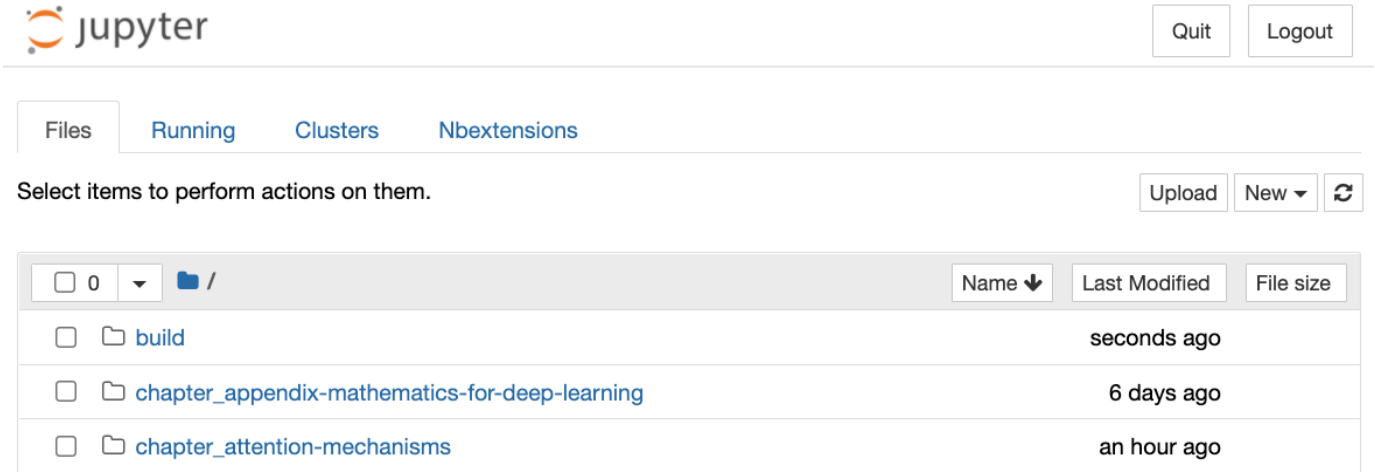
Using Jupyter Notebooks

:label: `sec_jupyter`

This section describes how to edit and run the code in each section of this book using the Jupyter Notebook. Make sure you have installed Jupyter and downloaded the code as described in :ref: `chap_installation` . If you want to know more about Jupyter see the excellent tutorial in their documentation.

