542509
min	1.000000	3.000000	1.000000	1.000000
25%	37335.750000	3.000000	33.000000	1.000000
50%	74727.500000	5.000000	47.000000	1.000000
75%	112094.250000	8.000000	60.000000	2.000000
max	149456.000000	8.000000	87.000000	8.000000

[8]:
```
df.isnull().sum() #checking for null
```

[8]:
```
transaction_id      0
transaction_date    0
transaction_time    0
```

```
store_id           0
store_location     0
product_id         0
transaction_qty    0
dtype: int64
```

[9]:
```
df.dtypes #checking for datatypes
```

[9]:
```
transaction_id       int64
transaction_date    object
transaction_time    object
store_id             int64
store_location      object
product_id           int64
transaction_qty      int64
dtype: object
```

[10]:
```
df.columns #how many columns have in a table
```

[10]:
```
Index(['transaction_id', 'transaction_date', 'transaction_time', 'store_id',
       'store_location', 'product_id', 'transaction_qty'],
      dtype='object')
```

[11]:
```
df.dropna() #remove the rows that contains the null values
```

[11]:

	transaction_id	transaction_date	transaction_time	store_id	store_location	product_id	transaction_qty
0	1	01-01-2023	07:06:11	5	Connaught Circle	32	2
1	2	01-01-2023	07:08:56	5	Connaught Circle	57	2
2	3	01-01-2023	07:14:04	5	Connaught Circle	59	2
3	4	01-01-2023	07:20:24	5	Connaught Circle	22	1
4	5	01-01-2023	07:22:41	5	Connaught Circle	57	2
...	...	...	...	...	...	...	...
149111	149452	30-06-2023	20:18:41	8	IGI Airport	44	2
149112	149453	30-06-2023	20:25:10	8	IGI Airport	49	2
149113	149454	30-06-2023	20:31:34	8	IGI Airport	45	1
149114	149455	30-06-2023	20:57:19	8	IGI Airport	40	1
149115	149456	30-06-2023	20:57:19	8	IGI Airport	64	2

149116 rows × 7 columns

[12]:
```
df.duplicated() #duplicate values will only be checked on primary key column
```

[12]:
```
0         False
1         False
2         False
3         False
4         False
          ...
149111    False
149112    False
149113    False
149114    False
149115    False
Length: 149116, dtype: bool
```

[13]:
```
df["transaction_date"] = pd.to_datetime(df["transaction_date"], dayfirst=True) #change datatype of T_date
df
```

[13]:

	transaction_id	transaction_date	transaction_time	store_id	store_location	product_id	transaction_qty
0	1	2023-01-01	07:06:11	5	Connaught Circle	32	2
1	2	2023-01-01	07:08:56	5	Connaught Circle	57	2
2	3	2023-01-01	07:14:04	5	Connaught Circle	59	2
3	4	2023-01-01	07:20:24	5	Connaught Circle	22	1
4	5	2023-01-01	07:22:41	5	Connaught Circle	57	2
...	...	...	...	...	...	...	...
149111	149452	2023-06-30	20:18:41	8	IGI Airport	44	2
149112	149453	2023-06-30	20:25:10	8	IGI Airport	49	2
149113	149454	2023-06-30	20:31:34	8	IGI Airport	45	1
149114	149455	2023-06-30	20:57:19	8	IGI Airport	40	1

149115	149456	2023-06-30	20:57:19	8	IGI Airport	64	2

149116 rows × 7 columns

[14]:
```
df.dtypes
```

[14]:
```
transaction_id              int64
transaction_date    datetime64[ns]
transaction_time            object
store_id                     int64
store_location              object
product_id                   int64
transaction_qty              int64
dtype: object
```

[15]:
```
df["transaction_time"] = pd.to_datetime(df["transaction_time"], format="%H:%M:%S")  #change datatype of t_time
print(df["transaction_time"].dtype)
```

```
datetime64[ns]
```

[16]:
```
df.dtypes
```

[16]:
```
transaction_id              int64
transaction_date    datetime64[ns]
transaction_time    datetime64[ns]
store_id                     int64
store_location              object
product_id                   int64
transaction_qty              int64
dtype: object
```

[17]:
```
df1 = pd.read_csv('Products_Table.csv') #importing another table
df1
```

[17]:

