CREDIT CARD ANALYSIS

using Python

PROJECT OBJECTIVE

Analyze credit card transaction data to identify patterns, assess risk factors, and optimize lending strategies. Explore spending behavior, repayment trends, enabling stakeholders to monitor and analyze credit card operations effectively.

TOOLS USED

1. Pandas

Pandas is a powerful library for data manipulation and analysis. It simplifies working with structured data, such as tabular data or time series. we can use it to read data from various sources , clean and transform data, and perform aggregation tasks.

2. Matplotlib & Seaborn

Matplotlib is a popular plotting library that allows us to create various types of visualizations, including line plots, scatter plots, histograms, and more. It’s essential for exploring data and presenting insights visually.

Seaborn is a statistical data visualization library built on top of Matplotlib. It simplifies creating attractive and informative plots with minimal code. Seaborn provides several predefined themes that enhance the aesthetics of the plots.

3. Plotly

Plotly is an open-source graphing library that enables users to create interactive and aesthetically appealing visualizations in Python, R, JavaScript, and other programming languages. Plotly supports a wide variety of chart types, including line charts, scatter plots, bar charts, and 3D graphs, among others

4. Jupyter Notebook

Jupyter Notebook which provides an interactive environment for writing and executing Python code. It’s widely used for data analysis because it allows us to combine code, visualizations, and explanatory text in a single document.

Import Necesary Libraries

To perform our data analysis, we'll use several powerful Python libraries, including pandas for data manipulation. Matplotlib, seaborn and Plotly for data visualization.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import plotly.express as px
```

Reading Data from csv file

```
cc_details = pd.read_csv('credit_card.csv')
```

```
cc_details.head()
```

```
cust_details = pd.read_csv('customer.csv')
```

```
cust_details.head()
```

Data Frames

We have two data frames, cc details and cust details, Let’s take a first look at them!

cc_details

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_Start_Date	Week_Num	Qtr	current_year	Credit_Limit	Total_Revolving_Bal	Total_Trans_Amt	Total_Trans_Vol
0	708082083	Blue	200	0	87	01-01-2023	Week-1	Q1	2023	3544.0	1661	15149	111
1	708083283	Blue	445	1	108	01-01-2023	Week-1	Q1	2023	3421.0	2517	992	21
2	708084558	Blue	140	0	106	01-01-2023	Week-1	Q1	2023	8258.0	1771	1447	23
3	708085458	Blue	250	1	150	01-01-2023	Week-1	Q1	2023	1438.3	0	3940	82
4	708086958	Blue	320	1	106	01-01-2023	Week-1	Q1	2023	3128.0	749	4369	59

cust_details

	Client_Num	Customer_Age	Gender	Dependent_Count	Education_Level	Marital_Status	state_cd	Zipcode	Car_Owner	House_Owner	Personal_loan	contact	Customer_Job	Income	Cust_Satisfaction_Score
0	708082083	24	F	1	Uneducated	Single	FL	91750	no	yes	no	unknown	Businessman	202326	3
1	708083283	62	F	0	Unknown	Married	NJ	91750	no	no	no	cellular	Selfemployeed	5225	2
2	708084558	32	F	1	Unknown	Married	NJ	91750	yes	no	no	unknown	Selfemployeed	14235	2
3	708085458	38	M	2	Uneducated	Single	NY	91750	no	no	no	cellular	Blue-collar	45683	1
4	708086958	48	M	4	Graduate	Single	TX	91750	yes	yes	no	cellular	Businessman	59279	1

Data Processing

1. Adding age group column in cust details data frame using lambda function

```
[ ]  #Adding a Age group column from Customer_Age column
     # used lambda function to create this column

     cust_details['Age_Group'] = cust_details['Customer_Age'].apply(lambda age:
                                 '20-30' if age < 30 else
                                 '30-40' if age >= 30 and  age < 40 else
                                 '40-50' if age >= 40 and  age < 50 else
                                 '50-60' if age >= 50 and  age < 60 else
                                 '60+' if age >=60 else
                                 'unknown')
```

output

Age_Group
20-30
60+
30-40
30-40
40-50

Data Processing

2. Adding income group column in cust details using lambda function

```
#Adding Revenue column in cc_details
cc_details['Revenue'] = cc_details['Annual_Fees'] + cc_details['Total_Trans_Amt'] + cc_details['Interest_Earned']
```

output

Revenue
19742.21
1506.44
1789.58
4426.40
5693.87

Data Processing

3. Adding Revenue column in cc details data frame

```
[ ]  #Adding an Income group column from Income column


     cust_details['Income_Group'] = cust_details['Income'].apply(lambda income:
                                    'Low' if income < 35000 else
                                    'Medium' if income >= 35000 and  income < 70000 else
                                    'High' if income >=70000 else
                                    'unknown')
```

output

Income_Group
High
Low
Low
Medium
Medium

Data Processing

4. Changing data type of ‘week_start_date’ column

We are changing the data type of this date column to perform some date and time manipulation, such as extracting specific parts of the date (year, month, day), performing arithmetic operations (e.g., adding or subtracting days), and comparing dates.

```
[ ]  cc_details['Week_Start_Date'] = pd.to_datetime(cc_details['Week_Start_Date'], format='%d-%m-%Y')
```

```
[ ]  cc_details.dtypes
```

```
Client_Num                int64
Card_Category            object
Annual_Fees               int64
Activation_30_Days        int64
Customer_Acq_Cost         int64
Week_Start_Date    datetime64[ns]
Week_Num                 object
Qtr                      object
current_year              int64
Credit_Limit            float64
Total_Revolving_Bal       int64
Total_Trans_Amt           int64
Total_Trans_Vol           int64
Avg_Utilization_Ratio   float64
Use Chip                 object
Exp Type                 object
Interest_Earned         float64
Delinquent_Acc            int64
Revenue                 float64
dtype: object
```

Data Processing

5. Adding Month and Week Number column from ‘Week_Start_Date’ column

```
[ ]  #Adding Month column
     cc_details['Month'] = cc_details['Week_Start_Date'].dt.strftime('%B')

