College Code-5113

Data Analytics with Cognos

Product Sales Analysis

E. Chanikya-(511321104013)-chanikyaeddula@gmail.com

E. Manoj- (511321104053)-manojerragopula@gmail.com

R.Bharath kumar-(511321104010)-bk7428891@gmail.com

V.Mohith Manindranath-(511321104055)-
v.mohitmanindranath162003@gmail.com

PHASE 3: Development part 1

Introduction:

The aim of this project is to demonstrate the data analysis skills I've learned thus far and to apply them to real-world scenarios. As such, this project asks and answers real-world questions about real-world sales data. For instance, "What was the best month for sales?", or, "Which time of the day should we display advertisements to maximize the likelihood of customers' purchasing products?"

Dataset:

The dataset is comprised of hundreds of thousands of electronics store purchases broken down by product type, prices, order date, purchase address, etc., corresponding to the following coloumns:

Column	Description
Order ID	Unique IDs that are used to track orders.
Product	Names of the products.
Quantity Ordered	Total quantity ordered of a particular item.
Price Each	Prices of the products ordered.
Order Date	Dates and time at which a customer made an order.
Purchase Address	Addresses the orders were delivered to.

https://www.kaggle.com/datasets/beekiran/sales-data-analysis

Project:

The study will make use of finding best month for sales, how much earned on that month. This complete evaluation will provide a clear picture of the analysis of the sales based on order ID, Quality ordered, price each, order date, purchase address, Month, Sales, City, Hours which involves in the best sales month, forecasting profit, and the demand of the product.

Importing the required libraries:

In this step we are going to import the required python libraries and modules to work with our data and perform various data processing and machine learning tasks.

```
import pandas as pd

import pathlib

import numpy as np

import matplotlib as mpl

import matplotlib.pyplot as plt

import warnings

warnings.simplefilter("ignore")
```

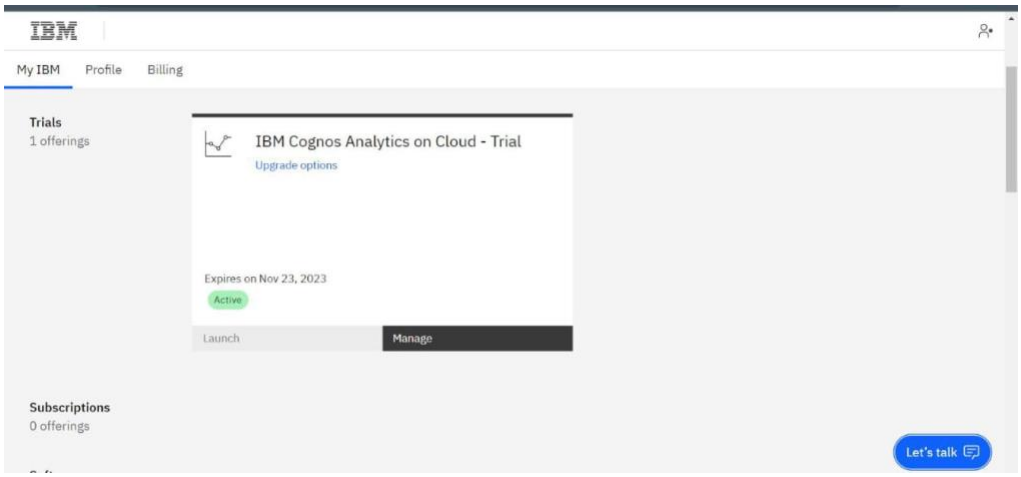
Loading the dataset:

This step involves loading our dataset into memory. We use libraries like pandas to read data from a CSV file or other formats.

```
all_data = pd.read_csv("/kaggle/input/sales-data-analysis/Sales Data.csv")
```

Preprocessing the data:

Preprocessing data in air quality analysis is a crucial step to ensure that the data is clean, reliable, and ready for in-depth analysis.



Data Cleaning:

Missing data handling: Identify and address missing data points, which can result from sensor malfunctions or communication issues. Options include imputing missing values or removing affected data points if necessary.

Outlier Detection: Detect and handle outliers, which can skew the analysis. Outliers may result from equipment malfunction or unusual events. You can choose to filter out extreme values or apply statistical techniques like Z-score analysis to identify them.

This step is vital for accurate comparisons and correlations between different datasets.

Data Transformation:

Feature Scaling: Normalize or standardize numerical features to bring them to a similar scale. This is important for algorithms sensitive to feature scales.

Feature Encoding: Convert categorical variables into a numerical format using techniques like one-hot encoding or label encoding.

Feature Engineering: Create new features or modify existing ones to capture relevant information and patterns in the data.

Binning: Group continuous data into bins or categories to simplify analysis.

