diwali-sales-analysis-project

November 1, 2024

[1]:
```
# import python libraries

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt # visualizing data
%matplotlib inline
import seaborn as sns
```

[2]:
```
# import csv file
df = pd.read_csv('Diwali Sales Data.csv', encoding= 'unicode_escape')
```

[3]:
```
#to know how many rows and columns are there
df.shape
```

[3]:
```
(11251, 15)
```

[4]:
```
#to see top 10 rows of the dataset
df.head(10)
```

[4]:

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Marital_Status	State	Zone	Occupation	Product_Category	Orders
0	1002903	Sanskriti	P00125942	F	26-35	28	0	Maharashtra	Western	Healthcare	Auto	1
1	1000732	Kartik	P00110942	F	26-35	35	1	Andhra Pradesh	Southern	Govt	Auto	3
2	1001990	Bindu	P00118542	F	26-35	35	1	Uttar Pradesh	Central	Automobile	Auto	3
3	1001425	Sudevi	P00237842	M	0-17	16	0	Karnataka	Southern	Construction	Auto	2
4	1000588	Joni	P00057942	M	26-35	28	1	Gujarat	Western	Food Processing	Auto	2
5	1000588	Joni	P00057942	M	26-35	28	1					
6	1001132	Balk	P00018042	F	18-25	25	1					
7	1002092	Shivangi	P00273442	F	55+	61	0					
8	1003224	Kushal	P00205642	M	26-35	35	0					
9	1003650	Ginny	P00031142	F	26-35	26	1					

```
5  Himachal Pradesh  Northern  Food Processing            Auto       1
6     Uttar Pradesh   Central           Lawyer            Auto       4
7       Maharashtra   Western       IT Sector             Auto       1
8     Uttar Pradesh   Central             Govt            Auto       2
9    Andhra Pradesh  Southern            Media            Auto       4

     Amount  Status  unnamed1
0  23952.00     NaN       NaN
1  23934.00     NaN       NaN
2  23924.00     NaN       NaN
3  23912.00     NaN       NaN
4  23877.00     NaN       NaN
5  23877.00     NaN       NaN
6  23841.00     NaN       NaN
7       NaN     NaN       NaN
8  23809.00     NaN       NaN
9  23799.99     NaN       NaN
```

[5]:
```
#now we are going to do Data Cleaning
#we want to know how many columns are there, what is the datatype(Dtype)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 15 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User_ID           11251 non-null  int64
 1   Cust_name         11251 non-null  object
 2   Product_ID        11251 non-null  object
 3   Gender            11251 non-null  object
 4   Age Group         11251 non-null  object
 5   Age               11251 non-null  int64
 6   Marital_Status    11251 non-null  int64
 7   State             11251 non-null  object
 8   Zone              11251 non-null  object
 9   Occupation        11251 non-null  object
 10  Product_Category  11251 non-null  object
 11  Orders            11251 non-null  int64
 12  Amount            11239 non-null  float64
 13  Status            0 non-null      float64
 14  unnamed1          0 non-null      float64
dtypes: float64(3), int64(4), object(8)
memory usage: 1.3+ MB
```

[6]:
```
#we can see Status and Unnamed1 these 2 columns has null values
#so we will remove this columns
```

```
#drop unrelated/blank columns
df.drop(['Status', 'unnamed1'], axis=1, inplace=True)
```

[7]:
```
#checking again if those 2 columns are deleted
#In pandas, the inplace=True parameter is used with methods that modify a␣
 ↪DataFrame directly,
#rather than returning a new DataFrame.
#This can save memory when working with large datasets because it avoids␣
 ↪creating a copy.

df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 11251 entries, 0 to 11250
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User_ID           11251 non-null  int64
 1   Cust_name         11251 non-null  object
 2   Product_ID        11251 non-null  object
 3   Gender            11251 non-null  object
 4   Age Group         11251 non-null  object
 5   Age               11251 non-null  int64
 6   Marital_Status    11251 non-null  int64
 7   State             11251 non-null  object
 8   Zone              11251 non-null  object
 9   Occupation        11251 non-null  object
 10  Product_Category  11251 non-null  object
 11  Orders            11251 non-null  int64
 12  Amount            11239 non-null  float64
dtypes: float64(1), int64(4), object(8)
memory usage: 1.1+ MB
```

[11]:
```
#now we want to check if there is any null values
#is null = false means there is no null value
#is null = true means there is null values
pd.isnull(df)
```

[11]:

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	\
0	False	False	False	False	False	False	
1	False	False	False	False	False	False	
2	False	False	False	False	False	False	
3	False	False	False	False	False	False	
4	False	False	False	False	False	False	
…	…	…	…	…	…	…	
11246	False	False	False	False	False	False	

```
11247    False      False       False   False      False  False
11248    False      False       False   False      False  False
11249    False      False       False   False      False  False
11250    False      False       False   False      False  False

       Marital_Status  State   Zone  Occupation  Product_Category  Orders  \
0               False  False  False       False             False   False
1               False  False  False       False             False   False
2               False  False  False       False             False   False
3               False  False  False       False             False   False
4               False  False  False       False             False   False
…                   …      …      …           …                 …       …
11246           False  False  False       False             False   False
11247           False  False  False       False             False   False
11248           False  False  False       False             False   False
11249           False  False  False       False             False   False
11250           False  False  False       False             False   False

       Amount
0       False
1       False
2       False
3       False
4       False
…           …
11246   False
11247   False
11248   False
11249   False
11250   False

[11251 rows x 13 columns]
```

[12]:
```
#but its hard to visualize
#we need too check manually
#so we are using .sum()
#now if there is null values in a column
#it will count how many null values are there


pd.isnull(df).sum()
```

[12]:
```
User_ID             0
Cust_name           0
Product_ID          0
Gender              0
Age Group           0
```

```
Age                 0
Marital_Status      0
State               0
Zone                0
Occupation          0
Product_Category    0
Orders              0
Amount             12
dtype: int64
```

[15]:
```
#Ammount has 12 null values
#we are going to delete the null values
#but before deleting we are going to check the row and column number

df.shape
```

[15]:
```
(11251, 13)
```

[16]:
```
#notice that number of rows is going to change
# drop null values
df.dropna(inplace=True)
```

[17]:
```
df.shape
```

[17]:
```
(11239, 13)
```

[18]:
```
#notice now there is no null values in the Amount column
pd.isnull(df).sum()
```

[18]:
```
User_ID             0
Cust_name           0
Product_ID          0
Gender              0
Age Group           0
Age                 0
Marital_Status      0
State               0
Zone                0
Occupation          0
Product_Category    0
Orders              0
Amount              0
dtype: int64
```