     #Adding week number column
     cc_details['Week_Number'] = (cc_details['Week_Start_Date'] - pd.Timestamp('2023-01-01')).dt.days // 7 + 1
```

```
[ ]  cc_details.head()
```

Month	Week_Number
January	1
January	1
January	1
January	1
January	1

Data Processing

6. Merging two Data frames

We are Merging two Data frames; ‘cc_details’ and ‘cust_details’ for further analysis and data visualisations

```
[ ]  #Merge two dataframes,cc_details and cust_details

     merged_df = pd.merge(cc_details, cust_details, on='Client_Num')
```

	Client_Num	Card_Category	Annual_Fees	Activation_30_Days	Customer_Acq_Cost	Week_Start_Date	Week_Num	Qtr	current_year	Credit_Limit	...	Zipcode	Car_Owner	House_Owner	Personal_loan	contact	Customer_Job	Income
0	708082083	Blue	200	0	87	2023-01-01	Week-1	Q1	2023	3544.0	...	91750	no	yes	no	unknown	Businessman	202326
1	708083283	Blue	445	1	108	2023-01-01	Week-1	Q1	2023	3421.0	...	91750	no	no	no	cellular	Selfemployeed	5225
2	708084558	Blue	140	0	106	2023-01-01	Week-1	Q1	2023	8258.0	...	91750	yes	no	no	unknown	Selfemployeed	14235
3	708085458	Blue	250	1	150	2023-01-01	Week-1	Q1	2023	1438.3	...	91750	no	no	no	cellular	Blue-collar	45683
4	708086958	Blue	320	1	106	2023-01-01	Week-1	Q1	2023	3128.0	...	91750	yes	yes	no	cellular	Businessman	59279

5 rows × 37 columns

The Analysis

Total Revenue

```
[ ]  #Sum of Revenue
     cc_details['Revenue'].sum()
```

```
55315410.23
```

Total Interest Earned

```
[ ]  #Sum of Interest Earned
     cc_details['Interest_Earned'].sum().round(2)
```

```
7843382.23
```

Total Transactions Amount

```
[ ]  cc_details['Total_Trans_Amt'].sum()
```

```
44522013
```

Average Age

```
[ ]  cust_details['Customer_Age'].mean().round()
```

```
46.0
```

55M
Revenue

7.9M
Total Interest

45M
Trans. Amount

46
Avg Age

The Analysis

Customer Satisfaction Score

```
[ ] cust_details['Cust_Satisfaction_Score'].mean().round(2)
```

```
3.19
```

Deliquent Rate

```
[ ] Delinquent_Acc = cc_details['Delinquent_Acc'].value_counts()
```

```
[ ] Delinquent_Acc
```

```
Delinquent_Acc
0    9494
1     614
Name: count, dtype: int64
```

```
[ ] non_Deliquent = (cc_details['Delinquent_Acc'] == 0).sum()
    Deliquent = (cc_details['Delinquent_Acc'] == 1).sum()
    Delinquent_Acc_Total = Delinquent_Acc.sum()
```

```
[ ] Delinquent_rate = (Deliquent/Delinquent_Acc_Total)*100
```

```
[ ] Delinquent_rate.round(2)
```

```
6.07
```

3.19
CSS

614
Deliquent Accts.

6.07%
Deliquent Rate

*CSS :- Customer Satisfaction Score

The Analysis

5809

Activated Accts.

57%

Activation Rate

Activation Rate

```
[ ] Activation = cc_details['Activation_30_Days'].value_counts()

[ ] Activation

Activation_30_Days
1    5809
0    4299
Name: count, dtype: int64

[ ] non_Activated = (cc_details['Activation_30_Days'] == 0).sum()
    Activated = (cc_details['Activation_30_Days'] == 1).sum()
    Activation_Total = Activation.sum()

[ ] Activation_rate = (Activated/Activation_Total)*100

[ ] Activation_rate.round(2)

57.47
```

Activation count by Customer Job

Activation count by Customer Job

```
[ ] Activation_job = merged_df.groupby('Customer_Job')['Activation_30_Days'].value_counts()

[ ] Activation_job

Customer_Job   Activation_30_Days
Blue-collar    1                    902
               0                    677
Businessman    1                   1096
               0                    805
Govt           1                    883
               0                    642
Retirees       1                    558
               0                    428
Selfemployeed  1                   1495
               0                   1080
White-collar   1                    875
               0                    667
Name: count, dtype: int64
```

The Analysis

Week over Week Change

```
#current week revenue
wow_revenue = cc_details.groupby('Week_Number')['Revenue'].sum().reset_index()

# Rename the columns
wow_revenue.columns = ['Week Number', 'Current_Week_Revenue']

# previous week's revenue
wow_revenue['Previous_Week_Revenue'] = wow_revenue['Current_Week_Revenue'].shift(1)

#Week over Week change%
wow_revenue['WoW_change'] = (wow_revenue['Current_Week_Revenue'] - wow_revenue['Previous_Week_Revenue']) / wow_revenue['Previous_Week_Revenue']

#round the values
wow_revenue['WoW_change'] = wow_revenue['WoW_change'].round(2)
```

The Analysis

Week over Week Change