Log Transformation: Apply logarithmic transformations to features when necessary to make their distribution more normal.

Data Reduction:

Dimensionality Reduction: Reduce the number of features, often using techniques like Principal Component Analysis (PCA) or feature selection to select the most relevant variables.

Outlier Detection and Handling: Identify and deal with outliers, which can distort analysis and modeling results.

Data Integration:

Merge data from multiple sources or datasets to create a consolidated dataset for analysis.

Exploratory Data Analysis:

It focuses on Exploratory Data Analysis (EDA). It involves exploring and visualizing the data to gain insights. In this example, a simple time series plot is created using Matplotlib to visualize the sales.

Question 1: What was the best month for sales? How much was earned that month?

To answer this question, first, we need to extract only the months from the 'Order Date' coloumn and store each separately in a new coloumn, 'Months'. Second, we need to get the total sales amounts per order by multiplying the quantity ordered with the price of each individual product, and creating and storing the results in a 'Sales' coloumn. Finally, I will group the data by month, calculate the total sum of sales per month, and, lastly, visualize the data to get a better view of how sales changed from one month to the next.

```
Months_col = pd.to_datetime(df['Order Date']).dt.month_name().str[:3]

df.insert(loc=5, column='Months', value=Months_col)

Sales_col = df['Quantity Ordered'] * df['Price Each']
```

```
df.insert(loc=4, column='Sales', value=Sales_col)

sales_per_month = df.groupby(['Months']).sum()[['Quantity Ordered', 'Sales']]

sort_order = ['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov',
'Dec']

sales_per_month = sales_per_month.reindex(sort_order)

sales_per_month_USD = sales_per_month.copy()

sales_per_month_USD['Sales'] = sales_per_month_USD['Sales'].apply(lambda sale:
'${:,.2f}'.format(sale))

print('The following table displays the total sales amount (and quantities ordered)
for each month:')

sales_per_month_USDsales_per_month_sorted=sales_per_month.sort_values(by=
'Sales', ascending=False)

best_month = sales_per_month_sorted.index[0]

best_month = pd.to_datetime(best_month, format='%b').month_name()

print('The best month for sales was:', best_month)

maxsale = sales_per_month_sorted['Sales'].iloc[0]

print('The total sales amount earned that month was: ${:,.2f}'.format(maxsale))

months = sales_per_month.index.values

sales = sales_per_month['Sales']

plt.figure(figsize=(12,7))

plt.bar(months, sales,color='#407bbf', linewidth=1,edgecolor='k')

plt.title('Sales Amount Per Month', fontsize=1

plt.xlabel('Month', fontsize=13)

plt.ylabel('Sales Amount in USD ($)', fontsize=13)

plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)
```

```
plt.gcf().axes[0].yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('${x:,.0f}'))

plt.tight_layout()

plt.show()
```

Question 2: Which city sold the most products?

To compare cities, first we'll have to extract the city corresponding to each order from the 'Purchase Address' coloumn and store them in a separate coloumn 'City'. Thereafter we can group the data by city and calculate the total sum of sales for each city separately.

```
def get_city_state(address):
    address = address.split(', ')
    city = address[1]
    state = address[2][0:2]
    return "{} ({})".format(city, state)

city_col = df['Purchase Address'].apply(lambda add: get_city_state(add))

df.insert(loc=8, column='City', value=city_col)

sales_per_city = df.groupby(['City']).sum()['Sales']

sales_per_city_USD = sales_per_city.apply(lambda sale: '${:,.2f}'.format(sale)).to_frame(name='Total Sales Amount')

print('The following table displays the total sales amount for each city:')

sales_per_city_USD

sales_per_city_sorted = sales_per_city.sort_values(ascending=False)

best_city = sales_per_city_sorted.index[0]

print('The city that sold the most products is:', best_city)

cities = sales_per_city.index.values

plt.figure(figsize=(10,7))
```

```
plt.bar(cities,sales_per_city,color='#44749d',width=0.6,linewidth=1,edgecolor='k')

plt.title('Sales Amount Per City', fontsize=15)

plt.xlabel('City', fontsize=13)

plt.ylabel('Sales Amount in USD ($)', fontsize=13)

plt.xticks(rotation=60)

plt.gcf().axes[0].yaxis.get_major_formatter().set_scientific(False)

plt.gcf().axes[0].yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('${x:,.0f}'))

plt.tight_layout()

plt.show()
```

Question 3: Which product sold the most? And why do you think it sold the most?

To answer this question, we would have to group the data based on product purchases and then calculate the total amount of quantities ordered for each product to determine which one sold the most amount of quantities.

```
products_sold = df.groupby(['Product']).sum()['Quantity Ordered']

products_sold = products_sold.sort_values(ascending=False)

most_sold_pro`duct = products_sold.index[0]

print('The product that was sold the most is:', most_sold_product)
```

Question 4: Is there a relationship between how much a product costs and the quantity sold?