[19]:
```
#now i want to change the data type
#to do that let us first see the data type
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 11239 entries, 0 to 11250
Data columns (total 13 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   User_ID           11239 non-null  int64
 1   Cust_name         11239 non-null  object
 2   Product_ID        11239 non-null  object
 3   Gender            11239 non-null  object
 4   Age Group         11239 non-null  object
 5   Age               11239 non-null  int64
 6   Marital_Status    11239 non-null  int64
 7   State             11239 non-null  object
 8   Zone              11239 non-null  object
 9   Occupation        11239 non-null  object
 10  Product_Category  11239 non-null  object
 11  Orders            11239 non-null  int64
 12  Amount            11239 non-null  float64
dtypes: float64(1), int64(4), object(8)
memory usage: 1.2+ MB
```

[21]:
```
#Amount's data type is float
#We want to change it to int

# change data type
df['Amount'] = df['Amount'].astype('int')
```

[22]:
```
#checking if its changed
df['Amount'].dtypes
```

[22]:
```
dtype('int64')
```

[24]:
```
#we want to change some column names
#let us see the columns
df.columns
```

[24]:
```
Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender', 'Age Group', 'Age',
       'Marital_Status', 'State', 'Zone', 'Occupation', 'Product_Category',
       'Orders', 'Amount'],
      dtype='object')
```

[26]:
```
#rename column
df.rename(columns= {'Marital_Status':'Shaadi'})
#it is not saved since we did not use inplace = true
```

[26]:

	User_ID	Cust_name	Product_ID	Gender	Age Group	Age	Shaadi	\
0	1002903	Sanskriti	P00125942	F	26-35	28	0	

```
1       1000732       Kartik  P00110942      F     26-35   35       1
2       1001990        Bindu  P00118542      F     26-35   35       1
3       1001425       Sudevi  P00237842      M      0-17   16       0
4       1000588         Joni  P00057942      M     26-35   28       1
…         …           …       …    …       …  …     …
11246   1000695      Manning  P00296942      M     18-25   19       1
11247   1004089  Reichenbach  P00171342      M     26-35   33       0
11248   1001209        Oshin  P00201342      F     36-45   40       0
11249   1004023       Noonan  P00059442      M     36-45   37       0
11250   1002744      Brumley  P00281742      F     18-25   19       0

                State      Zone       Occupation Product_Category  Orders  \
0         Maharashtra   Western       Healthcare             Auto       1
1       Andhra Pradesh  Southern             Govt             Auto       3
2        Uttar Pradesh   Central       Automobile             Auto       3
3            Karnataka  Southern     Construction             Auto       2
4              Gujarat   Western  Food Processing             Auto       2
…                …       …               …              …     …
11246     Maharashtra   Western         Chemical           Office       4
11247         Haryana  Northern       Healthcare       Veterinary       3
11248  Madhya Pradesh   Central          Textile           Office       4
11249       Karnataka  Southern      Agriculture           Office       3
11250     Maharashtra   Western       Healthcare           Office       3

       Amount
0       23952
1       23934
2       23924
3       23912
4       23877
…         …
11246     370
11247     367
11248     213
11249     206
11250     188

[11239 rows x 13 columns]
```

[28]:
```
#notice that its not changed
df.columns
```

[28]:
```
Index(['User_ID', 'Cust_name', 'Product_ID', 'Gender', 'Age Group', 'Age',
       'Marital_Status', 'State', 'Zone', 'Occupation', 'Product_Category',
       'Orders', 'Amount'],
      dtype='object')
```

[31]:
```
# describe() method returns description of the data in the DataFrame (i.e.␣
 ↪count, mean, std, q1, q2, q3 etc)
df.describe()
```

[31]:

	User_ID	Age	Marital_Status	Orders	Amount
count	1.123900e+04	11239.000000	11239.000000	11239.000000	11239.000000
mean	1.003004e+06	35.410357	0.420055	2.489634	9453.610553
std	1.716039e+03	12.753866	0.493589	1.114967	5222.355168
min	1.000001e+06	12.000000	0.000000	1.000000	188.000000
25%	1.001492e+06	27.000000	0.000000	2.000000	5443.000000
50%	1.003064e+06	33.000000	0.000000	2.000000	8109.000000
75%	1.004426e+06	43.000000	1.000000	3.000000	12675.000000
max	1.006040e+06	92.000000	1.000000	4.000000	23952.000000

[32]:
```
# use describe() for specific columns
df[['Age', 'Orders', 'Amount']].describe()
```

[32]:

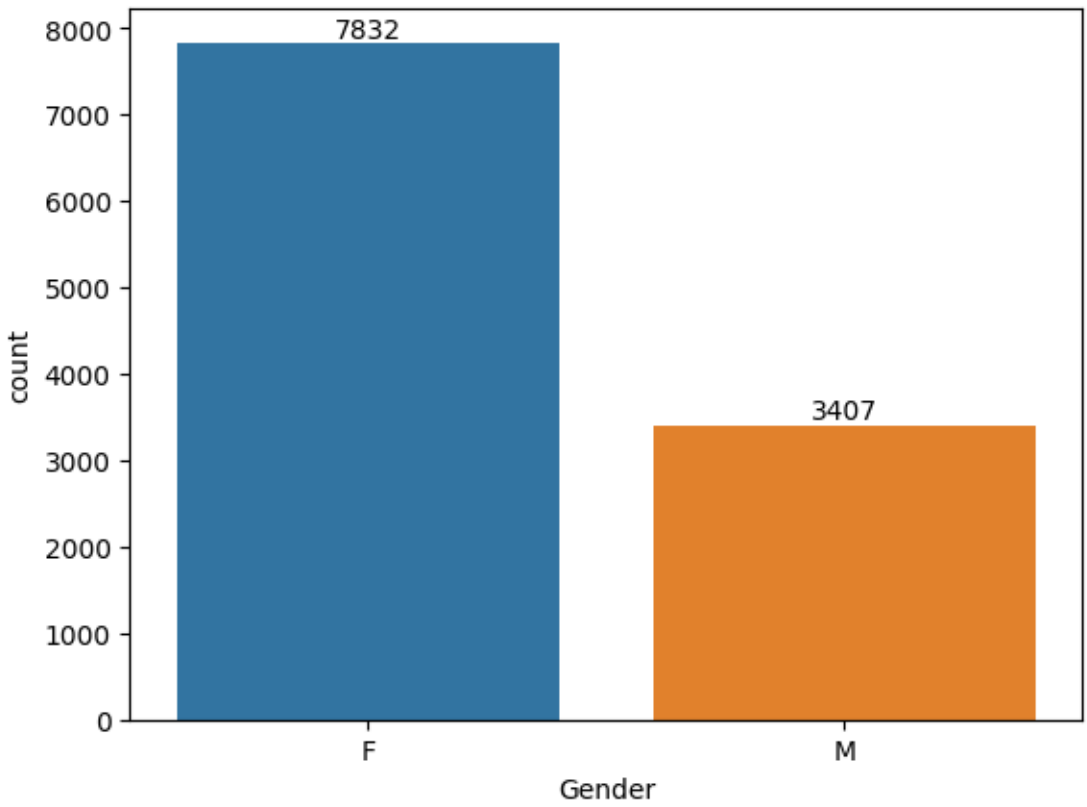
	Age	Orders	Amount
count	11239.000000	11239.000000	11239.000000
mean	35.410357	2.489634	9453.610553
std	12.753866	1.114967	5222.355168
min	12.000000	1.000000	188.000000
25%	27.000000	2.000000	5443.000000
50%	33.000000	2.000000	8109.000000
75%	43.000000	3.000000	12675.000000
max	92.000000	4.000000	23952.000000