	Column1	product_id	product_category	product_type	product_detail	unit_price
0	0	32	Coffee	Gourmet brewed coffee	Ethiopia Rg	252.00
1	1	57	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	260.40
2	2	59	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	378.00
3	3	22	Coffee	Drip coffee	Our Old Time Diner Blend Sm	168.00
4	5	77	Bakery	Scone	Oatmeal Scone	252.00
...	...	...	...	...	...	...
93	11187	72	Bakery	Scone	Ginger Scone	222.60
94	11668	82	Branded	Housewares	I Need My Bean! Diner mug	1932.00
95	16516	9	Coffee beans	Organic Beans	Organic Decaf Blend	1932.00
96	42970	81	Branded	Clothing	I Need My Bean! T-shirt	1932.00
97	88852	73	Bakery	Pastry	Almond Croissant	393.96

98 rows × 6 columns

[18]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 98 entries, 0 to 97
Data columns (total 6 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   Column1           98 non-null     int64
 1   product_id        98 non-null     int64
 2   product_category  98 non-null     object
 3   product_type      98 non-null     object
 4   product_detail    98 non-null     object
 5   unit_price        98 non-null     float64
dtypes: float64(1), int64(2), object(3)
memory usage: 4.7+ KB
```

[19]:
```
df1.dtypes
```

[19]:
```
Column1              int64
product_id           int64
product_category    object
product_type        object
product_detail      object
unit_price         float64
dtype: object
```

[20]:
```
df1.drop('Column1', axis = 1, inplace = True) #deleting unnecessary column
df1
```

[20]:

	product_id	product_category	product_type	product_detail	unit_price
0	32	Coffee	Gourmet brewed coffee	Ethiopia Rg	252.00
1	57	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg	260.40
2	59	Drinking Chocolate	Hot chocolate	Dark chocolate Lg	378.00
3	22	Coffee	Drip coffee	Our Old Time Diner Blend Sm	168.00
4	77	Bakery	Scone	Oatmeal Scone	252.00
...	...	...	...	...	...
93	72	Bakery	Scone	Ginger Scone	222.60
94	82	Branded	Housewares	I Need My Bean! Diner mug	1932.00
95	9	Coffee beans	Organic Beans	Organic Decaf Blend	1932.00
96	81	Branded	Clothing	I Need My Bean! T-shirt	1932.00
97	73	Bakery	Pastry	Almond Croissant	393.96

98 rows × 5 columns

[21]:
```
df1.describe()
```

[21]:

	product_id	unit_price
count	98.000000	98.000000
mean	45.744898	608.185714
std	26.375290	645.864397
min	1.000000	67.200000
25%	22.250000	252.000000
50%	46.500000	315.000000
75%	71.750000	751.800000
max	87.000000	3780.000000

[22]:
```
df1.isnull().sum()
```

[22]:
```
product_id          0
product_category    0
product_type        0
product_detail      0
unit_price          0
dtype: int64
```

[23]:
```
df1.duplicated()
```

[23]:
```
0     False
1     False
2     False
3     False
4     False
      ...
93    False
94    False
95    False
96    False
97    False
Length: 98, dtype: bool
```

On combining both Tables

[24]:
```
df2 = pd.merge(df,df1,on = 'product_id') #merging both tables i.e., products and orders
df2
```

[24]:

	transaction_id	transaction_date	transaction_time	store_id	store_location	product_id	transaction_qty	product_category	product_type	product_detail
0	1	2023-01-01	1900-01-01 07:06:11	5	Connaught Circle	32	2	Coffee	Gourmet brewed coffee	Ethiopia Rg
1	2	2023-01-01	1900-01-01 07:08:56	5	Connaught Circle	57	2	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg
2	3	2023-01-01	1900-01-01	5	Connaught	59	2	Drinking	Hot chocolate	Dark chocolate

			07:14:04		Circle			Chocolate		Lg
3	4	2023-01-01	1900-01-01 07:20:24	5	Connaught Circle	22	1	Coffee	Drip coffee	Our Old Time Diner Blend Sm
4	5	2023-01-01	1900-01-01 07:22:41	5	Connaught Circle	57	2	Tea	Brewed Chai tea	Spicy Eye Opener Chai Lg
...	...	...	...	...	...	...	...	...	...	...
176006	149452	2023-06-30	1900-01-01 20:18:41	8	IGI Airport	44	2	Tea	Brewed herbal tea	Peppermint Rg
176007	149453	2023-06-30	1900-01-01 20:25:10	8	IGI Airport	49	2	Tea	Brewed Black tea	English Br[illegible] Lg
176008	149454	2023-06-30	1900-01-01 20:31:34	8	IGI Airport	45	1	Tea	Brewed herbal tea	Peppermint Lg
176009	149455	2023-06-30	1900-01-01 20:57:19	8	IGI Airport	40	1	Coffee	Barista Espresso	Cappuccino
176010	149456	2023-06-30	1900-01-01 20:57:19	8	IGI Airport	64	2	Flavours	Regular syrup	Hazelnut syrup

176011 rows × 11 columns

[25]:
```
df2.describe()
```

[25]:

	transaction_id	transaction_date	transaction_time	store_id	product_id	transaction_qty	unit_price
count	176011.000000	176011	176011	176011.000000	176011.000000	176011.000000	176011.000000
mean	74766.386822	2023-04-15 12:37:44.667549184	1900-01-01 12:11:27.771844096	5.367210	51.787780	1.380936	305.591576
min	1.000000	2023-01-01 00:00:00	1900-01-01 06:00:00	3.000000	1.000000	1.000000	67.200000
25%	37361.500000	2023-03-06 00:00:00	1900-01-01 09:03:56	3.000000	36.000000	1.000000	210.000000
50%	74658.000000	2023-04-24 00:00:00	1900-01-01 11:10:44	5.000000	51.000000	1.000000	252.000000
75%	112043.500000	2023-05-30 00:00:00	1900-01-01 15:21:01	8.000000	71.000000	2.000000	315.000000
max	149456.000000	2023-06-30 00:00:00	1900-01-01 20:59:32	8.000000	87.000000	8.000000	3780.000000
std	43164.402702	NaN	NaN	2.068517	19.333663	0.527875	251.488216

[26]:
```
df2.isnull().sum()
```

[26]:
```
transaction_id      0
transaction_date    0
transaction_time    0
store_id            0
store_location      0
product_id          0
transaction_qty     0
product_category    0
product_type        0
product_detail      0
unit_price          0
dtype: int64
```

[27]:
```
df2.dtypes
```

[27]:
```
transaction_id              int64
transaction_date    datetime64[ns]
transaction_time    datetime64[ns]
store_id                    int64
store_location             object
product_id                  int64
transaction_qty             int64
product_category           object
product_type               object
product_detail             object
unit_price                float64
dtype: object
```

Now, start performing analysis on this table

Firstly import the data onto Mysql

[29]:
```
import mysql.connector
import os