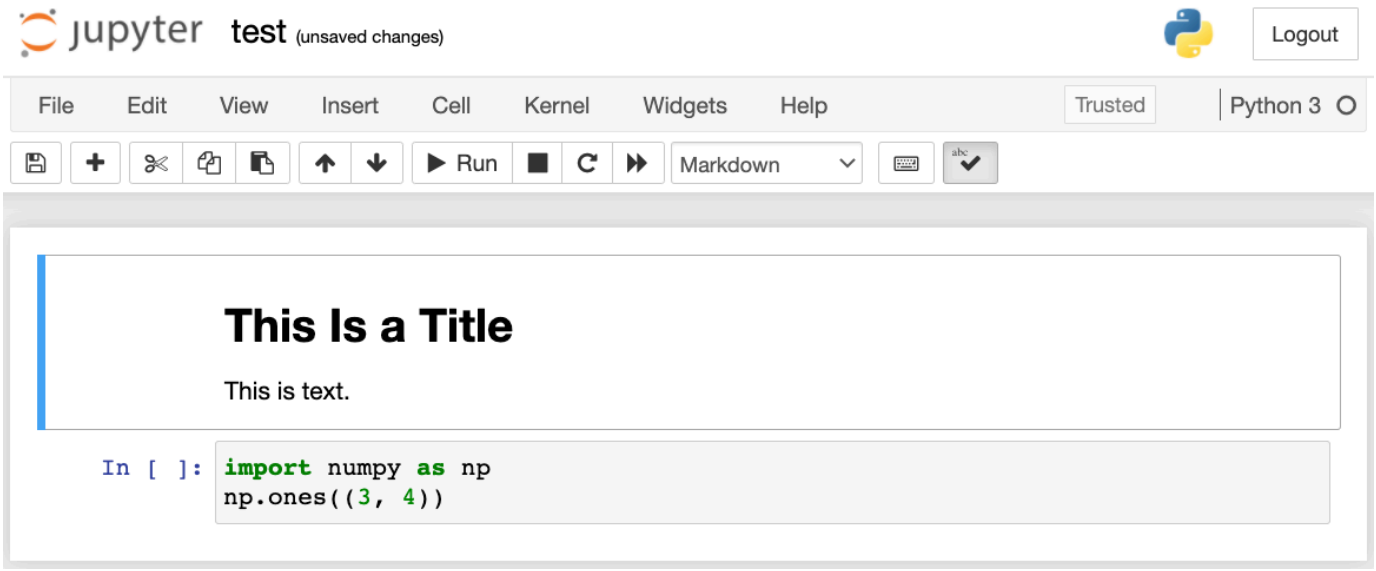
Editing and Running the Code Locally

Suppose that the local path of the book's code is `xx/yy/d2l-en/` . Use the shell to change the directory to this path (`cd xx/yy/d2l-en`) and run the command `jupyter notebook` . If your browser does not do this automatically, open http://localhost:8888 and you will see the interface of Jupyter and all the folders containing the code of the book, as shown in :numref: `fig_jupyter00` .



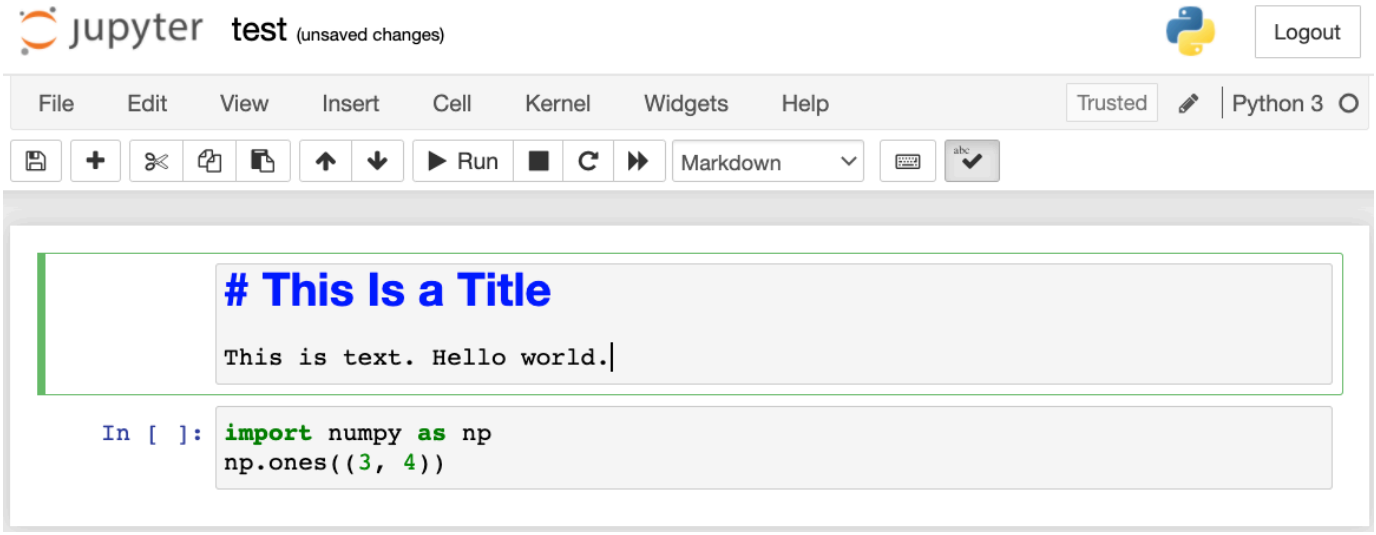
:width: `600px` :label: `fig_jupyter00`

You can access the notebook files by clicking on the folder displayed on the webpage. They usually have the suffix ".ipynb". For the sake of brevity, we create a temporary "test.ipynb" file. The content displayed after you click it is shown in :numref: `fig_jupyter01` . This notebook includes a markdown cell and a code cell. The content in the markdown cell includes "This Is a Title" and "This is text.". The code cell contains two lines of Python code.



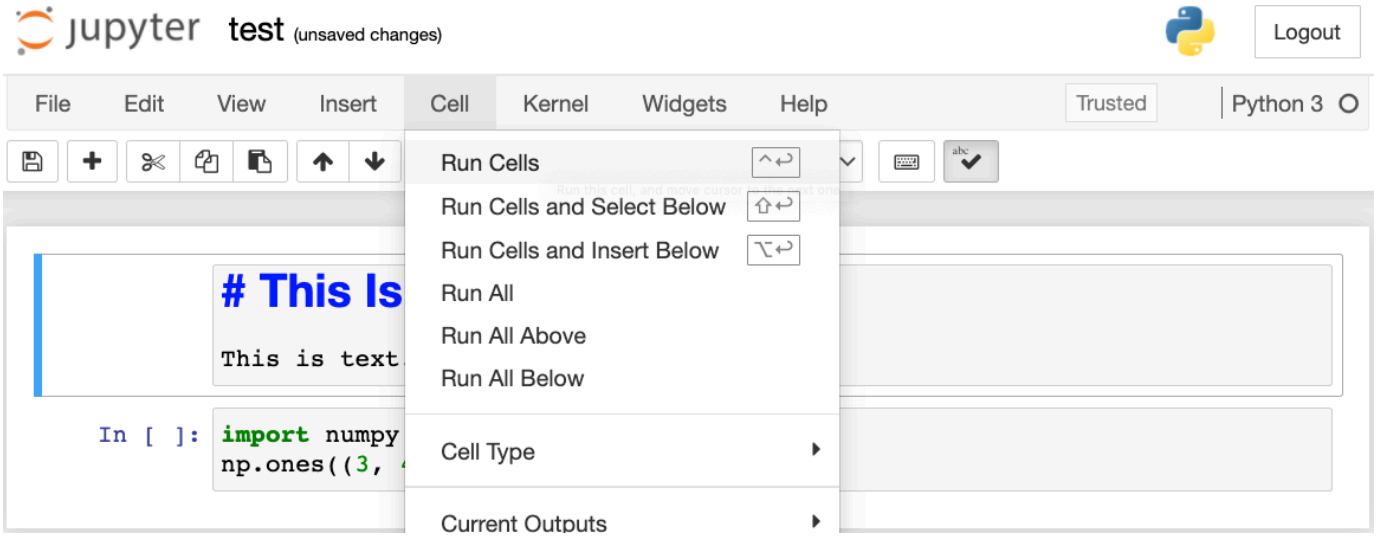
:width: `600px` :label: `fig_jupyter01`

Double click on the markdown cell to enter edit mode. Add a new text string "Hello world." at the end of the cell, as shown in :numref: `fig_jupyter02` .



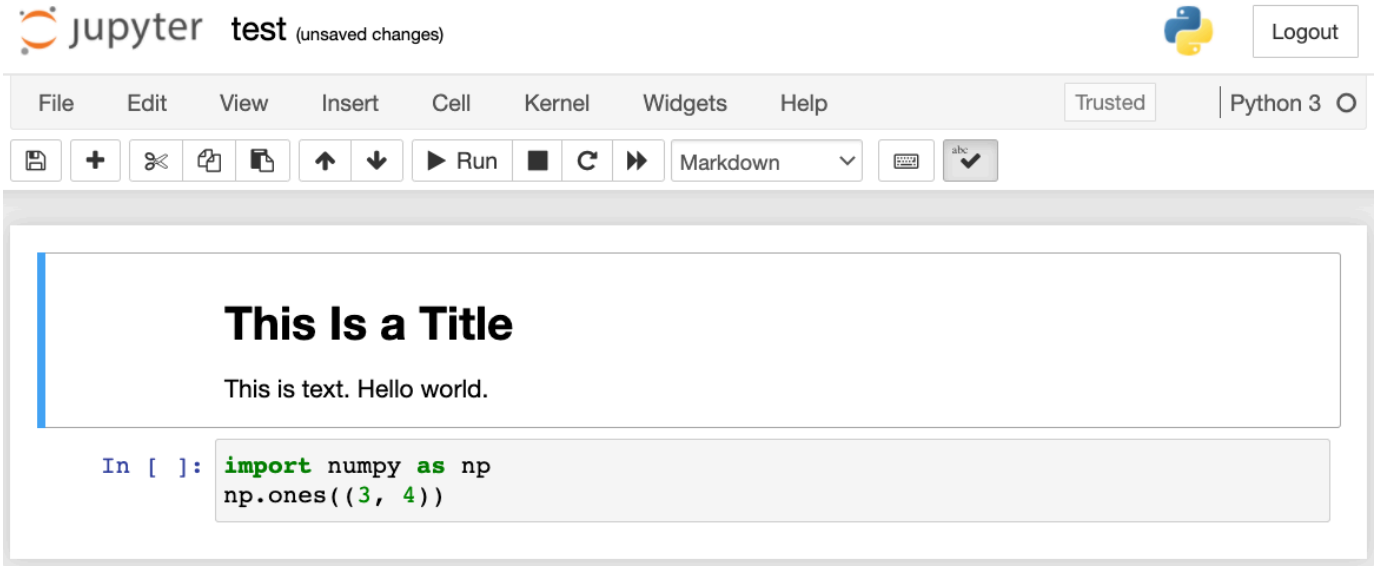
:width: `600px` :label: `fig_jupyter02`

As demonstrated in :numref: `fig_jupyter03` , click "Cell" → "Run Cells" in the menu bar to run the edited cell.