[33]:
```
#data cleaning is done
```

1 Exploratory Data Analysis

1.0.1 Gender

[43]:
```
# plotting a bar chart for Gender and it's count

ax = sns.countplot(x = 'Gender',data = df, hue='Gender')

#to show the values also
for bars in ax.containers:
    ax.bar_label(bars)
```



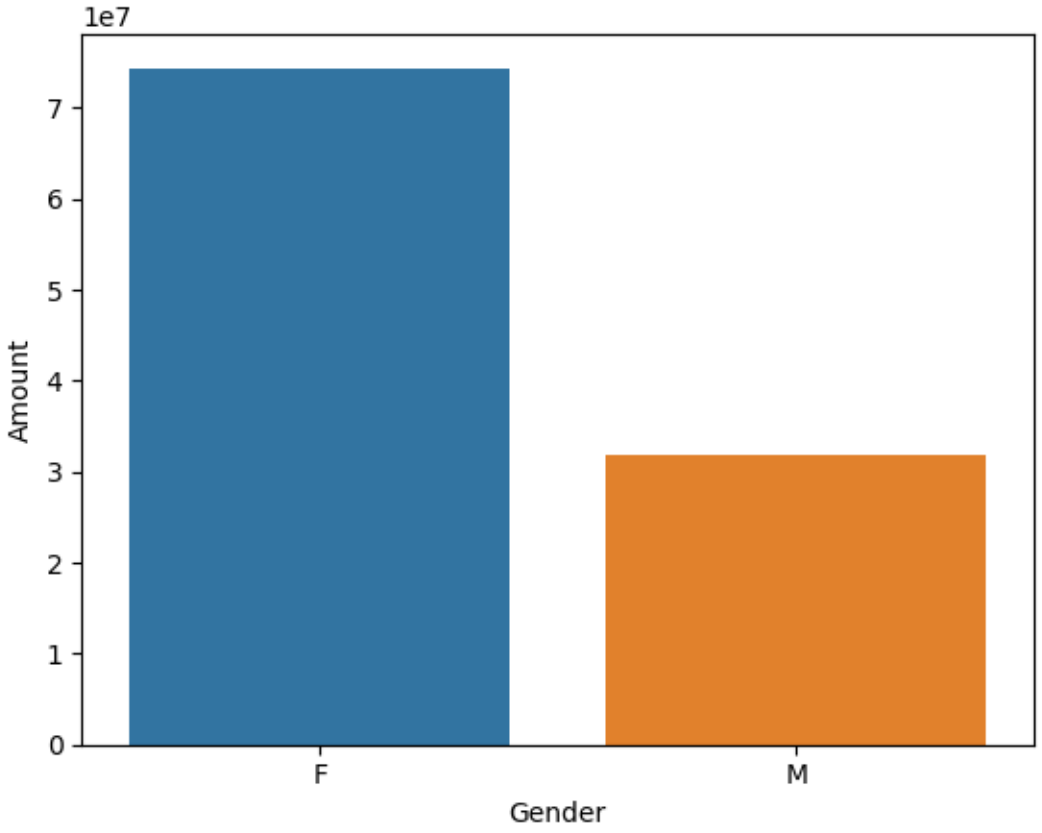
[38]:
```
sales_gen = df.groupby(['Gender'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False)
sales_gen

#This groups the data by the Gender column. Setting as_index=False means that␣
 ↪Gender will remain as a column rather than becoming the index in the result.
#After grouping, this calculates the sum of the Amount column for each gender
#Then it sorts the resulting DataFrame in descending order by the Amount␣
 ↪column, so the genders with higher total amounts appear first.
#The final result, sales_gen, will contain a DataFrame with each Gender and the␣
 ↪corresponding summed Amount, sorted from highest to lowest amount.
```

[38]:

	Gender	Amount
0	F	74335853
1	M	31913276

[46]:
```
# plotting a bar chart for gender vs total amount
#hue works as legend

sns.barplot(x = 'Gender',y= 'Amount' ,data = sales_gen, hue='Gender')
```

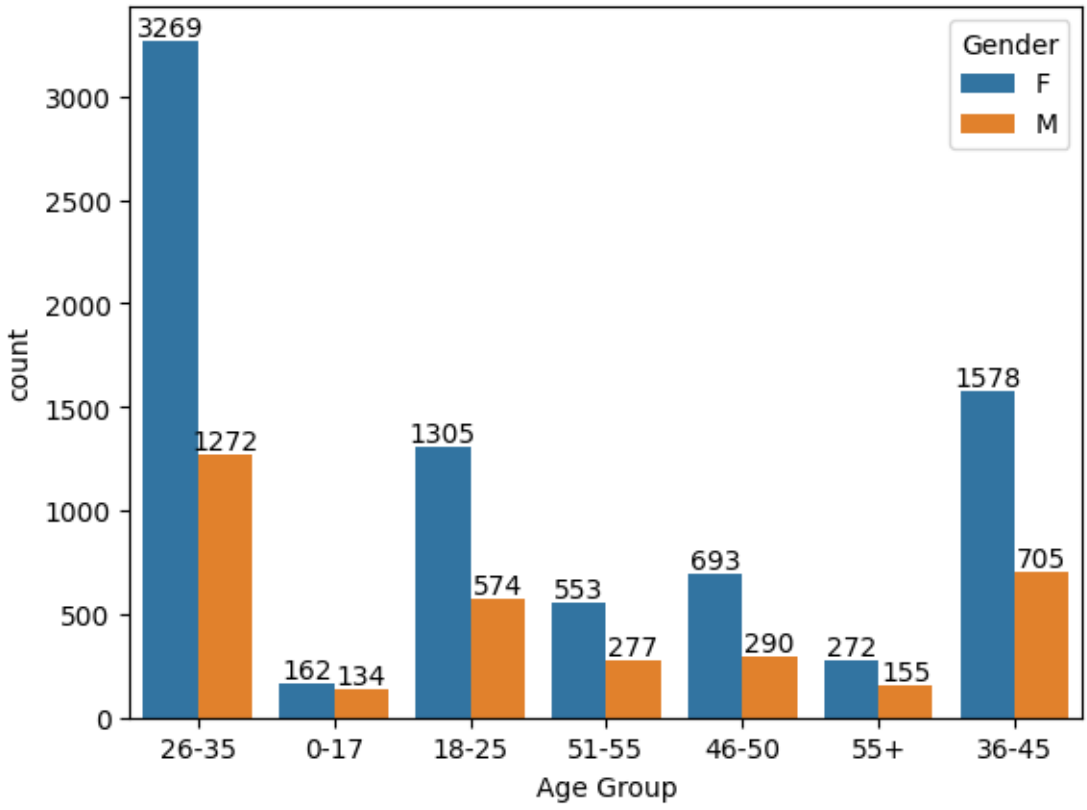
[46]: <Axes: xlabel='Gender', ylabel='Amount'>