# List of CSV files and their corresponding table names
csv_files = [
    ('Orders_Table.csv', 'Orders_Table'),
    ('Products_Table.csv', 'Products_Table') # Added payments.csv for specific handling
]

# Connect to the MySQL database
conn = mysql.connector.connect(
    host='localhost',
    user='root',
    password='0000',
    database='starbucks'
)
cursor = conn.cursor()

# Folder containing the CSV files
folder_path = 'C:/Users/parampara/Desktop/Shanu_DA/Starbucks'

def get_sql_type(dtype):
    if pd.api.types.is_integer_dtype(dtype):
        return 'INT'
    elif pd.api.types.is_float_dtype(dtype):
        return 'FLOAT'
    elif pd.api.types.is_bool_dtype(dtype):
        return 'BOOLEAN'
    elif pd.api.types.is_datetime64_any_dtype(dtype):
        return 'DATETIME'
    else:
        return 'TEXT'

for csv_file, table_name in csv_files:
    file_path = os.path.join(folder_path, csv_file)

    # Read the CSV file into a pandas DataFrame
    df = pd.read_csv(file_path)

    # Replace NaN with None to handle SQL NULL
    df = df.where(pd.notnull(df), None)

    # Debugging: Check for NaN values
    print(f"Processing {csv_file}")
    print(f"NaN values before replacement:\n{df.isnull().sum()}\n")

    # Clean column names
    df.columns = [col.replace(' ', '_').replace('-', '_').replace('.', '_') for col in df.columns]

    # Generate the CREATE TABLE statement with appropriate data types
    columns = ', '.join([f'`{col}` {get_sql_type(df[col].dtype)}' for col in df.columns])
    create_table_query = f'CREATE TABLE IF NOT EXISTS `{table_name}` ({columns})'
    cursor.execute(create_table_query)

