|  |  |
| --- | --- |
| Project Title | **Economic Data Analysis** |
| Tools | Python, ML, SQL, Excel |
| Domain | Finance Analyst |
| Project Difficulties level | intermediate |

Dataset : Dataset is available in the given link. You can download it at your convenience.

[Click](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing) [here](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing) [to](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing) [download](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing) [data](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing) [set](https://drive.google.com/file/d/1nStAt7MVLInAb551Hm2BOZxWpo3PhklH/view?usp=sharing)

**About Dataset**

The dataset contains information about sales transactions, including details such as the customer's age, gender, location, and the products sold.

The dataset includes data on both the cost of the product and the revenue generated from its sale, allowing for calculations of profit and profit margins.

The quantity column provides information on the volume of products sold, which could be used to analyze sales trends over time.

The dataset includes information on customer age and gender, which could be used to analyze purchasing behavior across different demographic groups.

The dataset likely includes both numeric and categorical data, which would require different types of analysis and visualization techniques.

Overall, the dataset appears to provide a comprehensive view of sales transactions, with the potential for analysis at multiple levels, including by product, customer, and location.

**Column Descriptors**

1. Year: This column represents the year in which the transaction occurred. It could be used to track trends over time or to filter the data based on a specific year or range of years.
2. Month: This column represents the month in which the transaction occurred. It could be used to track trends over time or to filter the data based on a specific month or range of months.
3. Customer Age: This column represents the age of the customer. It could be used to segment customers based on age ranges or to analyze the purchasing behavior of different age groups.
4. Customer Gender: This column represents the gender of the customer. It could be used to segment customers based on gender or to analyze the purchasing behavior of different genders.
5. Country: This column represents the country where the transaction occurred. It could be used to analyze sales by country or to filter the data based on a specific country or range of countries.
6. State: This column represents the state where the transaction occurred. It could be used to analyze sales by the state or to filter the data based on a specific state or range of states.
7. Product Category: This column represents the broad category of the product sold. It could be used to analyze sales by product category or to filter the data based on a specific product category.
8. Sub Category: This column represents the specific subcategory of the product sold. It could be used to analyze sales by subcategory or to filter the data based on a specific subcategory.
9. Quantity: This column represents the quantity of the product sold. It could be used to analyze sales volume or to calculate the total revenue generated from a particular product or product category.
10. Unit Cost: This column represents the cost of producing or acquiring one unit of the product. It could be used to calculate profit margins or to compare the costs of different products or product categories.
11. Unit Price: This column represents the price at which one unit of the product was sold. It could be used to analyze pricing strategies or to compare the prices of different products or product categories.
12. Cost: This column represents the total cost of the products sold, which is calculated as the product of the quantity and the unit cost. It could be used to analyze the cost structure of the business or to calculate the profit margin of each sale.
13. Revenue: This column represents the total revenue generated by the sales, which is calculated as the product of the quantity and the unit price. It could be used to analyze the overall sales performance of the business or to calculate the profit generated by each sale.

**Economic Data Analysis Project**

**Project Overview**

Objective: To analyze macroeconomic data to understand economic trends and their impact on markets.

**Steps to Follow:**

1. **Define the Scope and Objective**:

○ Identify the specific economic indicators to analyze (e.g., GDP, unemployment rate, inflation rate, interest rates).

○ Define the time frame and geographical scope (e.g., US economy over the past 10 years).

1. **Data Collection**:

○ Gather relevant data from reliable sources such as government databases, financial websites, and international organizations.

○ For this project, we'll use data from the World Bank and Federal Reserve Economic Data (FRED).

1. **Data Preparation**:

○ Clean the data to remove any inconsistencies or errors.

○ Combine data from different sources into a single dataset.

○ Use tools like Pandas for data cleaning and preparation.

1. **Exploratory Data Analysis (EDA)**:

○ Perform EDA to understand the data distribution and identify patterns.

○ Use visualization tools like Matplotlib and Seaborn to visualize the data.

1. **Statistical Analysis**:

○ Perform statistical analysis to identify correlations and trends.

○ Use tools like Python (Pandas, Statsmodels) for this purpose.

1. **Predictive Modeling**:

○ Build predictive models to forecast future economic trends.

* 1. Use machine learning algorithms like Linear Regression, ARIMA, or SARIMA.

1. **Reporting**:
   1. Summarize the findings in a comprehensive report.

○ Use visualizations to support the analysis and make the report more engaging.