From above graphs we can see that most of the buyers are females and even the purchasing power of females are greater than men

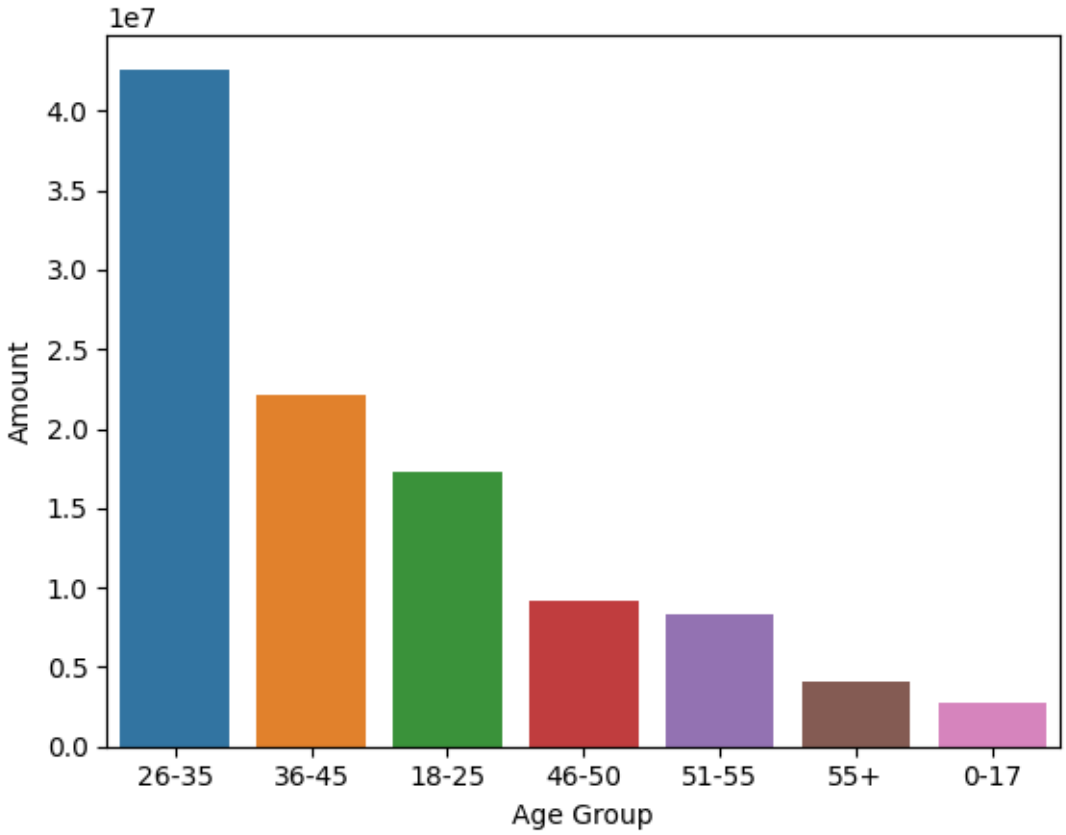
1.0.2 Age

[47]:
```
ax = sns.countplot(data = df, x = 'Age Group', hue = 'Gender')

for bars in ax.containers:
    ax.bar_label(bars)
```



[ ]:
```
# Total Amount vs Age Group
sales_age = df.groupby(['Age Group'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False)

sns.barplot(x = 'Age Group',y= 'Amount' ,data = sales_age)
```

[ ]: <Axes: xlabel='Age Group', ylabel='Amount'>



From above graphs we can see that most of the buyers are of age group between 26-35 yrs female

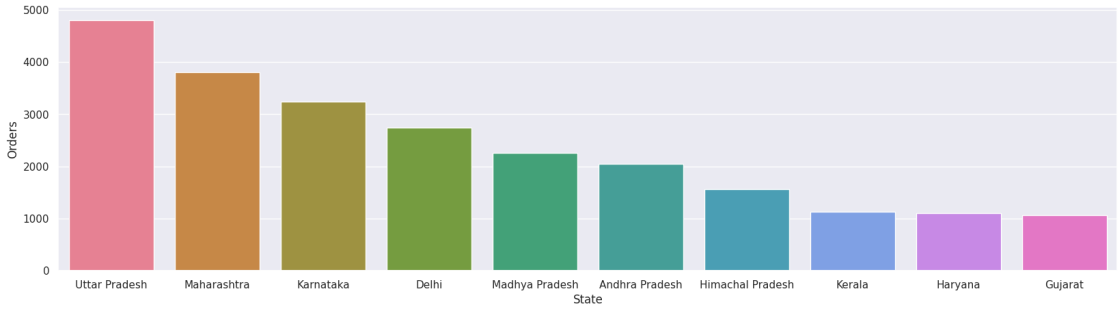
1.0.3 State

[55]:
```
# Group by State and calculate the total orders for the top 10 states
sales_state = df.groupby(['State'], as_index=False)['Orders'].sum().
 ↪sort_values(by='Orders', ascending=False).head(10)

# Set the figure size
sns.set(rc={'figure.figsize':(20, 5)})

# Use a custom color palette with 10 unique colors
colors = sns.color_palette("husl", len(sales_state))  # "husl" provides a range␣
 ↪of unique colors

# Plot with different colors for each bar
sns.barplot(data=sales_state, x='State', y='Orders', palette=colors)
```

```
<ipython-input-55-950919f2b194>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
```