    # Insert DataFrame data into the MySQL table
    for _, row in df.iterrows():
        # Convert row to tuple and handle NaN/None explicitly
        values = tuple(None if pd.isna(x) else x for x in row)
        sql = f"INSERT INTO `{table_name}` ({', '.join(['`' + col + '`' for col in df.columns])}) VALUES ({', '.join(['%s'] * len(row))})"
        cursor.execute(sql, values)

    # Commit the transaction for the current CSV file
    conn.commit()

# Close the connection
conn.close()
```

```
Processing Orders_Table.csv
NaN values before replacement:
transaction_id      0
transaction_date    0
transaction_time    0
store_id            0
store_location      0
product_id          0
transaction_qty     0
dtype: int64

Processing Products_Table.csv
NaN values before replacement:
Column1             0
product_id          0
product_category    0
product_type        0
```

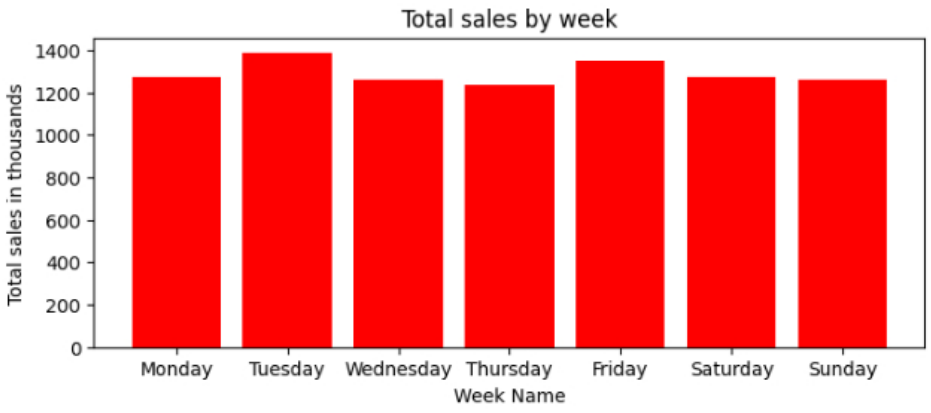
```
product_detail      0
unit_price          0
dtype: int64
```

Connection establish to mysql

[160]:
```
import mysql.connector
db = mysql.connector.connect(host = 'localhost',
                             username = 'root',
                             password = '0000',
                             database = 'starbucks')
cur = db.cursor()
```

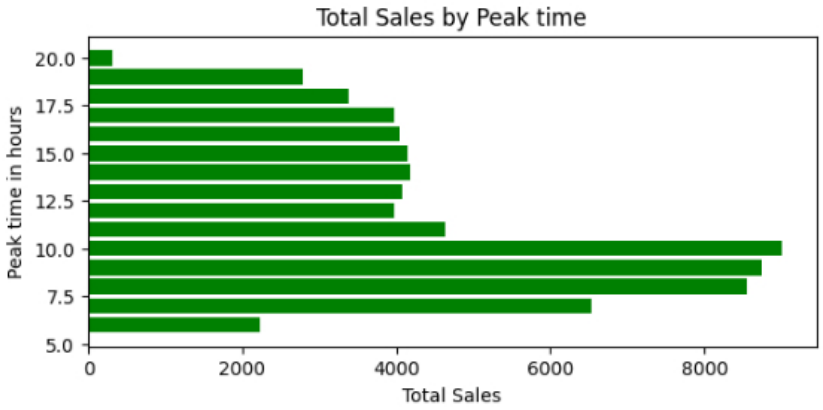
How do sales vary by day of the week and hour of day

[131]:
```
query = """ SELECT
    DAYNAME(transaction_date) AS day_of_week,
    HOUR(orders_table.transaction_time) AS hour_of_day,
    round(SUM(products_table.unit_price * orders_table.transaction_qty),2) AS total_sales
FROM orders_table join products_table
on orders_table.product_id = products_table.product_id
GROUP BY day_of_week, hour_of_day
ORDER BY FIELD(day_of_week, 'Monday','Tuesday','Wednesday','Thursday','Friday','Saturday','Sunday'),
        hour_of_day;
 """

cur.execute(query)
data = cur.fetchall()
data = pd.DataFrame(data, columns=['Day_Name', 'Hour', 'Total_sales'])

plt.figure(figsize = (8,3))
plt.bar(data['Day_Name'], data['Total_sales']/1000, color = 'red')