One way to answer this question is to create a dual-axis line chart displaying the prices of each product and the quantity sold in order to compare them.
First, we will have to create two groups, the first representing the prices of each product, the second representing the total quantity sold for each product.

```
products_quantity = df.groupby(['Product']).sum()['Quantity Ordered']

products = products_quantity.index.values

products_prices = df.groupby(['Product'])['Price Each'].apply(lambda price:
float(np.unique(price)))

fig, ax1 = plt.subplots(figsize=(12,7))

ax1.plot(products,products_quantity,marker='o',c='#407bbf',lw=2,

label='Quantities')

ax2 = ax1.twinx()

ax2.plot(products, products_prices,marker='o',c='#bf4040',lw=2,label='Prices')

ax1.set_title('The Relationship Between Product Price and Quantity
Sold',fontsize=15)

ax1.set_xlabel('Product Name', fontsize=13)

ax1.set_ylabel('Total Quantity Sold', fontsize=12, color='#407bbf')

ax2.set_ylabel('Prices in USD ($)', fontsize=12, color='#cc3333')

ax1.set_xticklabels(products, rotation='vertical')

ax1.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))

ax2.yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('${x:,.0f}'))

ax1.legend(loc='upper left')

x2.legend(loc='upper right')

plt.grid()

plt.show()
```

Question 5: Which products are most often sold together?

For starters, we can filter data based on whether there are duplicates in the 'Order ID' coloumn, indicating that the same person made multiple product purchases, and then join the multiple products sold together and count the instances of

particular products being sold together in order to extract those that most often ordered together.

```
order_filter = df['Order ID'].duplicated(keep=False)

df_multiple_orders = df[order_filter][['Order ID', 'Product']]

orders_per_person = df_multiple_orders.groupby(['Order ID'])['Product'].transform(lambda product: ", ".join(product))

df_orders_per_person = orders_per_person.to_frame(name='Products Sold Together').reset_index(drop=True)

orders_frequency = orders_per_person.value_counts()

df_orders_frequency = orders_frequency.to_frame(name='Frequency of products sold together')

df_orders_frequency

most_sold_together = orders_frequency.index[0]

print('The two products sold together the most often are: {}'.format(' and '.join(most_sold_together.split(', '))))
```

Question 6: Which time of the day should we display advertisments to maximize the likelihood of customer's purchasing products?

One way to answer this question is to extract the time of the day from the 'Order Date' coloumn, and then grouping the data based on the time of the day (hour) in which a product was purchased to determine which times are associated with the most product purchases.

```
Time_col = pd.to_datetime(df['Order Date'], format='%d/%m/%y %H:%M').dt.strftime('%I %p')

df.insert(loc=6, column='Time of Purchase', value=Time_col)

purchases_per_hour = df.groupby(['Time of Purchase']).sum()['Quantity Ordered']
```

```
time_sort_order = ['12 AM', '01 AM', '02 AM', '03 AM', '04 AM', '05 AM', '06 AM', '07 AM', '08 AM', '09 AM', '10 AM', '11 AM', '12 PM', '01 PM', '02 PM', '03 PM', '04 PM', '05 PM', '06 PM', '07 PM', '08 PM', '09 PM', '10 PM', '11 PM']

purchases_per_hour = purchases_per_hour.reindex(time_sort_order)

df_purchases_per_hour = purchases_per_hour.to_frame(name='Total Quantity Sold')

print('The following table displays the total sum of quantities ordered for each hour of the day:')

df_purchases_per_hour

purchases_per_hour_sorted = purchases_per_hour.sort_values(ascending=False)

best_hour = purchases_per_hour_sorted.index[0]

print('The best time of day for displaying advertisements is:', best_hour)

time_of_purchase = purchases_per_hour.index.values

plt.figure(figsize=(12,7))

plt.bar(time_of_purchase,purchases_per_hour,color='#4169e1',linewidth=1,

edgecolor='k')

plt.title('Quantities Sold Per Hour', fontsize=15)

plt.xlabel('Time of Day', fontsize=13)

plt.ylabel('Amount of Quantities Sold', fontsize=13)

plt.xticks(rotation=90)

plt.gcf().axes[0].yaxis.set_major_formatter(mpl.ticker.StrMethodFormatter('{x:,.0f}'))

plt.tight_layout()

plt.show()
```

Conclusion:

This code provides a basic structure for an air quality analysis and prediction project. It demonstrates essential data loading, preprocessing, EDA, model training, and evaluation steps. However, the success of such a project relies on comprehensive data preparation, domain knowledge, feature engineering, model selection, and careful consideration of ethical and practical aspects.