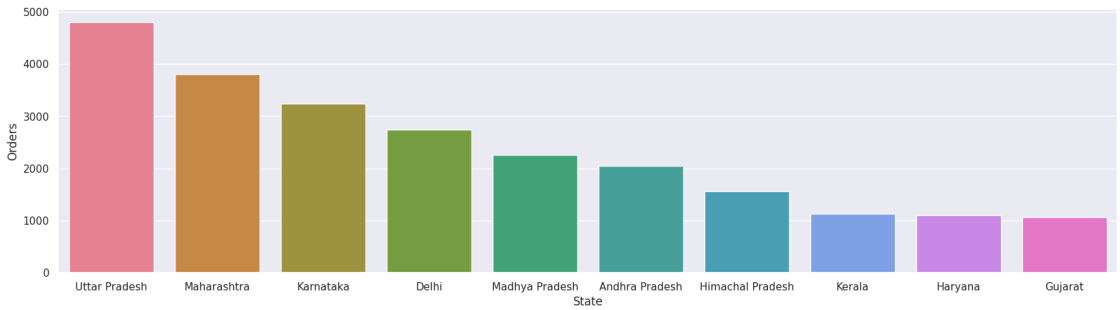
```
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=sales_state, x='State', y='Orders', palette=colors)
```

[55]: <Axes: xlabel='State', ylabel='Orders'>



[63]:
```
#for the version

# Group by State and calculate the total orders for the top 10 states
sales_state = df.groupby(['State'], as_index=False)['Orders'].sum().
 ↪sort_values(by='Orders', ascending=False).head(10)

# Set the figure size
sns.set(rc={'figure.figsize': (20, 5)})

# Use a custom color palette with 10 unique colors
colors = sns.color_palette("husl", len(sales_state))  # "husl" provides␣
 ↪distinct colors

# Plot with different colors for each bar by setting `hue` and disabling the␣
 ↪legend
sns.barplot(data=sales_state, x='State', y='Orders', hue='State',␣
 ↪palette=colors, dodge=False, legend=False)
```

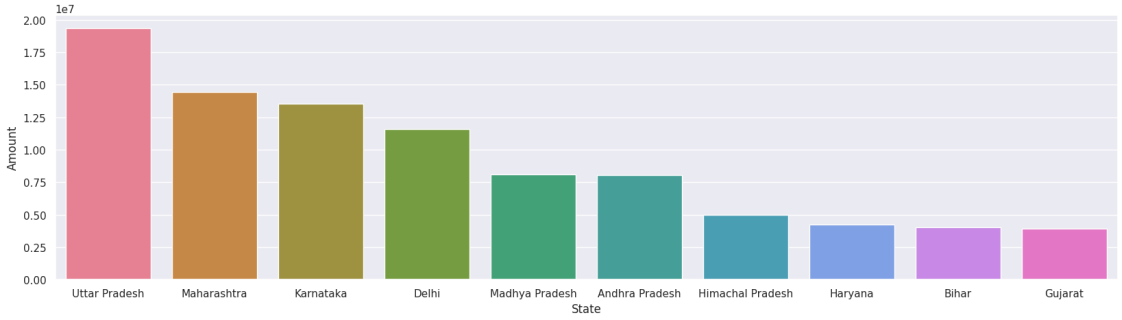
[63]: <Axes: xlabel='State', ylabel='Orders'>



[56]:
```
# Group by State and calculate the total amount for the top 10 states
sales_state = df.groupby(['State'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False).head(10)

# Set the figure size
sns.set(rc={'figure.figsize':(20, 5)})

# Use a custom color palette with 10 unique colors
colors = sns.color_palette("husl", len(sales_state))  # "husl" provides␣
 ↪distinct colors

# Plot with different colors for each bar
sns.barplot(data=sales_state, x='State', y='Amount', palette=colors)
```

```
<ipython-input-56-e27683afeb11>:11: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(data=sales_state, x='State', y='Amount', palette=colors)
```

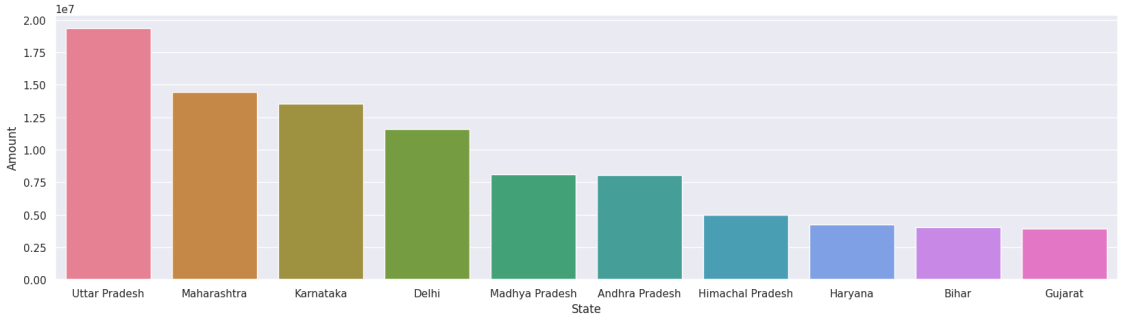
[56]:
```
<Axes: xlabel='State', ylabel='Amount'>
```



[64]:
```
#for the version

# Group by State and calculate the total amount for the top 10 states
sales_state = df.groupby(['State'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False).head(10)

# Set the figure size
sns.set(rc={'figure.figsize': (20, 5)})
```

```
# Use a custom color palette with 10 unique colors
colors = sns.color_palette("husl", len(sales_state))  # "husl" provides␣
 ↪distinct colors

# Plot with different colors for each bar by setting `hue` and disabling the␣
 ↪legend
sns.barplot(data=sales_state, x='State', y='Amount', hue='State',␣
 ↪palette=colors, dodge=False, legend=False)
```