plt.xlabel("Week Name")
plt.ylabel("Total sales in thousands")
plt.title("Total sales by week")
plt.show()
```



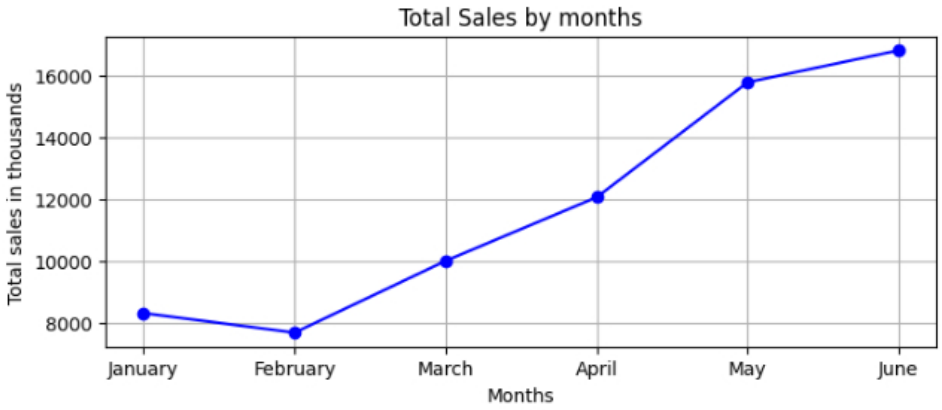
Are there any peak time for sales activity

[132]:
```
query = """ select hour(orders_table.transaction_time) as peak_time,
round(sum(products_table.unit_price * orders_table.transaction_qty),2) as total_sales
from orders_table join products_table
on orders_table.product_id  = products_table.product_id
group by peak_time
order by total_sales desc; """

cur.execute(query)
data = cur.fetchall()
data = pd.DataFrame(data, columns = ['Peak_time', 'Total_sales'])
plt.figure(figsize = (7,3))
plt.barh(data['Peak_time'], data['Total_sales']/1000, color = 'green')
plt.xlabel('Total Sales')
plt.ylabel('Peak time in hours')
plt.title('Total Sales by Peak time')

plt.show()
```



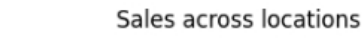
What is the total sales revenue for each month

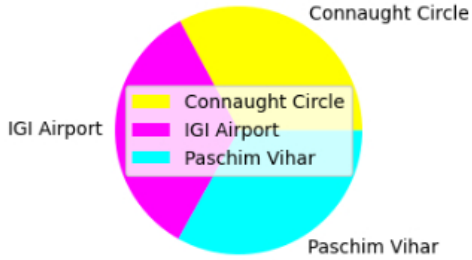
[133]:
```
query = """ select monthname(orders_table.transaction_date) as month_name,
round(sum(products_table.unit_price * orders_table.transaction_qty),2) as total_sales
from orders_table join products_table
on orders_table.product_id = products_table.product_id
group by month(orders_table.transaction_date), month_name
order by month(orders_table.transaction_date); """