```
wow_revenue
```

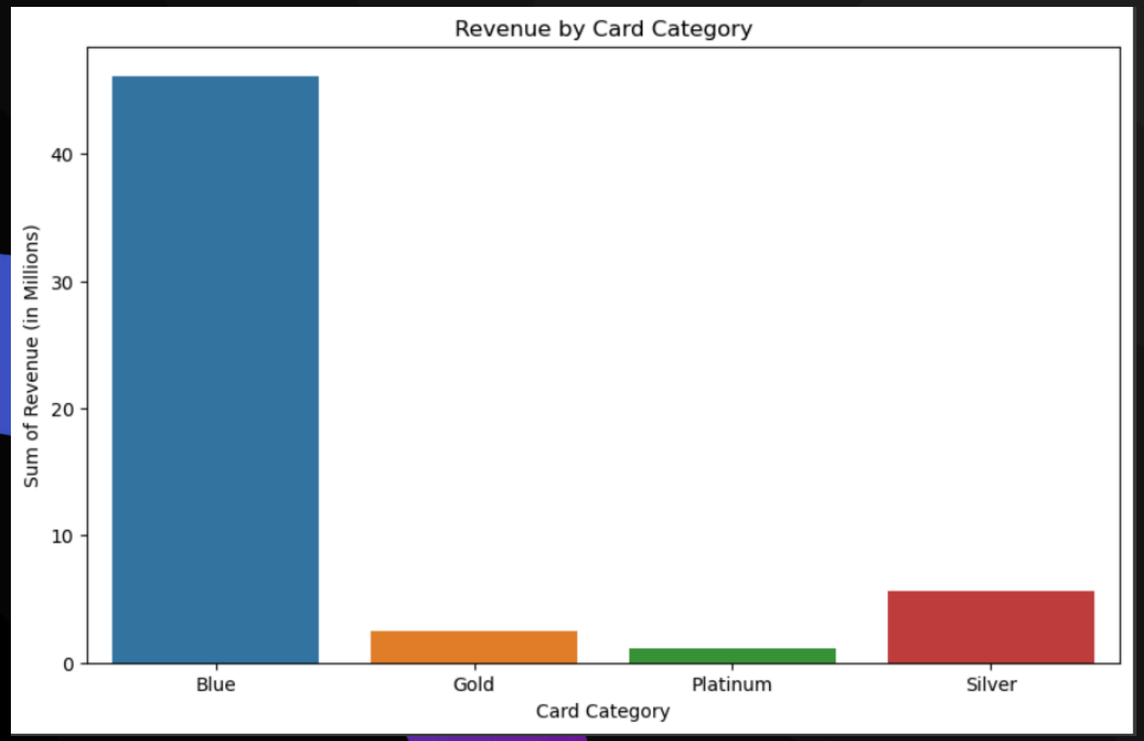
	Week Number	Current_Week_Revenue	Previous_Week_Revenue	WoW_change
0	1	1035629.32	NaN	NaN
1	2	1053088.81	1035629.32	1.69
2	3	1148249.80	1053088.81	9.04
3	4	1071919.27	1148249.80	-6.65
4	5	1064577.97	1071919.27	-0.68
5	6	1121745.13	1064577.97	5.37
6	7	1099909.39	1121745.13	-1.95
7	8	1071542.29	1099909.39	-2.58
8	9	1093501.86	1071542.29	2.05
9	10	987820.46	1093501.86	-9.66
10	11	1131280.79	987820.46	14.52
11	12	1106532.92	1131280.79	-2.19
12	13	978564.76	1106532.92	-11.56
13	14	1003843.69	978564.76	2.58
14	15	1056484.86	1003843.69	5.24
15	16	1082608.69	1056484.86	2.47
16	17	978441.30	1082608.69	-9.62
17	18	1063740.83	978441.30	8.72
18	19	1044817.36	1063740.83	-1.78
19	20	1035860.99	1044817.36	-0.86
20	21	1147869.75	1035860.99	10.81
21	22	1018518.29	1147869.75	-11.27
22	23	1070102.70	1018518.29	5.06
23	24	1099070.02	1070102.70	2.71
24	25	1115863.02	1099070.02	1.51
25	26	1103545.22	1115863.02	-1.09
26	27	1236294.67	1103545.22	12.03
27	28	1153364.56	1236294.67	-6.71
28	29	1041928.77	1153364.56	-9.66
29	30	1040273.26	1041928.77	-0.16
30	31	1182651.53	1040273.26	13.69

Data Visualization

matplotlib seaborn

Revenue by Card Category

```
[ ]  sum_revenue = cc_details.groupby('Card_Category')['Revenue'].sum().reset_index()

     plt.figure(figsize=(10, 6))
     sns.barplot(data=sum_revenue, x='Card_Category', y=sum_revenue['Revenue'] / 1e6, hue='Card_Category')
     plt.title('Revenue by Card Category')
     plt.xlabel('Card Category')
     plt.ylabel('Sum of Revenue (in Millions)')
     plt.show()
```

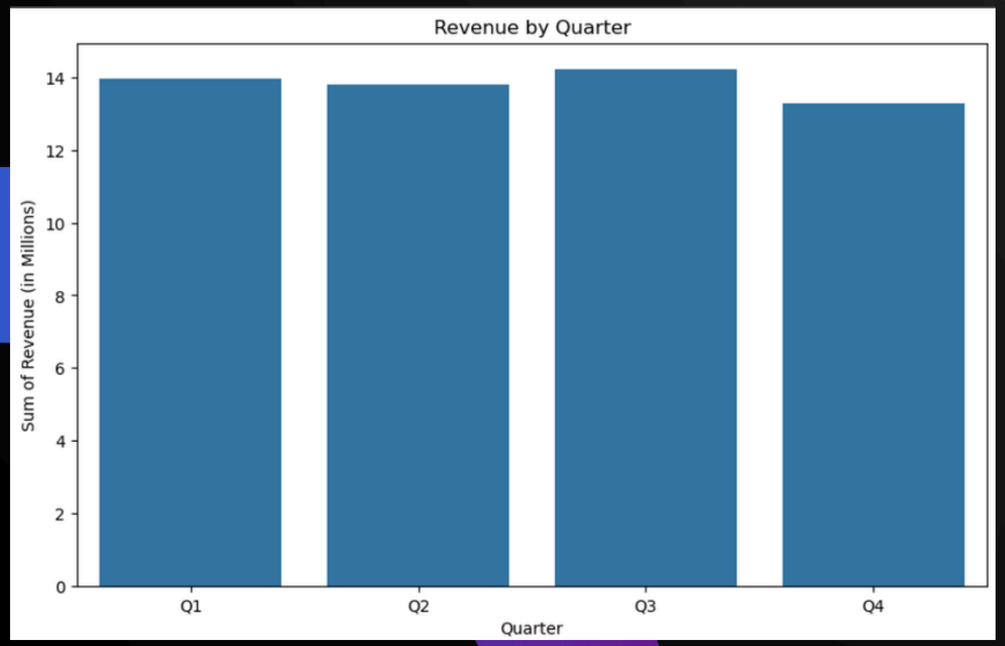


Data Visualization

matplotlib seaborn

Revenue by Quarter