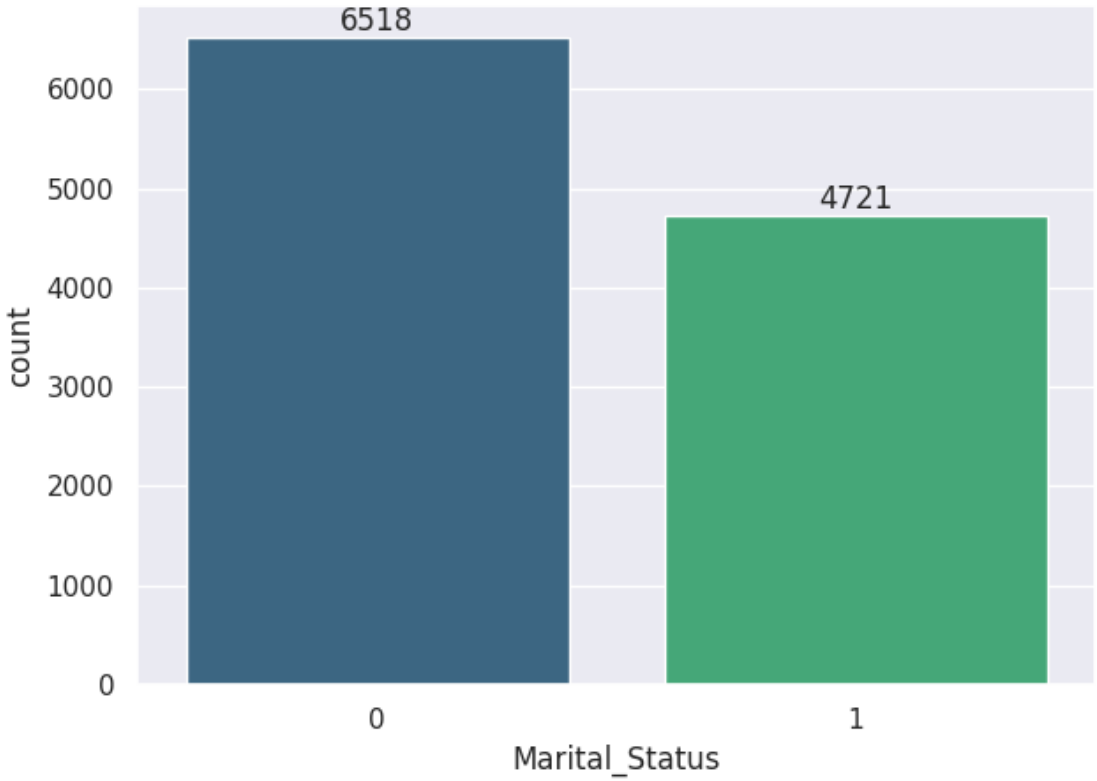
[64]: <Axes: xlabel='State', ylabel='Amount'>



From above graphs we can see that most of the orders & total sales/amount are from Uttar Pradesh, Maharashtra and Karnataka respectively

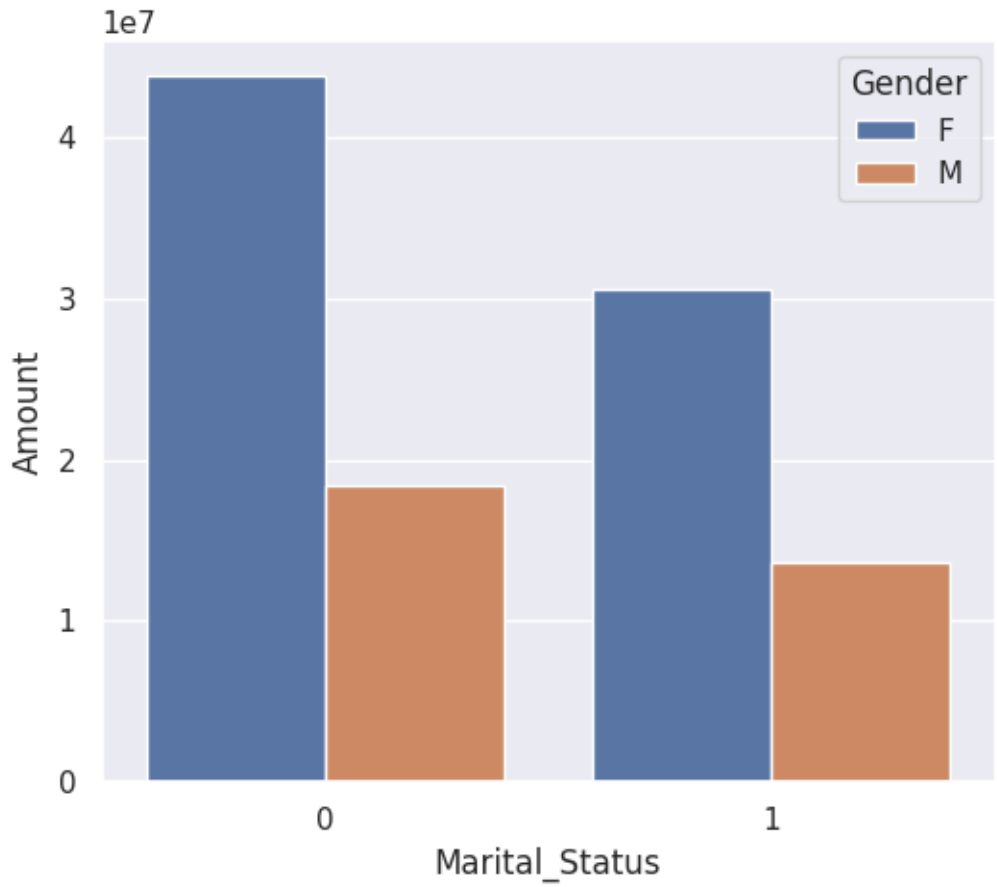
1.0.4 Marital Status

[62]:
```
# Set the figure size
sns.set(rc={'figure.figsize': (7, 5)})

# Plot the countplot with a custom color palette, assigning `Marital_Status` to␣
 ↪`hue` and disabling the legend
ax = sns.countplot(data=df, x='Marital_Status', hue='Marital_Status',␣
 ↪palette="viridis", legend=False)

# Display the values on top of each bar
for bars in ax.containers:
    ax.bar_label(bars)
```



[65]:
```
sales_state = df.groupby(['Marital_Status', 'Gender'],␣
↪as_index=False)['Amount'].sum().sort_values(by='Amount', ascending=False)

sns.set(rc={'figure.figsize':(6,5)})
sns.barplot(data = sales_state, x = 'Marital_Status',y= 'Amount', hue='Gender')
```

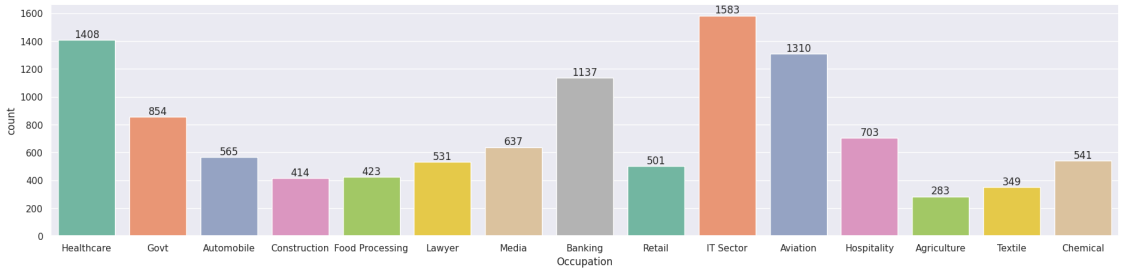
[65]:
```
<Axes: xlabel='Marital_Status', ylabel='Amount'>
```



From above graphs we can see that most of the buyers are married (women) and they have high purchasing power