cur.execute(query)
data = cur.fetchall()
data = pd.DataFrame(data, columns = ['Months', 'Sales'])
plt.figure(figsize = (8,3))
plt.plot(data['Months'], data['Sales']/1000, color = 'blue', marker = 'o')
plt.xlabel('Months')
plt.ylabel('Total sales in thousands')
plt.title('Total Sales by months')
plt.grid(True)
plt.show()
```



How to sales vary across different store locations

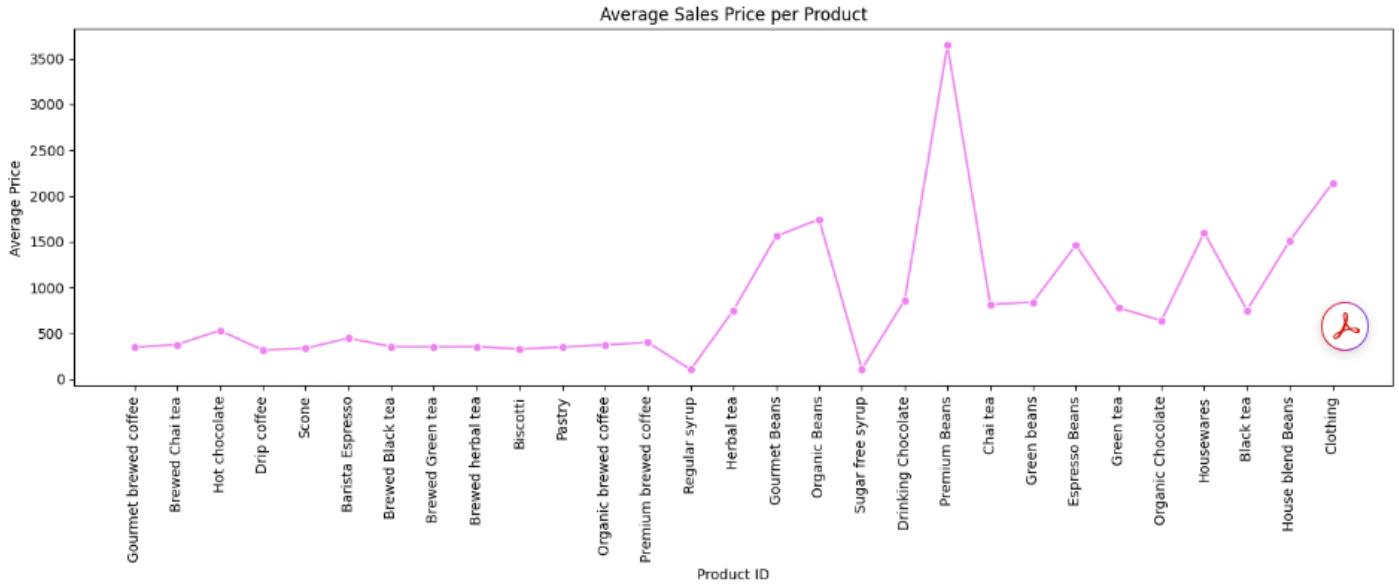
[134]:
```
query = """ select orders_table.store_location as location,
round(sum(products_table.unit_price * orders_table.transaction_qty),2) as total_sales
from orders_table join products_table
on orders_table.product_id = products_table.product_id
group by location; """

cur.execute(query)
data = cur.fetchall()
plt.figure(figsize = (3,3))

x = [17672741.54, 18312750.98, 17801986.46]
y = ['Connaught Circle', 'IGI Airport', 'Paschim Vihar']
c = ['yellow', 'magenta', 'aqua']
plt.pie(x , labels = y, colors = c)
plt.legend()
plt.title('Sales across locations')
plt.show()
```





What is the average price/order per person

[184]:
```
query = """ select products_table.product_id,
round(avg(products_table.unit_price * orders_table.transaction_qty),2) as average_price
from products_table join orders_table
group by products_table.product_id; """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(rows, columns=['product_id', 'product_type', 'average_price'])