```
[ ]  sum_revenue = cc_details.groupby('Qtr')['Revenue'].sum().reset_index()

     plt.figure(figsize=(10, 6))
     sns.barplot(data=sum_revenue, x='Qtr', y=sum_revenue['Revenue'] / 1e6)
     plt.title('Revenue by Quarter')
     plt.xlabel('Quarter')
     plt.ylabel('Sum of Revenue (in Millions)')
     plt.show()
```

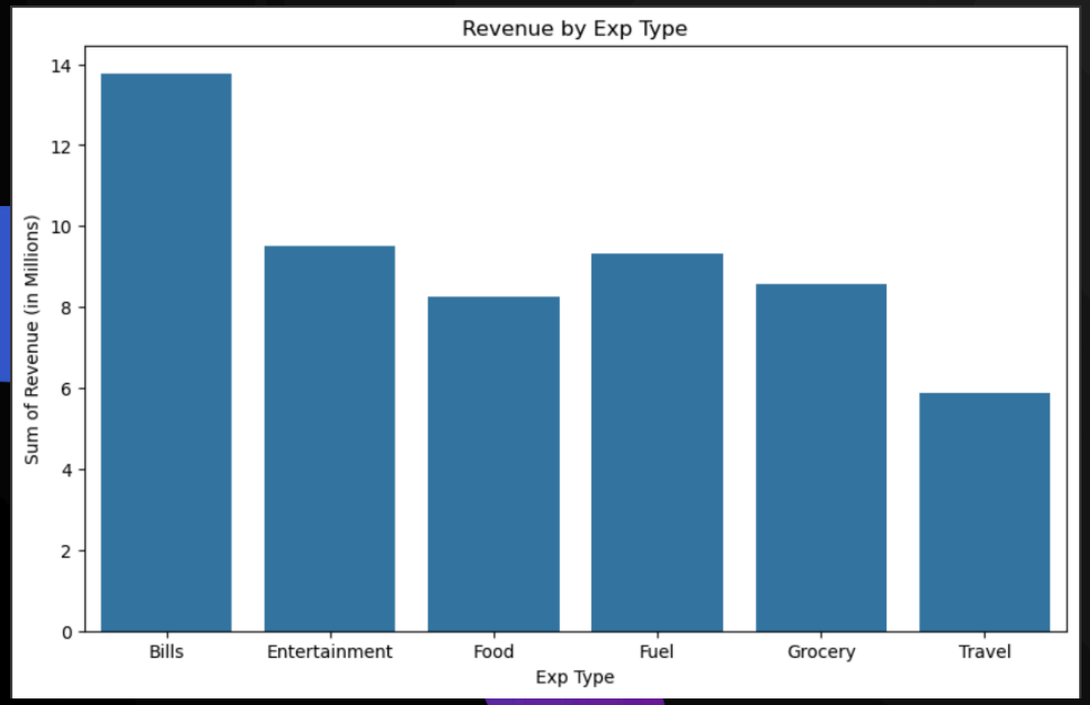


Data Visualization

matplotlib seaborn

Revenue by Expenditure Type

```
[ ]  sum_revenue = cc_details.groupby('Exp Type')['Revenue'].sum().reset_index()

     plt.figure(figsize=(10, 6))
     sns.barplot(data=sum_revenue, x='Exp Type', y=sum_revenue['Revenue'] / 1e6,palette=None)
     plt.title('Revenue by Exp Type')
     plt.xlabel('Exp Type')
     plt.ylabel('Sum of Revenue (in Millions)')
     plt.show()
```

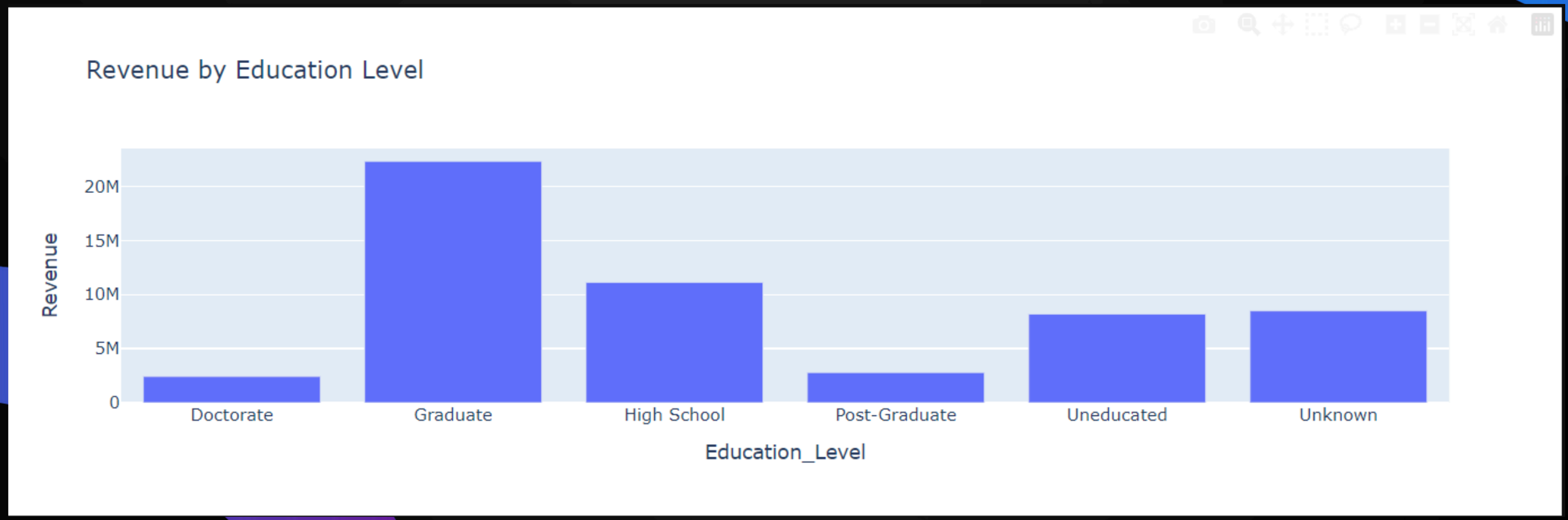


Data Visualization

plotly

Revenue by Education Level

```
sum_revenue = merged_df.groupby('Education_Level')['Revenue'].sum().reset_index()

fig = px.bar(sum_revenue,x='Education_Level', y='Revenue',title='Revenue by Education Level')
fig.show()
```

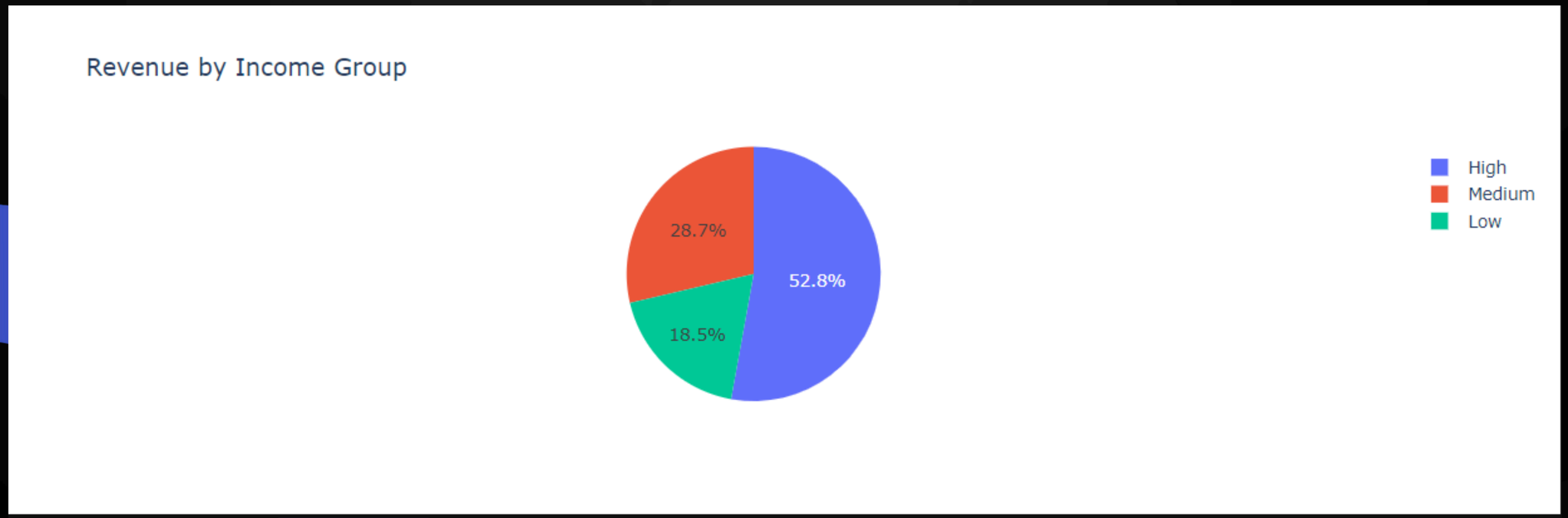


Data Visualization

plotly

Revenue by Income

```
rev_income = merged_df.groupby('Income_Group')['Revenue'].sum().reset_index()

fig = px.pie(rev_income, values='Revenue', names='Income_Group', title='Revenue by Income Group')

# Show the plot
fig.show()
```

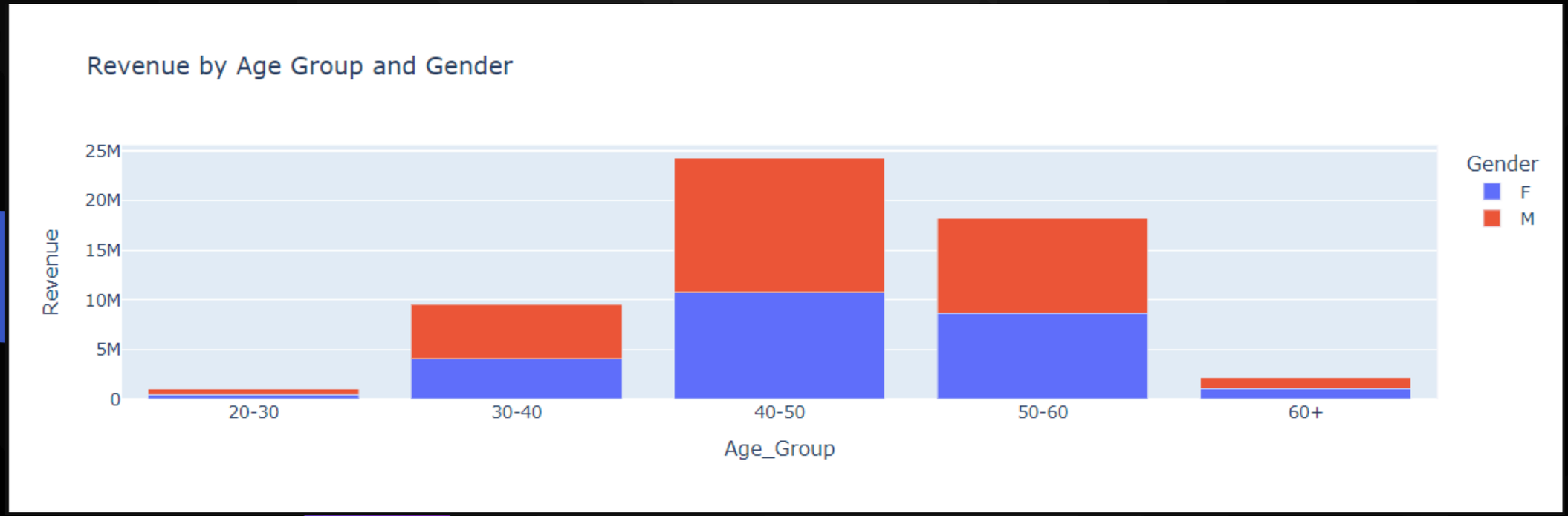


Data Visualization

plotly

Revenue by Age and Gender

```
rev_age = merged_df.groupby(['Age_Group','Gender'])['Revenue'].sum().reset_index()

fig = px.bar(rev_age, x='Age_Group', y='Revenue', color='Gender', title='Revenue by Age Group and Gender', barmode='stack'

fig.show()
```

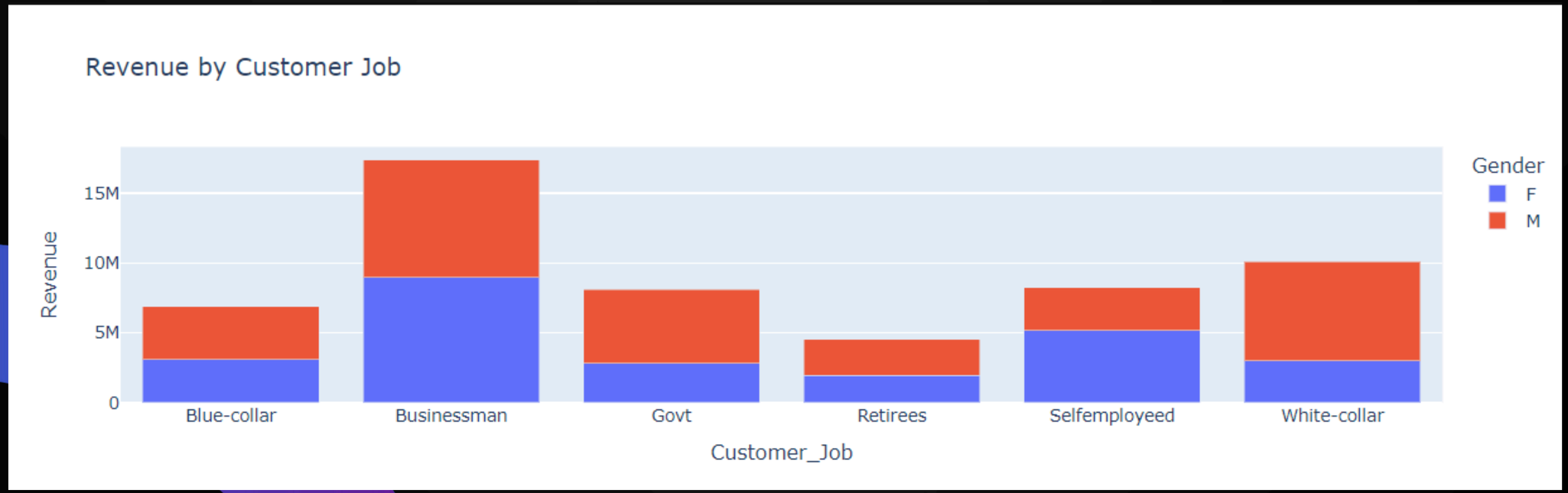


Data Visualization

plotly

Revenue by Customer Job

```
sum_revenue = merged_df.groupby(['Customer_Job','Gender'])['Revenue'].sum().reset_index()

fig = px.bar(sum_revenue,x='Customer_Job', y='Revenue',color='Gender', title='Revenue by Customer Job',barmode='stack')
fig.show()
```



Data Visualization

plotly

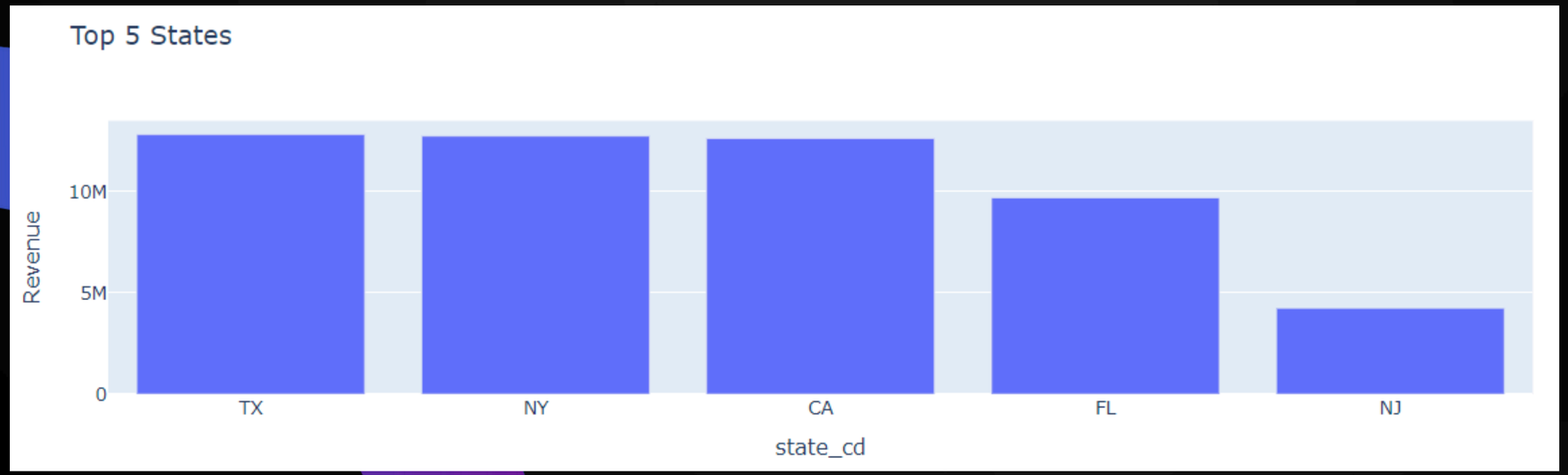
Top 5 States by Revenue

```
#Revenue by State
rev_state = merged_df.groupby('state_cd')['Revenue'].sum().reset_index()
rev_state = rev_state.sort_values(by='Revenue', ascending=False)
top_5_states = rev_state.head(5)
```

```
top_5_states
```

```
#Plotting
fig = px.bar(top_5_states,x='state_cd', y='Revenue',title='Top 5 States')
fig.show()
```

	state_cd	Revenue
24	TX	12809300.67
19	NY	12733454.75
3	CA	1261274.39
6	FL	9690947.18
16	NJ	4245441.70

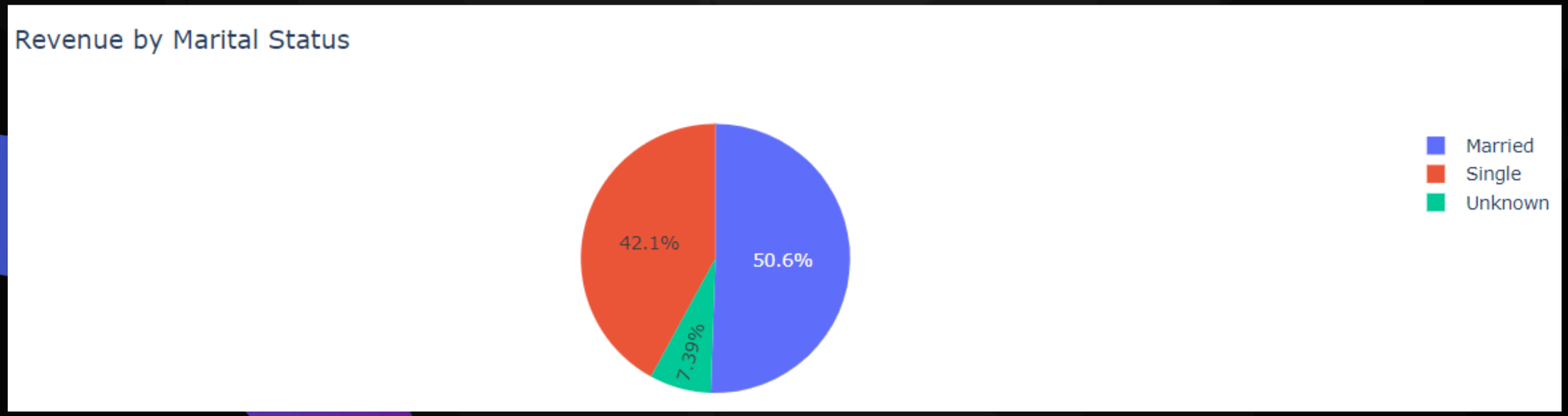


Data Visualization

plotly

Revenue by Marital Status

```
rev_status = merged_df.groupby('Marital_Status')['Revenue'].sum().reset_index()

fig = px.pie(rev_status, values='Revenue', names='Marital_Status', title='Revenue by Marital Status')

fig.show()
```

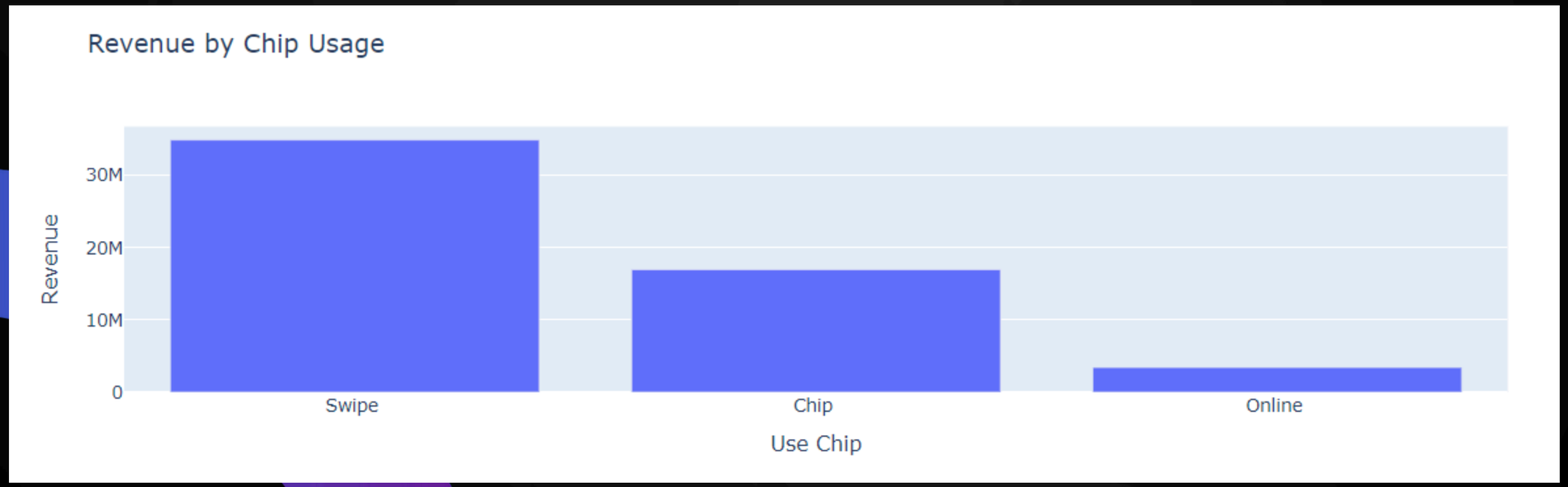


Data Visualization

plotly

Revenue by Chip Usage

```
sum_revenue = cc_details.groupby('Use Chip')['Revenue'].sum().reset_index()
sum_revenue = sum_revenue.sort_values(by='Revenue', ascending=False)

fig = px.bar(sum_revenue,x='Use Chip', y='Revenue',title='Revenue by Chip Usage')
fig.show()
```

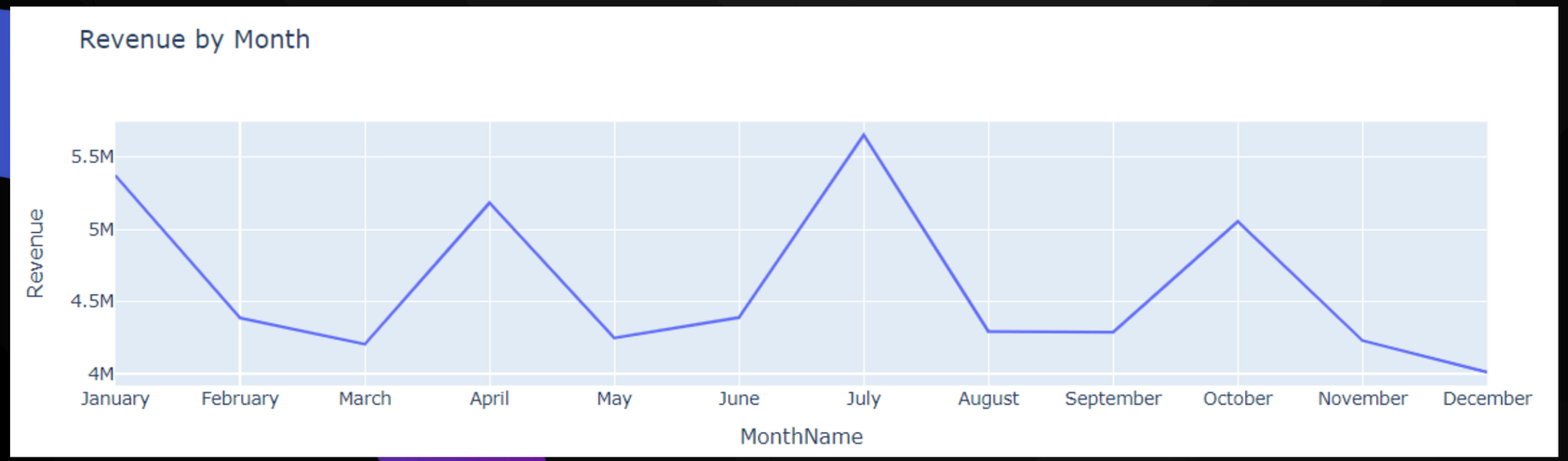


Data Visualization

plotly

Revenue by Month

```
revenue_by_month = cc_details.groupby('Month')['Revenue'].sum().reset_index()

# Convert month names to datetime objects to get the correct order
revenue_by_month['Month'] = pd.to_datetime(revenue_by_month['Month'], format='%B')

# Sort by the datetime objects (this sorts by month in chronological order)
revenue_by_month = revenue_by_month.sort_values(by='Month')

revenue_by_month['MonthName'] = revenue_by_month['Month'].dt.strftime('%B')

fig = px.line(revenue_by_month, x='MonthName', y='Revenue', title='Revenue by Month')

fig.show()
```

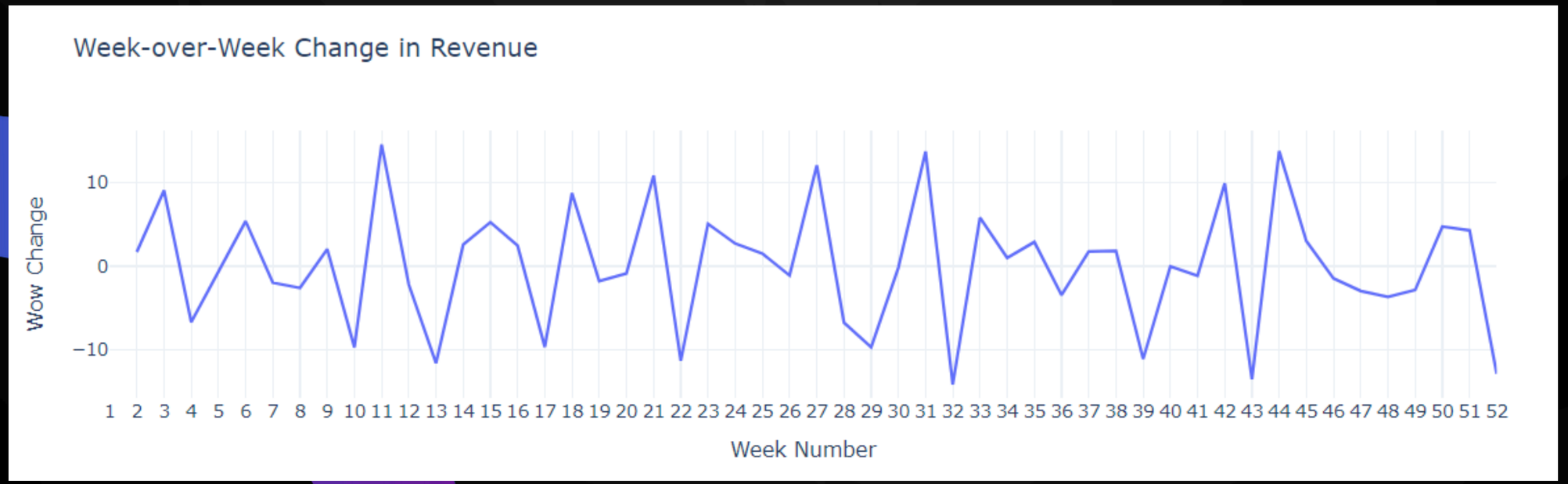


Data Visualization

plotly

Week over Week change in Revenue

```
fig = px.line(wow_revenue, x='Week Number', y='WoW_change', title='Week-over-Week Change in Revenue')
fig.update_layout(
    xaxis_title='Week Number',
    yaxis_title='Wow Change',
    xaxis=dict(tickmode='linear', tick0=1, dtick=1),  # Show every week
    template='plotly_white'
)
fig.show()
```



Data Visualization

Card Summary

```
#Table showing Card Category,Sum of Revenue, Sum of total trasaction amount, sum of interst earned
card_summary = cc_details.groupby('Card_Category')[['Revenue','Total_Trans_Amt','Interest_Earned']].sum().reset_index()
```

```
card_summary
```

	Card_Category	Revenue	Total_Trans_Amt	Interest_Earned
0	Blue	46139397.74	36957875	6495887.74
1	Gold	2454072.16	2024078	373784.16
2	Platinum	1135608.05	953314	161629.05
3	Silver	5586332.28	4586746	812081.28

INSIGHTS

- Overall revenue is 57M
- Total interest is 8M
- Total transaction amount is 45M
- Male customers are contributing more in revenue 31M, female 26M
- Blue & Silver credit card are contributing to 93% of overall transactions
- TX, NY & CA is contributing to 68%
- Overall Activation rate is 57.5%
- Overall Delinquent rate is 6.06%

THANK YOU