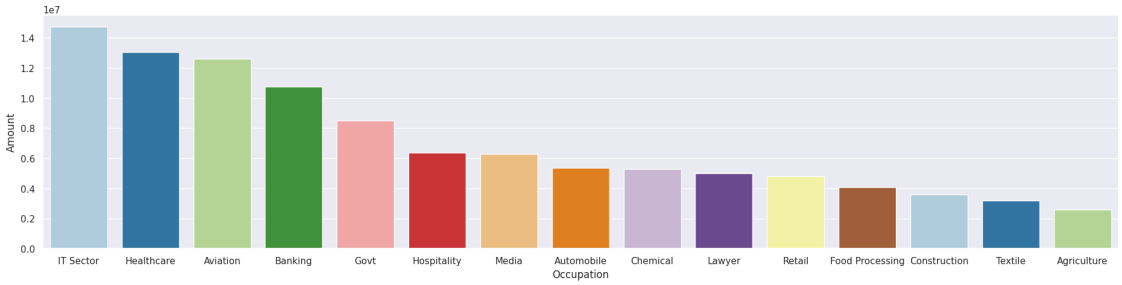
1.0.5 Occupation

[77]:
```
# Set the figure size
sns.set(rc={'figure.figsize': (23, 5)})

# Plot the countplot with the Set2 color palette
ax = sns.countplot(data=df, x='Occupation', hue='Occupation', palette="Set2",␣
 ↪legend=False)
#used a different palette
# Display the values on top of each bar
for bars in ax.containers:
    ax.bar_label(bars)
```



[78]:
```
# Group by Occupation and calculate the total amount
sales_state = df.groupby(['Occupation'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False)

# Set the figure size
sns.set(rc={'figure.figsize': (23, 5)})

# Plot with the Paired color palette, assigning hue to 'Occupation' and␣
 ↪disabling the legend
sns.barplot(data=sales_state, x='Occupation', y='Amount', hue='Occupation',␣
 ↪palette="Paired", dodge=False, legend=False)
```

[78]:
```
<Axes: xlabel='Occupation', ylabel='Amount'>
```

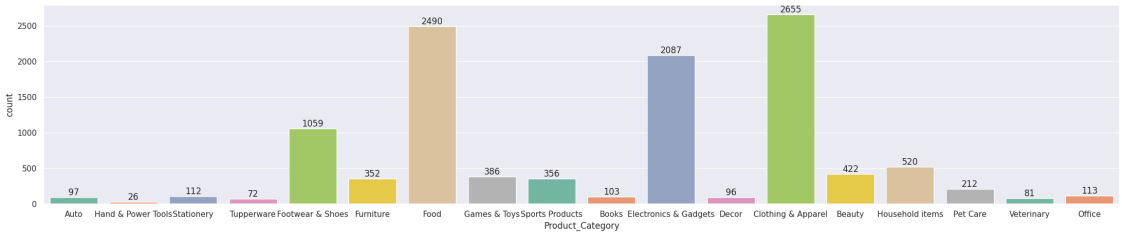


From above graphs we can see that most of the buyers are working in IT, Healthcare and Aviation sector

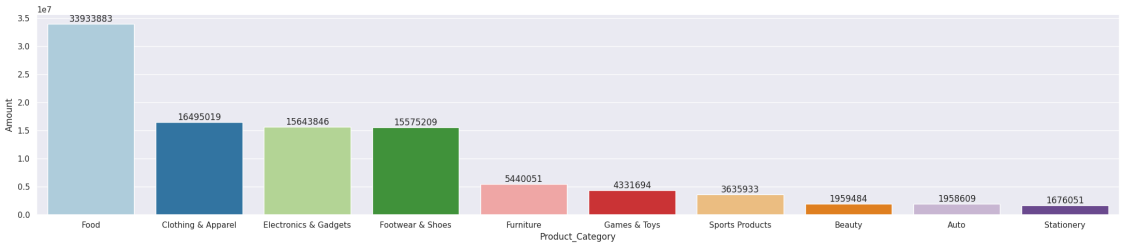
1.0.6 Product Category

[87]:
```
# Set the figure size
sns.set(rc={'figure.figsize': (27, 5)})

# Plot the countplot with a specified color palette
ax = sns.countplot(data=df, x='Product_Category', hue='Product_Category',␣
 ↪palette='Set2', legend=False)
```

```
# Display the values on top of each bar
for bars in ax.containers:
    ax.bar_label(bars)
```



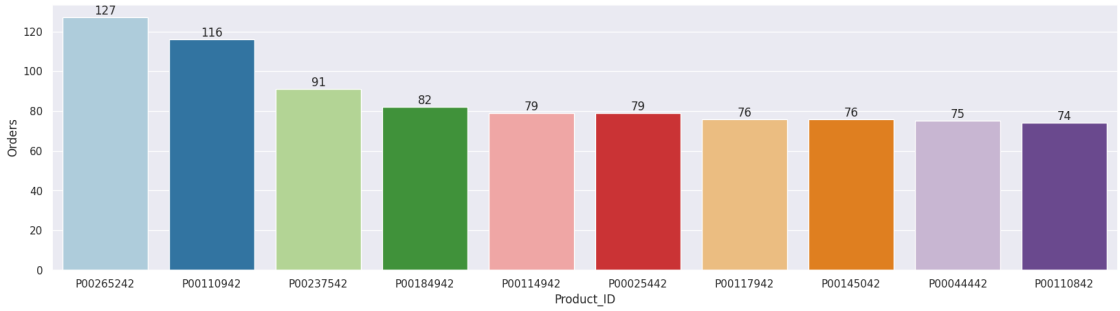
[92]:
```
sales_state = df.groupby(['Product_Category'], as_index=False)['Amount'].sum().
 ↪sort_values(by='Amount', ascending=False).head(10)

sns.set(rc={'figure.figsize': (27, 5)})
sns.barplot(data=sales_state, x='Product_Category', y='Amount',␣
 ↪hue='Product_Category', palette="Paired", dodge=False)

# Optionally, display the values on top of each bar
for p in plt.gca().patches:
    plt.annotate(format(p.get_height(), '.0f'),
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='bottom')  # va: vertical alignment
```



From above graphs we can see that most of the sold products are from Food, Clothing and Electronics category