**Example: You can get the basic idea how you can create a project from here**

**Detailed Python Code Example**

**Step-by-Step Implementation**

1. **Data Collection**:

○ Assume you have downloaded the GDP, unemployment rate, inflation rate, and interest rates data from the World Bank and FRED.

|  |
| --- |
| # Import necessary libraries import pandas as pd import numpy as np  import matplotlib.pyplot as plt import seaborn as sns  from statsmodels.tsa.seasonal import seasonal\_decompose from statsmodels.tsa.arima.model import ARIMA from sklearn.metrics import mean\_squared\_error  # Load the datasets gdp = pd.read\_csv('gdp.csv')  unemployment = pd.read\_csv('unemployment\_rate.csv') inflation = pd.read\_csv('inflation\_rate.csv') interest\_rate = pd.read\_csv('interest\_rate.csv')  # Display the first few rows of each dataset print(gdp.head()) print(unemployment.head()) |

print(inflation.head()) print(interest\_rate.head())

1. **Data Preparation**:

|  |
| --- |
| # Convert date columns to datetime format gdp['Date'] = pd.to\_datetime(gdp['Date'])  unemployment['Date'] = pd.to\_datetime(unemployment['Date']) inflation['Date'] = pd.to\_datetime(inflation['Date']) interest\_rate['Date'] = pd.to\_datetime(interest\_rate['Date'])  # Merge the datasets on the Date column  data = pd.merge(gdp, unemployment, on='Date', how='inner') data = pd.merge(data, inflation, on='Date', how='inner') data = pd.merge(data, interest\_rate, on='Date', how='inner')  # Rename columns for clarity  data.columns = ['Date', 'GDP', 'Unemployment\_Rate', 'Inflation\_Rate', 'Interest\_Rate']  # Display the first few rows of the merged dataset print(data.head()) |

1. **Exploratory Data Analysis (EDA)**:

|  |
| --- |
| # Set the date column as the index  data.set\_index('Date', inplace=True)  # Plot the time series data  plt.figure(figsize=(12, 8)) plt.subplot(2, 2, 1)  plt.plot(data['GDP'], label='GDP') plt.title('GDP Over Time') plt.legend()  plt.subplot(2, 2, 2)  plt.plot(data['Unemployment\_Rate'], label='Unemployment Rate') plt.title('Unemployment Rate Over Time') plt.legend()  plt.subplot(2, 2, 3)  plt.plot(data['Inflation\_Rate'], label='Inflation Rate') plt.title('Inflation Rate Over Time') plt.legend()  plt.subplot(2, 2, 4)  plt.plot(data['Interest\_Rate'], label='Interest Rate') plt.title('Interest Rate Over Time') plt.legend()  plt.tight\_layout() |

plt.show()

1. **Statistical Analysis**:

|  |
| --- |
| # Compute correlations between the economic indicators correlation\_matrix = data.corr()  # Plot the correlation matrix plt.figure(figsize=(8, 6))  sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm') plt.title('Correlation Matrix') plt.show() |

1. **Predictive Modeling**:

# Perform seasonal decomposition on GDP

decomposition = seasonal\_decompose(data['GDP'], model='multiplicative', period=12) decomposition.plot() plt.show()

|  |
| --- |
| # Fit an ARIMA model to the GDP data model = ARIMA(data['GDP'], order=(5, 1, 0)) model\_fit = model.fit(disp=0) print(model\_fit.summary())  # Make predictions  predictions = model\_fit.forecast(steps=12)[0]  # Plot the predictions plt.figure(figsize=(10, 6))  plt.plot(data['GDP'], label='Actual GDP')  plt.plot(pd.date\_range(start=data.index[-1], periods=12, freq='M'), predictions, label='Predicted GDP', color='red') plt.title('GDP Forecast') plt.legend() plt.show()  # Evaluate the model  mse = mean\_squared\_error(data['GDP'][-12:], predictions) print(f'Mean Squared Error: {mse}') |

1. **Reporting**:

# Generate a summary report

|  |
| --- |
| report = f"""  Economic Data Analysis Report  ============================  1. Data Overview  ----------------   * Time Frame: {data.index.min()} to {data.index.max()} * Indicators: GDP, Unemployment Rate, Inflation Rate, Interest Rate   2. Exploratory Data Analysis  ----------------------------   * GDP, Unemployment Rate, Inflation Rate, and Interest Rate trends were plotted over time. * Correlation analysis revealed the following relationships: * GDP and Unemployment Rate: {correlation\_matrix.loc['GDP',   'Unemployment\_Rate']:.2f}   * GDP and Inflation Rate: {correlation\_matrix.loc['GDP', 'Inflation\_Rate']:.2f} * GDP and Interest Rate: {correlation\_matrix.loc['GDP', 'Interest\_Rate']:.2f}   3. Statistical Analysis  -----------------------   * Seasonal decomposition of GDP showed clear seasonal patterns. * Correlation analysis showed strong relationships between the indicators.   4. Predictive Modeling  ----------------------   * An ARIMA model was used to forecast GDP for the next 12 months. * The model's Mean Squared Error (MSE) was: {mse:.2f} |

5. Conclusions

--------------

* The analysis provided insights into the trends and relationships between key economic indicators.
* The predictive model can be used to forecast future GDP trends, aiding in economic planning and decision-making.

"""

print(report)

**Conclusion**

This project provides a comprehensive analysis of economic data, including data collection, preparation, exploratory analysis, statistical analysis, and predictive modeling. The resulting report summarizes key findings and insights, which can be useful for economic planning and decision-making.

**Example: You can get the basic idea how you can create a project from here**

**Sample code with output**

|  |
| --- |
| *# This Python 3 environment comes with many helpful analytics libraries installed*  *# It is defined by the kaggle/python Docker image:*  *https://github.com/kaggle/docker-python*  *# For example, here's several helpful packages to load*  import numpy as np *# linear algebra* import pandas as pd *# data processing, CSV file I/O (e.g. pd.read\_csv)*  *# Input data files are available in the read-only "../input/" directory*  *# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory*  import os for dirname, \_, filenames **in** os.walk('/kaggle/input'):  for filename **in** filenames:  print(os.path.join(dirname, filename))  *# You can write up to 20GB to the current directory*  *(/kaggle/working/) that gets preserved as output when you create* |

|  |
| --- |
| *a version using "Save & Run All"*  *# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session*  /kaggle/input/sales-data-for-economic-data-analysis/salesforcou rse-4fe2kehu.csv  /kaggle/input/sales-data-for-economic-data-analysis/salesforcou rse-4fe2kehu.xlsx  Overall, the dataset appears to provide a comprehensive view of sales transactions, with the potential for analysis at multiple levels, including by product, customer, and location.  Year: This column represents the year in which the transaction occurred. It could be used to track trends over time or to filter the data based on a specific year or range of years.  Month: This column represents the month in which the transaction occurred. It could be used to track trends over time or to filter the data based on a specific month or range of months.  Customer Age: This column represents the age of the customer. It could be used to segment customers based on age ranges or to analyze the purchasing behavior of different age groups.  Customer Gender: This column represents the gender of the customer. It could be used to segment customers based on gender or to analyze the purchasing behavior |

|  |
| --- |
| of different genders.  Country: This column represents the country where the transaction occurred. It could be used to analyze sales by country or to filter the data based on a specific country or range of countries.  State: This column represents the state where the transaction occurred. It could be used to analyze sales by the state or to filter the data based on a specific state or range of states.  Product Category: This column represents the broad category of the product sold. It could be used to analyze sales by product category or to filter the data based on a specific product category.  Sub Category: This column represents the specific subcategory of the product sold. It could be used to analyze sales by subcategory or to filter the data based on a specific subcategory.  Quantity: This column represents the quantity of the product sold. It could be used to analyze sales volume or to calculate the total revenue generated from a particular product or product category.  Unit Cost: This column represents the cost of producing or acquiring one unit of the product. It could be used to calculate profit margins or to compare the costs of different products or product categories.  Unit Price: This column represents the price at which one unit of the product was sold. It could be used to analyze pricing strategies or to compare the prices of different products or product categories.  Cost: This column represents the total cost of the products sold, which is calculated as the product of the quantity and the unit cost. It could be used to analyze the cost |

structure

of

the

business

or

to

calculate

the

profit

margin

of

each

sale.

Revenue:

This

column

represents

the

total

revenue

generated

by

the

sales,

which

is

calculated

as

the

product

of

the

quantity

and

the

unit

price.

It

could

be

used

to

analyze

the

overall

sales

performance

of

the

business

or

to

calculate

the

profit

generated

by

each

sale.

In

[2]:

import

pandas

as

pd

import

seaborn

as

sns

import

matplotlib.pyplot

as

plt

import

plotly.express

as

px

import

statsmodels.api

as

sm

In

[3]:

df

=

pd

.

read\_csv(

"/kaggle/input/sales-data-for-economic-data-analysi

s/salesforcourse-4fe2kehu.csv"

)

df

.

head()

Out[3]:

i

n

d

Dat

e

Y

e

Mo

nth

Cu

sto

me

Cus

tom

er

Co

unt

Stat

e

Prod

uct

Cate

Su

b

Cat

Q

ua

nti

U

ni

t

Unit

Pric

C

o

Re

ve

nu

Co

lu

mn

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | | | | | | |
|  |  | e x |  | ar |  | r  Ag e | Ge nde  r | ry |  | gory | ego  ry | ty | C o  st | e | st | e | 1 |
| 0 | 0 | 2/1  9/2  01  6 | 2 0 1  6.  0 | Fe br ua  ry | 29.  0 | F | Un ite d St ate s | Wa  shin gto n | Acc esso  ries | Tire s and Tub es | 1.  0 | 8  0.  0 0 | 109  .00  000  0 | 8  0  .  0 | 10 9.0 | Na  N |
| 1 | 1 | 2/2  0/2  01  6 | 2 0 1  6.  0 | Fe  br ua  ry | 29.  0 | F | Un  ite  d St ate s | Wa  shin gto n | Clot hing | Glo ves | 2.  0 | 2  4.  5 0 | 28.  500 000 | 4  9  .  0 | 57.  0 | Na  N |
| 2 | 2 | 2/2  7/2  01  6 | 2 0 1  6.  0 | Fe  br ua  ry | 29.  0 | F | Un  ite  d St ate s | Wa  shin gto n | Acc esso  ries | Tire s and Tub es | 3.  0 | 3.  6 7 | 5.0  000  00 | 1  1  .  0 | 15.  0 | Na  N |
|  | | | | | | | | | | | | | | | | |

In [4]:

3

3

1

/

3

2

2

/

01

6

2

0

1

6.

0

Ma

rch

29.

0

F

Un

ite

d

St

ate

s

Wa

shin

gto

n

Acc

esso

ries

Tire

s

and

Tub

es

2.

0

8

7.

5

0

116.

500

000

1

7

5

.

0

23

3.0

Na

N

4

4

3

1

/

2

/

2

01

6

2

0

1

6.

0

Ma

rch

29.

0

F

Un

ite

d

St

ate

s

Wa

shin

gto

n

Acc

esso

ries

Tire

s

and

Tub

es

3.

0

3

5.

0

0

41.

666

667

1

0

5

.

0

12

5.0

Na

N

*##Identifies Non-Null and data types in datasets* df.info()

<class 'pandas.core.frame.DataFrame'> RangeIndex: 34867 entries, 0 to 34866 Data columns (total 16 columns):

# Column Non-Null Count Dtype

--- ------ -------------- -----

0 index 34867 non-null int64

|  |
| --- |
| 1. Date 34866 non-null object 2. Year 34866 non-null float64 3. Month 34866 non-null object 4. Customer Age 34866 non-null float64 5. Customer Gender 34866 non-null object 6. Country 34866 non-null object 7. State 34866 non-null object 8. Product Category 34866 non-null object 9. Sub Category 34866 non-null object 10. Quantity 34866 non-null float64 11. Unit Cost 34866 non-null float64 12. Unit Price 34866 non-null float64 13. Cost 34866 non-null float64 14. Revenue 34867 non-null float64 15 Column1 2574 non-null float64   dtypes: float64(8), int64(1), object(7) memory usage: 4.3+ MB  In [5]: df.tail()  Out[5]: |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  | in  d e x | Dat e | Y e  ar | Mo nth | Cu sto me  r  Ag e | Cu sto me  r  Ge nd er | C  ou ntr y | S  ta te | Pr od  uct  Ca teg ory | Su b Ca teg ory | Q  ua  nti ty | U ni  t  C  o  st | Unit Pric e | C  o  st | Rev enu e | Co  lu  mn  1 | | 3 4 8 6 2 | 3 4 8 6 2 | 2/7  /20  16 | 2 0 1  6.  0 | Fe  br ua  ry | 38.  0 | M | Fr an ce | H a  ut s d e S ei  n e | Bik es | Mo unt ain Bik es | 2.  0 | 1 1 6  0.  0 | 985.  500 000 | 2 3 2  0.  0 | 197  1.00  000  0 | Na  N | | 3 4 8 6 3 | 3 4 8 6 3 | 3/1  3/2  01  5 | 2 0 1  5.  0 | Ma rch | 38.  0 | M | Fr an ce | H  a  ut s d e | Bik es | Mo unt ain Bik es | 1.  0 | 2 0 4  9.  0 | 158  3.00  000  0 | 2 0 4  9.  0 | 158  3.00  000  0 | Na  N | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | |  |  |  |  |  |  |  |  | S ei  n e |  |  |  |  |  |  |  |  | | 3 4 8 6 4 | 3 4 8 6 4 | 4/5  /20  15 | 2 0 1  5.  0 | Ap  ril | 38.  0 | M | Fr an ce | H a  ut s d e S ei  n e | Bik es | Mo unt ain Bik es | 3.  0 | 6  8  3.  0 | 560.  666 667 | 2 0 4  9.  0 | 168  2.00 000  0 | Na  N | | 3 4 8 6 5 | 3 4 8 6 5 | 8/3  0/2  01  5 | 2 0 1  5.  0 | Au gu  st | 38.  0 | M | Fr an ce | H  a  ut s d e S ei  n e | Bik es | Mo unt ain Bik es | 1.  0 | 2 3 2  0.  0 | 156  8.00  000  0 | 2 3 2  0.  0 | 156  8.00  000  0 | Na  N | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | | 3  4 8 6 6 | 3  4 8 6 6 | Na  N | N  a  N | Na  N | Na  N | Na  N | N  a  N | N  a  N | Na  N | Na  N | Na  N | N  a  N | NaN | N  a  N | 641.  532 095 | Na  N |   In [6]:  *#Removes data in row 34866* df = df.drop(34866, axis=0)  In [7]:  *#Delete data in column "Column1"* df = df.drop('Column1', axis=1)  In [8]:  *# Convert "Customer Age, Year,and Quantity " column to integer format* df['Customer Age'] = df['Customer Age'].astype(int) df['Year'] = df['Year'].astype(int) df['Quantity'] = df['Quantity'].astype(int) |

In

[9]:

*#*

*Convert*

*"Date"*

*column*

*to*

*datetime*

*format*

df[

'Date'

]

=

pd

.

to\_datetime(df[

'Date'

])

In

[10]:

*#Extract*

*the*

*year*

*and*

*month*

*from*

*the*

*"Date"*

*column*

df[

'Year'

]

=

df[

'Date'

]

.

dt

.

year

df[

'Year\_Month'

]

=

df[

'Date'

]

.

dt

.

strftime(

'%Y-%m'

)

In

[11]:

*#*

*Calculate*

*profit*

*for*

*each*

*product*

df[

'profit'

]

=

df[

'Revenue'

]

-

df[

'Cost'

]

*#*

*Calculate*

*profit*

*margin*

*for*

*each*

*product*

df[

'profit\_margin'

]

=

df[

'profit'

]

/

df[

'Revenue'

]

In

[12]:

df

.

describe()

Out[12]:

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | |
|  |  | inde x | Date | Year | Cust omer Age | Qua  ntity | Unit  Cost | Unit  Price | Cost | Reve nue | profit | profit  \_mar  gin |
| c o u n  t | 3486  6.00  0000 | 34866 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 | 3486  6.00  0000 |
| m  e a n | 1743  2.50  0000 | 2016-0  1-19  18:35:0  5.1109  96224 | 2015  .569  237 | 36.3  8289  5 | 2.00 2524 | 349.  8805  67 | 389.  2324  85 | 576.  0045  32 | 640.  8700  74 | 64.8  6554  2 | 0.13 4077 |
| m  i  n | 0.00 0000 | 2015-0  1-01  00:00:0  0 | 2015  .000  000 | 17.0  0000  0 | 1.00 0000 | 0.67 0000 | 0.66 6667 | 2.00 0000 | 2.00 0000 | -937.  0000  00 | -0.68  6747 |
| 2  5 | 8716  .250 | 2015-1  0-26 | 2015  .000 | 28.0 0000 | 1.00 | 45.0 0000 | 53.6 6666 | 85.0 0000 | 102. 0000 | 5.00 | 0.06 |
|  | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | | | | | |
|  | % | 000 | 00:00:0  0 | 000 | 0 | 0000 | 0 | 7 | 0 | 00 | 0000 | 1679 |
| 5  0  % | 1743  2.50  0000 | 2016-0  1-28  00:00:0  0 | 2016  .000  000 | 35.0  0000  0 | 2.00 0000 | 150.  0000  00 | 179.  0000  00 | 261.  0000  00 | 319.  0000  00 | 27.0  0000  0 | 0.14 7963 |
| 7  5  % | 2614  8.75  0000 | 2016-0  4-26  00:00:0  0 | 2016  .000  000 | 44.0  0000  0 | 3.00 0000 | 455.  0000  00 | 521.  0000  00 | 769.  0000  00 | 902.  0000  00 | 96.0  0000  0 | 0.22 5677 |
| m  a x | 3486  5.00  0000 | 2016-0  7-31  00:00:0  0 | 2016  .000  000 | 87.0  0000  0 | 3.00 0000 | 3240  .000  000 | 5082  .000  000 | 3600  .000  000 | 5082  .000  000 | 1842  .000  000 | 0.50 0000 |
| s  t  d | 1006  5.09  1579 | NaN | 0.49 5190 | 11.11  2902 | 0.81 3936 | 490.  0158  46 | 525.  3190  91 | 690.  5003  95 | 736.  6505  97 | 152.  8799  08 | 0.13 5445 |
| In [13]: | | | | | | | | | | | |

*#*

*Group*

*data*

*by*

*Revenue*

*and*

*month*

Month\_Revenue

=

df

.

groupby([

'Year\_Month'

])[

'Revenue'

]

.

sum()

.

reset\_index()

sns

.

relplot(data

=

Month\_Revenue,

x

=

"Year\_Month"

,

y

=

"Revenue"

,

kind

=

"line"

,

height

=

10

,

aspect

=

2.1

)

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118

:

UserWarning:

The

figure

layout

has

changed

to

tight

self.\_figure.tight\_layout(\*args,

\*\*kwargs)

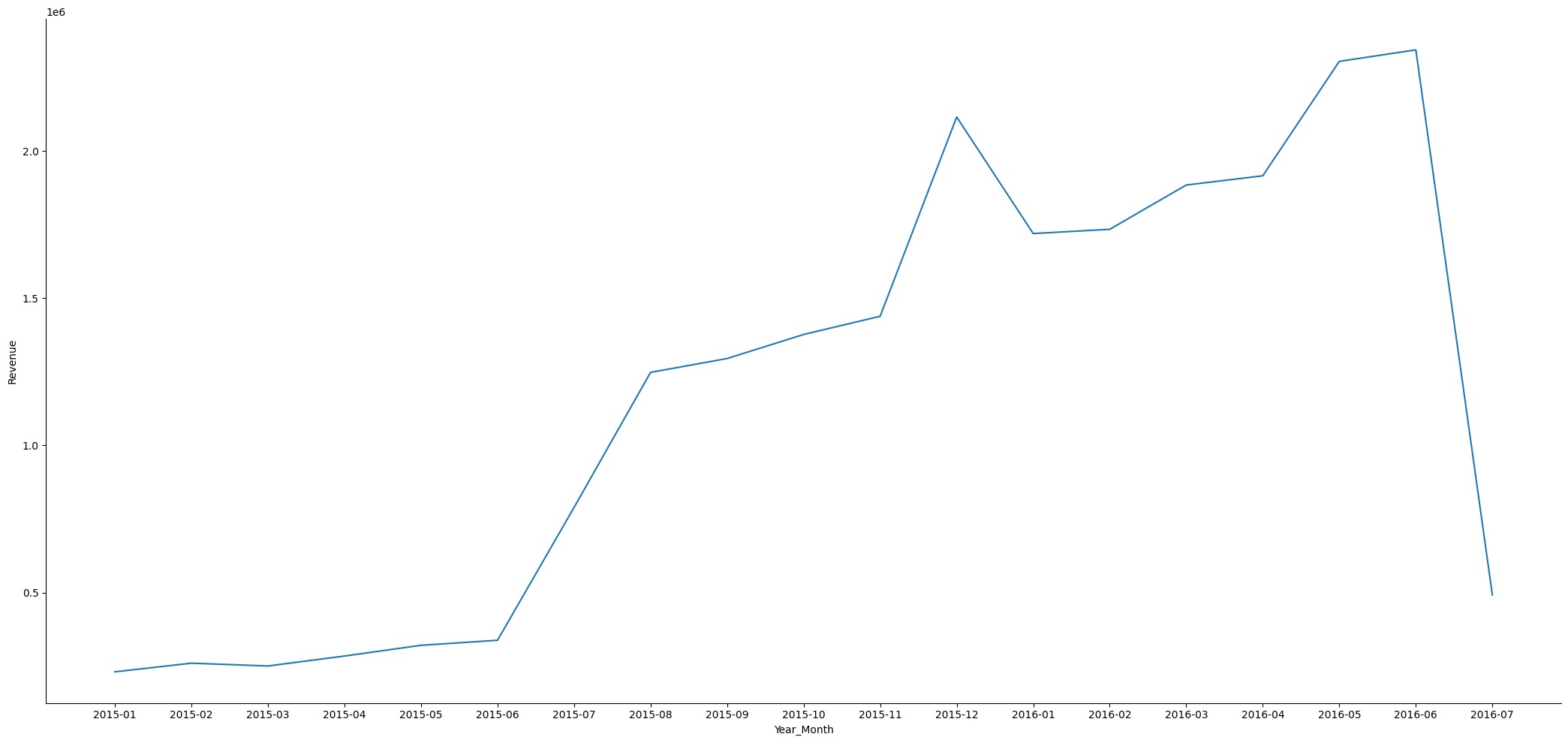
Out[13]:

<

seaborn.axisgrid.FacetGrid

at

0x7a8224913490>



|  |
| --- |
| **Revenue from sales tended to increase from month to month in 2015 and early 2016, then decreased drastically in July 2016. December 2015 was the month with the highest sales revenue, while July 2016 was the month with the lowest sales revenue**  **Then let's look at the costs incurred each month**  In [14]:  monthly\_cost =  df.groupby(['Year\_Month'])['Cost'].sum().reset\_index()  In [15]:  sns.relplot(data=monthly\_cost, x="Year\_Month", y="Cost", kind="line", height =10, aspect = 2.1)  /opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118  : UserWarning: The figure layout has changed to tight self.\_figure.tight\_layout(\*args, \*\*kwargs)  Out[15]:  <seaborn.axisgrid.FacetGrid at 0x7a8205826a10> |

**From**

**this**

**graph**

**it**

**can**

**be**

**seen**

**that**

**the**

**trend**

**of**

**Cost**

**and**

**Revenue**

**tends**

**to**

**be**

**the**

**same,**

**namely**

**increasing**

**every**

**month,**

**but**

**Cost**

**from**

**January**

**2015**

**to**

**June**

**2015**

**tends**

**to**

**be**

**higher**

**than**

**Revenue**

In

[16]:

grouped

=

df

.

groupby([

"Year\_Month"

])[[

'Cost'

,

'Revenue'

,

'profit'

]]

.

sum()

.

reset\_index()

grouped

Out[16]:

Year\_M

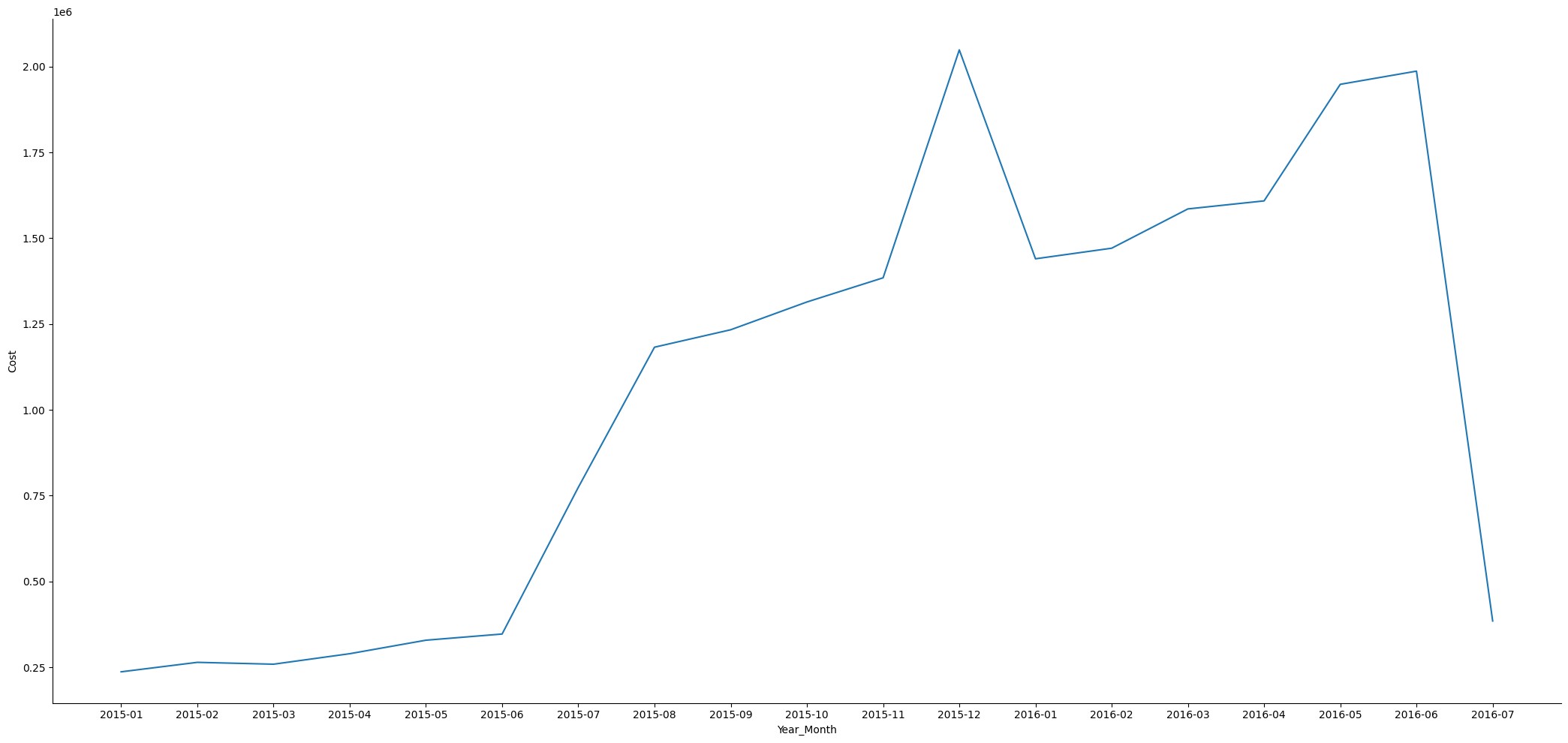
onth

Cost

Reven

ue

profit



|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 0 | 2015-0  1 | 23632  8.0 | 23054  9.0 | -5779  .0 | | 1 | 2015-0  2 | 26393  7.0 | 25985  7.0 | -4080  .0 | | 2 | 2015-0  3 | 25852  2.0 | 25035  8.0 | -8164  .0 | | 3 | 2015-0  4 | 28908  9.0 | 28414  3.0 | -4946  .0 | | 4 | 2015-0  5 | 32843  1.0 | 32062  9.0 | -7802  .0 | | 5 | 2015-0  6 | 34644  7.0 | 33775  6.0 | -8691  .0 | | 6 | 2015-0  7 | 77395  0.0 | 78905  4.0 | 1510  4.0 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 7 | 2015-0  8 | 11822  59.0 | 12481  85.0 | 6592  6.0 | | 8 | 2015-0  9 | 12330  74.0 | 12952  46.0 | 6217  2.0 | | 9 | 2015-1  0 | 13140  18.0 | 13769  69.0 | 6295  1.0 | | 1  0 | 2015-1  1 | 13844  47.0 | 14389  28.0 | 5448  1.0 | | 1  1 | 2015-1  2 | 20486  49.0 | 21160  97.0 | 6744  8.0 | | 1  2 | 2016-0  1 | 14398  68.0 | 17200  72.0 | 2802  04.0 | | 1  3 | 2016-0  2 | 14707  36.0 | 17343  76.0 | 2636  40.0 | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | 1  4 | 2016-0  3 | 15852  01.0 | 18849  78.0 | 2997  77.0 | | 1  5 | 2016-0  4 | 16086  01.0 | 19163  47.0 | 3077  46.0 | | 1  6 | 2016-0  5 | 19482  77.0 | 23051  91.0 | 3569  14.0 | | 1  7 | 2016-0  6 | 19866  80.0 | 23442  29.0 | 3575  49.0 | | 1  8 | 2016-0  7 | 38446  0.0 | 49161  2.0 | 1071  52.0 |   In [17]: fig = px.line(grouped, x='Year\_Month', y=['Cost', 'Revenue',  'profit'], title='Monthly Performance')  fig.update\_xaxes(title='Year-Month') fig.update\_yaxes(title='Amount ($)') |

|  |
| --- |
| *# menampilkan plot* fig.show()  Jan 2015Mar 2015May 2015Jul 2015Sep 2015Nov 2015Jan 2016Mar  2016May 2016Jul 201600.5M1M1.5M2M variableCostRevenueprofitMonthly PerformanceYear-MonthAmount ($)  **There is a clear seasonal pattern in this data, where profits tend to increase at the end of the year and decrease at the beginning of the next.**  In [18]:  *# Group by sub category and calculate total quantity sold* category\_sales = df.groupby('Sub  Category')['Quantity'].sum().reset\_index()  In [19]: fig = px.bar(category\_sales, y='Sub Category', x='Quantity', text\_auto='.2s', title="Product Sales Quantity Based on Sub  Categories") fig.show()  20030011k3.0k1.1k1.5k9108.4k7904.0k5.5k6.1k1.1k75022k2.7k64005k1 |

|  |
| --- |
| 0k15k20kBike RacksBike StandsBottles and  CagesCapsCleanersFendersGlovesHelmetsHydration PacksJerseysMountain  BikesRoad BikesShortsSocksTires and TubesTouring BikesVests  Product Sales Quantity Based on Sub CategoriesQuantitySub Category  In [20]:  category\_profit= df.groupby('Sub  Category')['profit'].sum().reset\_index()  In [21]:  fig = px.bar(category\_profit, y='Sub Category', x='profit', text\_auto='.2s', title="Profit by Sub Category")  fig.update\_xaxes(title='Profit($)') fig.show()  35k25k130k44k15k71k46k520k72k300k140k98k87k9.5k510k95k58k0100 k200k300k400k500kBike RacksBike StandsBottles and  CagesCapsCleanersFendersGlovesHelmetsHydration PacksJerseysMountain  BikesRoad BikesShortsSocksTires and TubesTouring BikesVests  Profit by Sub CategoryProfit($)Sub Category  In [22]:  category\_margin = df.groupby('Sub |

|  |
| --- |
| Category')['profit\_margin'].mean().reset\_index()  In [23]:  fig = px.bar(category\_margin, y='Sub Category', x='profit\_margin', title="Profit Margin by Sub Category")  fig.show()  00.050.10.150.2Bike RacksBike StandsBottles and  CagesCapsCleanersFendersGlovesHelmetsHydration PacksJerseysMountain  BikesRoad BikesShortsSocksTires and TubesTouring BikesVests  Profit Margin by Sub Categoryprofit\_marginSub Category  **Bike Racks had the highest profit margin of 22.7%, followed by Fenders with a profit margin of 20.7%. Meanwhile, Road Bikes and Mountain Bikes have very low profit margins, respectively 0.5% and 1.0%.**  In [24]: fig = px.histogram(df, x="Customer Age") fig.show()  20304050607080020040060080010001200  Customer Agecount |

|  |
| --- |
| **What products are purchased the most based on the age of the customer?**  In [25]:  *# Group by Customer Age and product category, sum quantity sold* df\_grouped = df.groupby(["Customer Age", "Product Category"])["Quantity"].sum().reset\_index()  *# Find top selling product for each Customer Age* top\_products = df\_grouped.groupby("Customer Age").apply(lambda x: x.loc[x.Quantity.idxmax()])  *# Create bar chart* fig = px.bar(top\_products, x="Customer Age", y="Quantity", color="Product Category", title="Top Selling Products by Customer Age") fig.show()  2030405060708002004006008001000120014001600  Product CategoryAccessoriesClothingTop Selling Products by Customer  AgeCustomer AgeQuantity  In [26]:  *# Group by Customer Age and product category, sum quantity sold* df\_grouped = df.groupby(["Customer Age", "Sub |

|  |
| --- |
| Category"])["Quantity"].sum().reset\_index()  *# Find top selling product for each Customer Age* top\_products = df\_grouped.groupby("Customer Age").apply(lambda x: x.loc[x.Quantity.idxmax()])  *# Create bar chart* fig = px.bar(top\_products, x="Customer Age", y="Quantity", color="Sub Category", title="Top Selling Sub Category Products by Customer Age") fig.show()  203040506070800100200300400500600700800  Sub CategoryTires and TubesHelmetsHydration PacksBottles and  CagesShortsTop Selling Sub Category Products by Customer AgeCustomer AgeQuantity  **Which country has the highest profit?**  In [27]:  country\_sales =  df.groupby('Country')['profit'].sum().reset\_index()  In [28]: |

|  |
| --- |
| fig = px.pie(df, values='profit', names='Country', color\_discrete\_sequence=px.colors.sequential.RdBu) fig.show()  42.4%31%14.5%12.1%  GermanyUnited StatesUnited KingdomFrance  **\*\*What products are purchased the most in each country** \*\*  In [29]:  *# Group by country and sub category, sum quantity sold* df\_grouped = df.groupby(["Country", "Sub Category"])["Quantity"].sum().reset\_index()  *# Find top selling product for each country* top\_products = df\_grouped.groupby("Country").apply(lambda x: x.loc[x.Quantity.idxmax()])  *# Create bar chart* fig = px.bar(top\_products, x="Country", y="Quantity", color="Sub Category", title="Top Selling Sub Category Products by Country") fig.show() |

FranceGermanyUnited KingdomUnited States02k4k6k8k10k12k

Sub CategoryTires and TubesTop Selling Sub Category Products by

CountryCountryQuantity

**"Tires and Tubes" are the most frequently purchased products in all countries in the dataset. However, what are the products with the highest profit margins in each country?**

In [30]:

[**Reference**](https://github.com/riyouuyt/Data-Analyst-Web-Scraping-Wikipedia-for-Economic-Analysis-/blob/master/Data_analyst_Web_Scrapping_Wikipedia_for_Economic_Analyst_in_Southeast_Asian.ipynb)[**link**](https://github.com/riyouuyt/Data-Analyst-Web-Scraping-Wikipedia-for-Economic-Analysis-/blob/master/Data_analyst_Web_Scrapping_Wikipedia_for_Economic_Analyst_in_Southeast_Asian.ipynb)