plt.figure(figsize=(14,6))
sns.lineplot(data=df, x='product_id', y='average_price', marker='o', color = 'violet')
plt.title('Average Sales Price per Product')
plt.xlabel('Product ID')
plt.ylabel('Average Price')
plt.xticks(rotation=90)
plt.tight_layout()
plt.show()
```



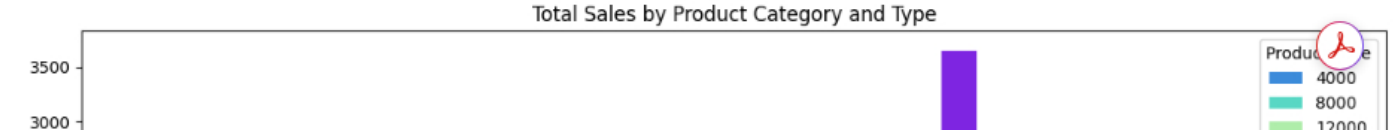
Which products are best selling in terms of quantity and revenue

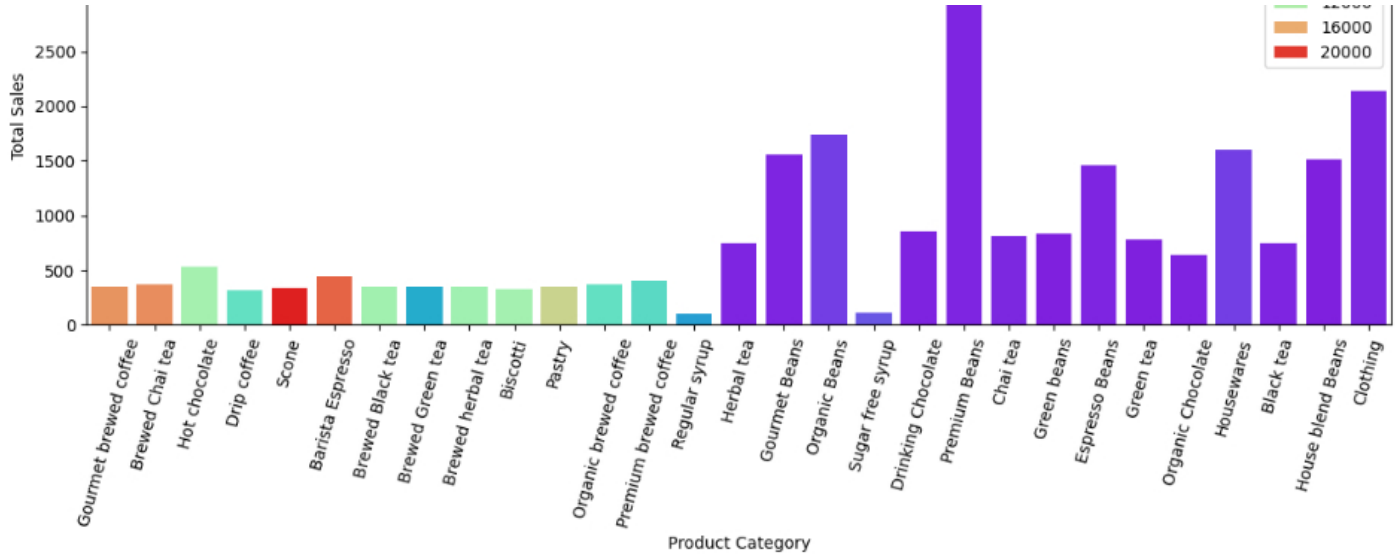
[164]:
```
query = """ select products_table.product_type, count(orders_table.transaction_qty) as total_count,
round(avg(products_table.unit_price * orders_table.transaction_qty),2) as revenue
from products_table join orders_table
on products_table.product_id = orders_table.product_id
group by products_table.product_type;
 """

cur.execute(query)
rows = cur.fetchall()
data = pd.DataFrame(rows, columns=['Product', 'Count', 'Revenue'])
x = np.arange(len(data['Product']))  # product positions
width = 0.4  # bar width

plt.figure(figsize=(13,6))

# Plot Count
plt.bar(x - width/2, data['Count'], width, label='Count', color='orange')
```

```
# Plot Revenue
plt.bar(x + width/2, data['Revenue'], width, label='Revenue', color='maroon')

# Labels & formatting

plt.xlabel("Products")
plt.ylabel("Values")
plt.title("Product-wise Count & Revenue")
plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.6)
plt.xticks(rotation = 90)

plt.show()
```

How do sales vary by product, category and type

[172]:
```
query = """ select products_table.product_category, products_table.product_type,
round(sum(products_table.unit_price * orders_table.transaction_qty),2) as total_sales
from products_table join orders_table
on products_table.product_id = orders_table.product_id
group by products_table.product_category, products_table.product_type
order by total_sales desc; """

cur.execute(query)
data = cur.fetchall()
df = pd.DataFrame(rows, columns=['product_category', 'product_type', 'total_sales'])

plt.figure(figsize=(12,6))
sns.barplot(
    data=df,
    x='product_category',
    y='total_sales',
    hue='product_type',  # now matches the column name
    palette='rainbow'
)
plt.title('Total Sales by Product Category and Type')
plt.xlabel('Product Category')
plt.ylabel('Total Sales')
plt.xticks(rotation=75)
plt.legend(title='Product Type')
plt.tight_layout()
plt.show()
```




16000
20000
Total Sales
0
500
1000
1500
2000
2500
Gourmet brewed coffee
Brewed Chai tea
Hot chocolate
Drip coffee
Scone
Barista Espresso
Brewed Black tea
Brewed Green tea
Brewed herbal tea
Biscotti
Pastry
Organic brewed coffee
Premium brewed coffee
Regular syrup
Herbal tea
Gourmet Beans
Organic Beans
Sugar free syrup
Drinking Chocolate
Premium Beans
Chai tea
Green beans
Espresso Beans
Green tea
Organic Chocolate
Housewares
Black tea
House blend Beans
Clothing
Product Category

[ ]: