Idea: Exploratory Data Analysis (EDA) on Retail Sales Data Description: PROJECT 1 PROPOSAL LEVEL 1 In this project, you will work with a dataset containing information about retail sales. The goal is to perform exploratory data analysis (EDA) to uncover patterns, trends, and insights that can help the retail business make informed decisions. Dataset 1 Link Dataset 2 Link Key Concepts and Challenges: 1. 2. 3. 4. 5. 6. Data Loading and Cleaning: Load the retail sales dataset. Descriptive Statistics: Calculate basic statistics (mean, median, mode, standard deviation). Time Series Analysis: Analyze sales trends over time using time series techniques. Customer and Product Analysis: Analyze customer demographics and purchasing behavior. Visualization: Present insights through bar charts, line plots, and heatmaps. Recommendations: Provide actionable recommendations based on the EDA

**ChatGPT said:**

That’s a great foundational project for developing and showcasing your data analysis skills! Here's a refined version of your **Project Proposal** that gives a clear and professional overview while remaining concise and practical.

**🛍️ Project Proposal: Exploratory Data Analysis (EDA) on Retail Sales Data**

**Level**: Beginner (Level 1)  
**Objective**:  
To explore a retail sales dataset to uncover trends, patterns, and insights that will support better business decisions. This project will emphasize data wrangling, time series analysis, customer behavior profiling, and data visualization.

**📂 Datasets**

* **Dataset 1**: [Insert Link]
* **Dataset 2**: [Insert Link]

**🔍 Key Concepts and Analytical Goals**

1. **Data Loading & Cleaning**
   * Load both datasets using Python (e.g., pandas)
   * Handle missing values, duplicates, and incorrect data types
   * Combine datasets if applicable (e.g., sales + customer data)
2. **Descriptive Statistics**
   * Calculate key metrics: mean, median, mode, standard deviation
   * Summarize sales per product, customer, region, and time period
3. **Time Series Analysis**
   * Identify trends, seasonality, and fluctuations in sales
   * Use rolling averages and resampling (monthly, quarterly)
   * Highlight peak seasons and off-seasons
4. **Customer and Product Analysis**
   * Segment customers by frequency and value (RFM Analysis)
   * Identify top-performing products and underperformers
   * Explore purchasing behaviors across demographics (if available)
5. **Visualization**
   * Bar charts for product/category sales
   * Line plots for time series analysis
   * Heatmaps for correlation between variables or temporal patterns
6. **Recommendations**
   * Provide data-driven insights and suggest strategies such as:
     + Inventory optimization
     + Promotion timing
     + Customer retention strategies

**🛠️ Tools & Technologies**

* **Python** (Pandas, NumPy, Matplotlib, Seaborn, Plotly)
* **Jupyter Notebook or Google Colab**
* Optionally: Excel or Power BI for dashboarding

Would you like help creating a Jupyter Notebook template or setting up the first step (like loading and cleaning the data)?

4o

**You said:**

yes

**ChatGPT said:**

Great! Here's a **starter Jupyter Notebook template** for your **Retail Sales EDA project**. You can copy this into a .ipynb file or paste it into your notebook environment (Jupyter or Colab) to begin your analysis.

python

CopyEdit

# Retail Sales EDA Project

# 📦 1. Import Libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from datetime import datetime

# Set visualization styles

sns.set(style='whitegrid')

plt.rcParams['figure.figsize'] = (12, 6)

# 📁 2. Load Data

# Replace with your dataset paths or URLs

df1 = pd.read\_csv('path\_or\_url\_to\_dataset1.csv')

df2 = pd.read\_csv('path\_or\_url\_to\_dataset2.csv')

# 🧹 3. Initial Data Cleaning

# Display first few rows

display(df1.head())

display(df2.head())

# Check for missing values

print("Dataset 1 missing values:")

print(df1.isnull().sum())

print("\nDataset 2 missing values:")

print(df2.isnull().sum())

# Check data types

print("\nData types:")

print(df1.dtypes)

# Optional: Merge datasets if applicable

# df = pd.merge(df1, df2, on='common\_column', how='inner')

# 🧮 4. Descriptive Statistics

print("\nDescriptive Stats (Dataset 1):")

print(df1.describe())

# 📈 5. Time Series Analysis

# Convert to datetime if needed

# df1['Order Date'] = pd.to\_datetime(df1['Order Date'])

# Sales over time

# time\_series = df1.groupby(df1['Order Date'].dt.to\_period('M')).sum()['Sales']

# time\_series.plot(title='Monthly Sales Trend')

# 📊 6. Customer and Product Analysis

# Top customers

# top\_customers = df1.groupby('Customer Name')['Sales'].sum().sort\_values(ascending=False).head(10)

# top\_customers.plot(kind='bar', title='Top 10 Customers by Sales')

# Product analysis

# top\_products = df1.groupby('Product Name')['Sales'].sum().sort\_values(ascending=False).head(10)

# top\_products.plot(kind='bar', title='Top 10 Products by Sales')

# 🔥 7. Correlation and Heatmap

# corr = df1.corr(numeric\_only=True)

# sns.heatmap(corr, annot=True, cmap='coolwarm')

# ✅ 8. Recommendations

# Based on findings in your analysis, write your conclusions here

Would you like help customizing this with your actual dataset structure (column names, key fields, etc.)? If you can share the column names or a sample of the data, I can tailor it perfectly for your use case.

4o

**You said:**

how do l calculate basic statistics (mean, median, mode, standard deviation) in python

**ChatGPT said:**

To calculate **basic statistics** in Python using **pandas**, here's how you can do it step-by-step:

Assume your DataFrame is called df and you’re analyzing a numeric column like Sales.

**🔢 1. Mean**

python

CopyEdit

df['Sales'].mean()

**🔢 2. Median**

python

CopyEdit

df['Sales'].median()

**🔢 3. Mode**

python

CopyEdit

df['Sales'].mode()

⚠️ Note: mode() can return multiple values if there's a tie.

**🔢 4. Standard Deviation**

python

CopyEdit

df['Sales'].std()

**💡 Bonus: Use describe() for a quick summary**

python

CopyEdit

df['Sales'].describe()

This includes:

* Count
* Mean
* Std
* Min
* 25%, 50% (median), 75%
* Max

**📌 Example with all together:**

python

CopyEdit

column = 'Sales'

print("Mean:", df[column].mean())

print("Median:", df[column].median())

print("Mode:", df[column].mode().values)

print("Standard Deviation:", df[column].std())

Would you like a reusable function that takes any column name and prints these stats nicely formatted?

4o

**You said:**

yes

**ChatGPT said:**

Perfect! Here's a **reusable function** in Python that calculates and prints the **mean, median, mode, and standard deviation** for any numeric column in a pandas DataFrame.

**✅ Function: basic\_stats()**

python

CopyEdit

def basic\_stats(df, column):

if column not in df.columns:

print(f"❌ Column '{column}' not found in DataFrame.")

return

if not pd.api.types.is\_numeric\_dtype(df[column]):

print(f"⚠️ Column '{column}' is not numeric.")

return

print(f"📊 Basic Statistics for '{column}':")

print(f"Mean: {df[column].mean():.2f}")

print(f"Median: {df[column].median():.2f}")

print(f"Mode: {df[column].mode().values}")

print(f"Standard Deviation: {df[column].std():.2f}")

**🧪 Example Usage:**

python

CopyEdit

basic\_stats(df, 'Sales')

basic\_stats(df, 'Profit')

Let me know if you’d like a version that also plots a histogram or boxplot for visual insight along with the stats!

4o

**You said:**

I have column for quantity, price per unit and total amount which column can l calculate mean, median, mode etc on

**ChatGPT said:**

Great question! Here's how you can approach it:

**📊 Your Columns:**

| **Column** | **Description** | **Suitable for Stats?** |
| --- | --- | --- |
| Quantity | Number of units sold in a transaction | ✅ Yes |
| Price per Unit | Price for each item/unit | ✅ Yes |
| Total Amount | Quantity × Price per Unit for each transaction | ✅ Yes |

**✅ You can calculate mean, median, mode, and standard deviation on all three, depending on the insight you want:**

**1. Quantity**

* Mean: Average number of units sold per transaction
* Mode: Most frequently sold quantity (e.g., people often buy in 2s)
* Std Dev: How much quantity varies across sales

**2. Price per Unit**

* Mean: Average price per product
* Mode: Most common price (useful for price bands or promotions)
* Std Dev: Helps detect outlier prices

**3. Total Amount**

* Mean: Average transaction value
* Median: Typical spend per order
* Std Dev: Indicates high vs low-value transactions

**🧪 Suggested:**

Start with:

python

CopyEdit

basic\_stats(df, 'Quantity')

basic\_stats(df, 'Price per Unit')

basic\_stats(df, 'Total Amount')

Would you like help generating histograms or boxplots for these to visualize distribution?

4o

**You said:**

How do i define basic\_stats and make it callable

**ChatGPT said:**

Awesome! Here's a quick step-by-step to define and use the basic\_stats function in your notebook or Python script.

**✅ 1. Define the Function**

Place this block **once** at the top of your notebook or script (before calling it):

python

CopyEdit

def basic\_stats(df, column):

"""

Prints basic statistics: mean, median, mode, and standard deviation

for a given numeric column in a DataFrame.

"""

if column not in df.columns:

print(f"❌ Column '{column}' not found in DataFrame.")

return

if not pd.api.types.is\_numeric\_dtype(df[column]):

print(f"⚠️ Column '{column}' is not numeric.")

return

print(f"\n📊 Basic Statistics for '{column}':")

print(f"Mean: {df[column].mean():.2f}")

print(f"Median: {df[column].median():.2f}")

print(f"Mode: {df[column].mode().values}")

print(f"Standard Deviation: {df[column].std():.2f}")

**✅ 2. Call the Function**

After defining the function and loading your dataset into a DataFrame (df), just call:

python

CopyEdit

basic\_stats(df, 'Quantity')

basic\_stats(df, 'Price per Unit')

basic\_stats(df, 'Total Amount')

**🎯 Bonus Tip:**

You can loop through a list of columns like this:

python

CopyEdit

columns\_to\_check = ['Quantity', 'Price per Unit', 'Total Amount']

for col in columns\_to\_check:

basic\_stats(df, col)

Let me know if you want this to also include a **histogram** or **boxplot** for each column!

4o

**You said:**

what does f before basic stattistics means

**ChatGPT said:**

Great question!

The f in front of a string—like in this line:

python

CopyEdit

print(f"\n📊 Basic Statistics for '{column}':")

—means you're using a **formatted string literal**, or **f-string**.

**📘 What’s an f-string?**

An **f-string** lets you **embed variables directly inside a string** using curly braces {}. It was introduced in **Python 3.6** and makes your code cleaner and easier to read.

**✅ Example:**

python

CopyEdit

name = "Victoria"

print(f"Hello, {name}!")

🟰 This will print:

CopyEdit

Hello, Victoria!

Compare that to the old way:

python

CopyEdit

print("Hello, " + name + "!")

**🧠 In Your Function:**

python

CopyEdit

print(f"\n📊 Basic Statistics for '{column}':")

This is saying:

* Print "Basic Statistics for [whatever column name you pass in]"
* \n is just a line break to make it cleaner in the console output

Let me know if you’d like a cheat sheet on f-strings or more formatting tips!

4o

**You said:**

what does .2f means

**ChatGPT said:**

Great follow-up!

**🧮 .2f in Python means:**

**Format a number to 2 decimal places** (the f stands for “float”).

It’s part of the **format specification mini-language** used inside f-strings.

**🔍 Example:**

python

CopyEdit

num = 5.6789

print(f"{num:.2f}")

➡️ Output:

CopyEdit

5.68

It **rounds** the number to **2 digits after the decimal point**.

**🧠 Breakdown:**

* : → tells Python you're about to give formatting instructions
* .2 → keep 2 digits after the decimal
* f → floating-point number

**More Examples:**

python

CopyEdit

num = 123.4567

print(f"{num:.1f}") # 123.5

print(f"{num:.3f}") # 123.457

**In Your Case:**

When you write:

python

CopyEdit

print(f"Mean: {df[column].mean():.2f}")

It ensures the mean is displayed like 123.45 instead of 123.456789012.

Let me know if you want a quick formatter to display numbers as currency (e.g., ₦123.45)!

4o

**You said:**

i type retailsales = basic\_stats and retailsales as a dataframe has disapeared how do l undo or recall retailsales as a dataframe back

**ChatGPT said:**

Ahh, no worries—this is a common mix-up!

What likely happened is:

python

CopyEdit

retailsales = basic\_stats

You **overwrote** your retailsales DataFrame by assigning it the function basic\_stats instead 😅.

**🛠️ How to fix it:**

**✅ Option 1: Reload the Data**

If you originally loaded your data like this:

python

CopyEdit

retailsales = pd.read\_csv('your\_dataset.csv')

Just re-run that cell to load the data back.

**✅ Option 2: Undo in Jupyter Notebook**

If you're using **Jupyter Notebook or Google Colab**:

1. Press Z immediately to **undo delete cell** (if that was the issue).
2. Or click **Edit > Undo** (or use Ctrl+Z) to undo the last change.
3. If that doesn’t work, re-run the cell where you first loaded the data.

**🧠 Pro Tip:**

Avoid overwriting your data accidentally by using different names for functions and variables. For example:

* Keep retailsales for your DataFrame
* Keep basic\_stats as the function name

Let me know what platform you're using (Jupyter, Colab, VS Code), and I can guide you more specifically!

4o

**You said:**

Time Series Analysis: Analyze sales trends over time using time series techniques.

**ChatGPT said:**

Perfect! Time Series Analysis is **key for understanding trends and seasonality** in your retail data—like finding peak sales periods or slow months.

Let’s walk through how to do **Time Series Analysis** in Python step by step using your retailsales DataFrame. We'll assume you have a Date column and a Total Amount column (or equivalent).

**🔁 Step-by-Step: Time Series Analysis in Python**

**✅ 1. Ensure the Date column is in datetime format**

python

CopyEdit

retailsales['Date'] = pd.to\_datetime(retailsales['Date'])

**📅 2. Set the date as index (optional but useful)**

python

CopyEdit

retailsales.set\_index('Date', inplace=True)

**📊 3. Aggregate sales by month**

This groups all sales by month and sums them:

python

CopyEdit

monthly\_sales = retailsales['Total Amount'].resample('M').sum()

**📈 4. Plot the monthly trend**

python

CopyEdit

monthly\_sales.plot(title='Monthly Sales Trend', ylabel='Sales (Total Amount)', xlabel='Date')

plt.show()

**📉 5. Optional: Rolling average to smooth the trend**

python

CopyEdit

monthly\_sales.rolling(window=3).mean().plot(title='3-Month Rolling Average of Sales')

plt.show()

**🔍 Optional Explorations:**

* **Seasonality**: Are sales higher in certain months or quarters?
* **YoY Comparison**: Compare this year to last year
* **Weekday patterns**: Do certain days have higher sales?

**🧠 Sample Use Case**

If you want to see **which months perform best**, try:

python

CopyEdit

monthly\_sales.sort\_values(ascending=False).head(5)

Would you like me to help build a reusable time series function or add line plots for both raw + smoothed sales?

4o

**You said:**

why do you use .resample

**ChatGPT said:**

Awesome question, Victoria!

.resample() is one of **the most powerful tools** in time series analysis with pandas. Here's what it does:

**🔄 What does .resample() do?**

.resample() is used to **change the frequency** of your time series data.

Think of it like **grouping by date**, but smarter—because it works directly with time-based data.

**📅 Example Use Case:**

You have daily sales data, but you want to analyze it **by month**:

python

CopyEdit

monthly\_sales = retailsales['Total Amount'].resample('M').sum()

* 'M' = **Monthly frequency**
* .sum() = Sum all sales that occurred in each month

This gives you one total sales number **per month**, even if the original data was daily or hourly.

**⏱ Common .resample() Frequencies:**

| **Code** | **Meaning** |
| --- | --- |
| 'D' | Daily |
| 'W' | Weekly |
| 'M' | Monthly |
| 'Q' | Quarterly |
| 'Y' | Yearly |
| 'H' | Hourly |

**🧠 Why Use It?**

Because raw transaction data is often too detailed (daily or per sale), resampling helps:

* **Reveal long-term trends** (e.g., monthly or quarterly)
* **Smooth out noise** from day-to-day fluctuations
* **Make seasonal patterns easier to see**

Would you like me to show a chart comparing **daily vs monthly sales trends** so you can visually see the difference?

4o

**You said:**

when l run this command in Spyder Time\_Series = RetailSales.groupby(RetailSales['Date'].dt.to\_period('M').sum()) ['Total Amount'] is given error

**ChatGPT said:**

Ah! Good catch, Victoria 👀 — that line has a small **syntax error** that's causing the issue.

Let’s break it down and fix it together. Here's what you wrote:

python

CopyEdit

Time\_Series = RetailSales.groupby(RetailSales['Date'].dt.to\_period('M').sum())

['Total Amount']

**❌ What’s wrong?**

* .sum() is accidentally **inside** the .groupby() — which breaks the logic.
* 'Total Amount' is outside the groupby expression, which also causes confusion for Python.

**✅ Here's the Correct Version:**

python

CopyEdit

Time\_Series = RetailSales.groupby(RetailSales['Date'].dt.to\_period('M'))['Total Amount'].sum()

**🔍 What this does:**

* RetailSales['Date'].dt.to\_period('M') converts dates to monthly periods (like Jan 2025, Feb 2025, etc.)
* groupby(...) groups rows by each month
* ['Total Amount'].sum() adds up the total sales for each month

Now Time\_Series will be a Series showing total sales per month, like:

yaml

CopyEdit

2023-01 15420.00

2023-02 18950.00

2023-03 20345.50

...

**📊 You can now plot it:**

python

CopyEdit

Time\_Series.plot(title='Monthly Sales Trend', ylabel='Total Sales', xlabel='Month')

plt.show()

Let me know if your column names are different or if you’d like to add year-on-year comparisons!

4o

**You said:**

is still given this error AttributeError: 'list' object has no attribute 'sum'

**ChatGPT said:**

Ahh yes — that error makes total sense now. Let’s walk through **exactly what’s going wrong** and how to fix it, step-by-step.

**🛑 Your code likely looks like this:**

python

CopyEdit

Time\_Series = RetailSales.groupby(RetailSales['Date'].dt.to\_period('M')['Total Amount'].sum())

Or similar.

**❗ Problem:**

You're trying to do this:

python

CopyEdit

RetailSales['Date'].dt.to\_period('M')['Total Amount']

But RetailSales['Date'].dt.to\_period('M') returns a **Series**, and you’re treating it like a DataFrame — then trying to index it with 'Total Amount'.

That’s why you get:

AttributeError: 'list' object has no attribute 'sum'

**✅ The correct way:**

Here’s the **exact working line** you need:

python

CopyEdit

Time\_Series = RetailSales.groupby(RetailSales['Date'].dt.to\_period('M'))['Total Amount'].sum()

**🔍 What this does:**

1. RetailSales['Date'].dt.to\_period('M'): Converts each date to a monthly period (like "2025-04")
2. .groupby(...): Groups all the rows that fall in the same month
3. ['Total Amount'].sum(): Sums up sales for each month

**📈 To plot it:**

python

CopyEdit

Time\_Series.plot(title='Monthly Sales Trend', ylabel='Total Amount', xlabel='Month')

plt.show()

**💡 Pro tip:**

If RetailSales is still throwing errors, double-check:

* That 'Date' is in datetime format:

python

CopyEdit

RetailSales['Date'] = pd.to\_datetime(RetailSales['Date'])

* That 'Total Amount' is numeric:

python

CopyEdit

RetailSales['Total Amount'] = pd.to\_numeric(RetailSales['Total Amount'], errors='coerce')

Want me to review a sample of your column names or code to check if something's off?

4o

**You said:**

time\_series.plot(title='Monthly Sales Trend') is not displaying anything

**ChatGPT said:**

Got it! If time\_series.plot(title='Monthly Sales Trend') is **not displaying anything**, you're super close — we just need to explicitly **show the plot** using matplotlib.

**✅ Add plt.show() after your plot command:**

python

CopyEdit

import matplotlib.pyplot as plt

time\_series.plot(title='Monthly Sales Trend')

plt.xlabel('Month')

plt.ylabel('Total Sales')

plt.show()

**🧠 Why this happens:**

* In **Spyder** (or some other editors), plots don’t automatically show up unless plt.show() is called.
* plt.show() forces the plot window to appear.

**Bonus 🎁: Make It Prettier**

python

CopyEdit

plt.figure(figsize=(10, 5))

time\_series.plot(kind='line', marker='o', color='teal')

plt.title('Monthly Sales Trend', fontsize=14)

plt.xlabel('Month')

plt.ylabel('Total Amount')

plt.grid(True)

plt.tight\_layout()

plt.show()

Let me know if you'd like to overlay a **rolling average** or compare sales by year!

4o

Top of Form



Bottom of Form