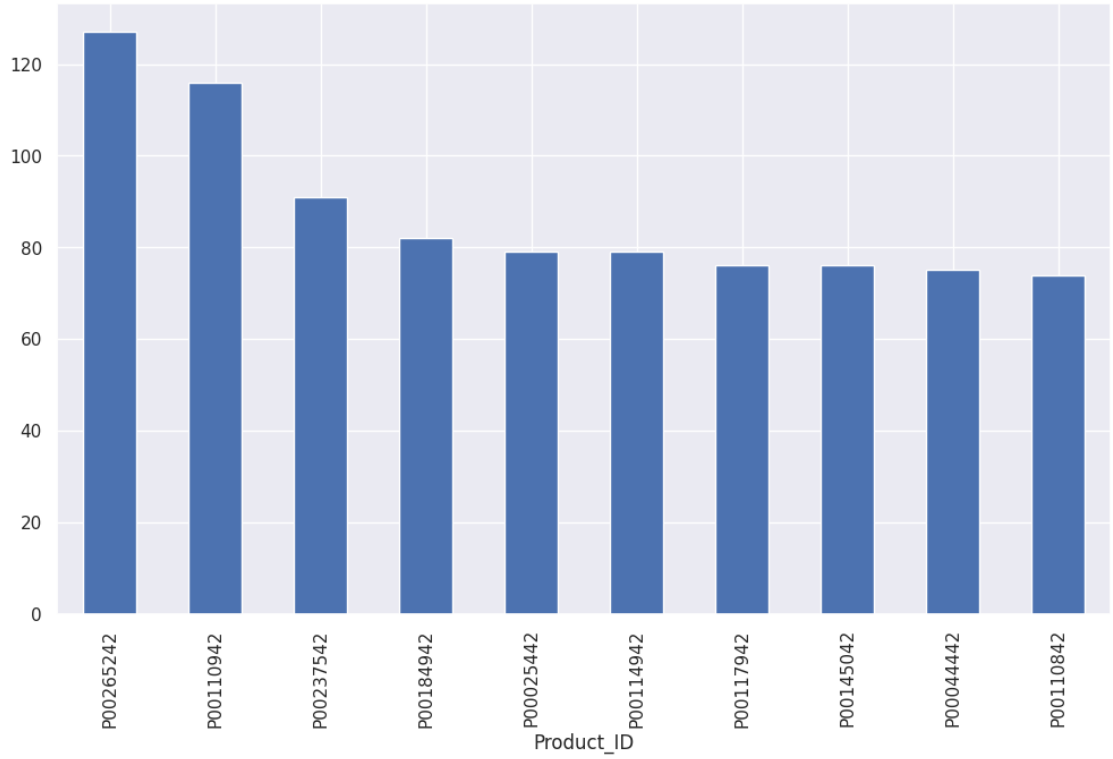
[94]:
```
sales_state = df.groupby(['Product_ID'], as_index=False)['Orders'].sum().
 ↪sort_values(by='Orders', ascending=False).head(10)

sns.set(rc={'figure.figsize': (20, 5)})
sns.barplot(data=sales_state, x='Product_ID', y='Orders', hue='Product_ID',␣
 ↪palette="Paired", dodge=False)
```

```
# Optionally, display the values on top of each bar
for p in plt.gca().patches:
    plt.annotate(format(p.get_height(), '.0f'),
                 (p.get_x() + p.get_width() / 2., p.get_height()),
                 ha='center', va='bottom')  # va: vertical alignment
```



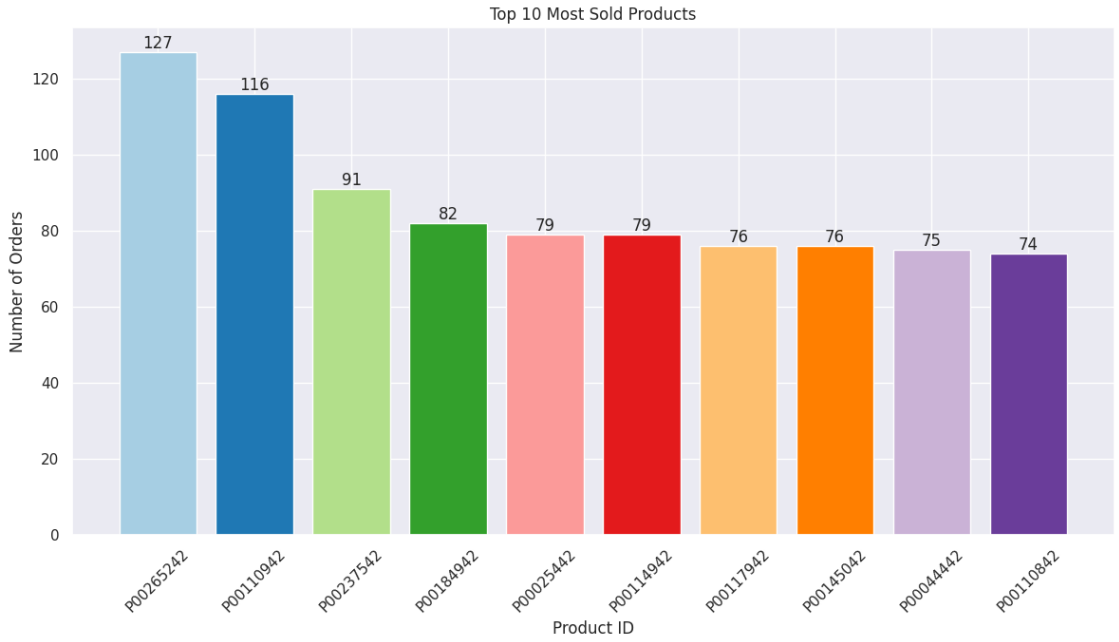
[95]:
```
# top 10 most sold products (same thing as above)

fig1, ax1 = plt.subplots(figsize=(12,7))
df.groupby('Product_ID')['Orders'].sum().nlargest(10).
 ↪sort_values(ascending=False).plot(kind='bar')
```

[95]: <Axes: xlabel='Product_ID'>



[96]:
```
import matplotlib.pyplot as plt
import pandas as pd

# Calculate the top 10 most sold products
top_products = df.groupby('Product_ID')['Orders'].sum().nlargest(10).
 ↪sort_values(ascending=False)

# Create a bar plot with different colors
fig1, ax1 = plt.subplots(figsize=(12, 7))
bars = ax1.bar(top_products.index, top_products.values, color=plt.cm.
 ↪Paired(range(len(top_products))))

# Optionally, display the values on top of each bar
for bar in bars:
    ax1.text(bar.get_x() + bar.get_width() / 2, bar.get_height(),
             f'{int(bar.get_height())}', ha='center', va='bottom')

# Set labels and title
ax1.set_xlabel('Product ID')
ax1.set_ylabel('Number of Orders')
ax1.set_title('Top 10 Most Sold Products')
```

```
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.tight_layout()  # Adjust layout
plt.show()
```



##**Conclusion:**

Married women age group 26-35 yrs from UP, Maharastra and Karnataka working in IT, Healthcare and Aviation are more likely to buy products from Food, Clothing and Electronics category

Performed data cleaning and manipulation

Performed exploratory data analysis (EDA) using pandas, matplotlib and seaborn libraries

Improved customer experience by identifying potential customers across different states, occupation, gender and age groups

Improved sales by identifying most selling product categories and products, which can help to plan inventory and hence meet the demands