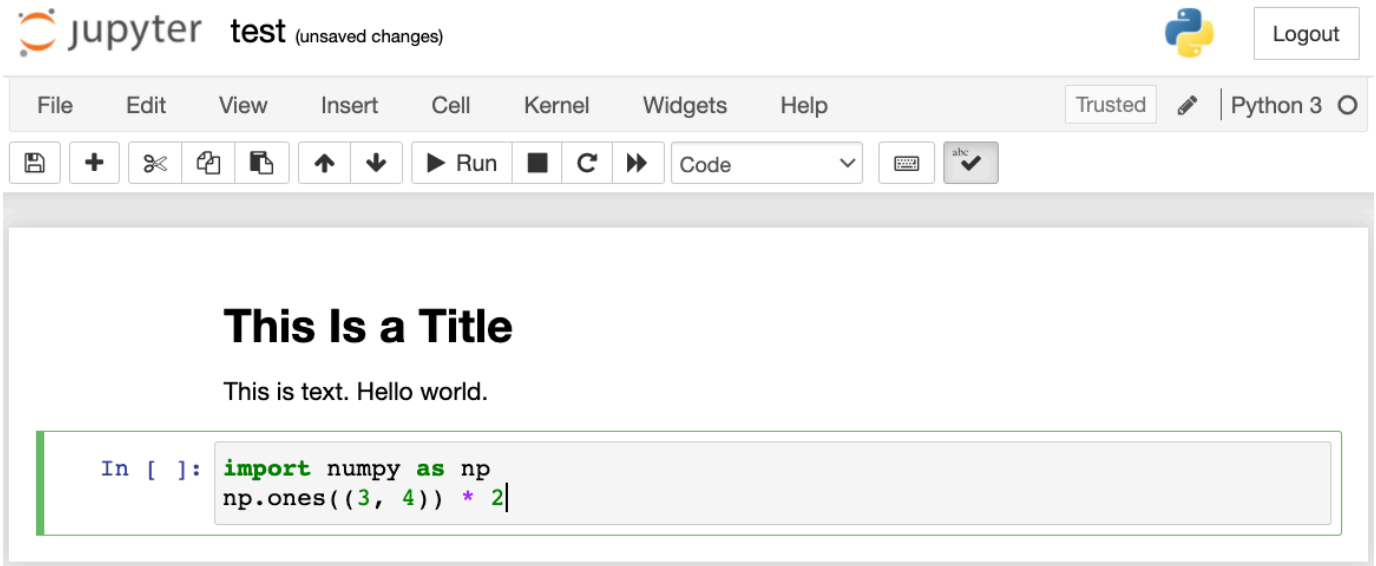
:width: `600px` :label: `fig_jupyter03`

After running, the markdown cell is shown in :numref: `fig_jupyter04` .



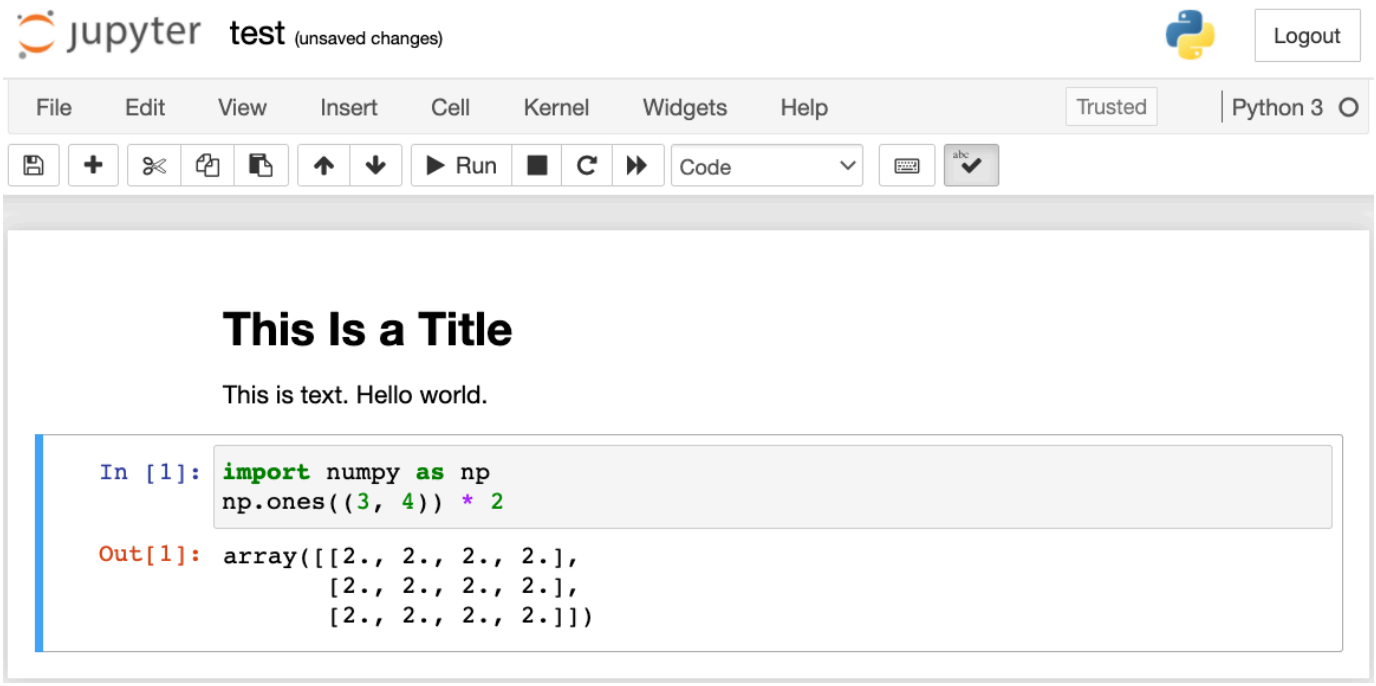
:width: `600px` :label: `fig_jupyter04`

Next, click on the code cell. Multiply the elements by 2 after the last line of code, as shown in :numref: `fig_jupyter05` .



:width: `600px` :label: `fig_jupyter05`

You can also run the cell with a shortcut ("Ctrl + Enter" by default) and obtain the output result from :numref: `fig_jupyter06` .



:width: `600px` :label: `fig_jupyter06`

When a notebook contains more cells, we can click "Kernel" → "Restart & Run All" in the menu bar to run all the cells in the entire notebook. By clicking "Help" → "Edit Keyboard Shortcuts" in the menu bar, you can edit the shortcuts according to your preferences.

Advanced Options

Beyond local editing two things are quite important: editing the notebooks in the markdown format and running Jupyter remotely. The latter matters when we want to run the code on a faster server. The former matters since Jupyter's native ipynb format stores a lot of auxiliary data that is irrelevant to the content, mostly related to how and where the code is run. This is confusing for Git, making reviewing contributions very difficult. Fortunately there is an alternative---native editing in the markdown format.

Markdown Files in Jupyter

If you wish to contribute to the content of this book, you need to modify the source file (md file, not ipynb file) on GitHub. Using the notedown plugin we can modify notebooks in the md format directly in Jupyter.

First, install the notedown plugin, run the Jupyter Notebook, and load the plugin:

```
pip install d2l-notedown  # You may need to uninstall the original notedown.
jupyter notebook --NotebookApp.contents_manager_class='notedown.NotedownContentsManager'
```

You may also turn on the notedown plugin by default whenever you run the Jupyter Notebook. First, generate a Jupyter Notebook configuration file (if it has already been generated, you can

skip this step).

```
jupyter notebook --generate-config
```

Then, add the following line to the end of the Jupyter Notebook configuration file (for Linux or macOS, usually in the path `~/.jupyter/jupyter_notebook_config.py`):

```
c.NotebookApp.contents_manager_class = 'notedown.NotedownContentsManager'
```

After that, you only need to run the `jupyter notebook` command to turn on the notedown plugin by default.

Running Jupyter Notebooks on a Remote Server

Sometimes, you may want to run Jupyter notebooks on a remote server and access it through a browser on your local computer. If Linux or macOS is installed on your local machine (Windows can also support this function through third-party software such as PuTTY), you can use port forwarding:

```
ssh myserver -L 8888:localhost:8888
```

The above string `myserver` is the address of the remote server. Then we can use http://localhost:8888 to access the remote server `myserver` that runs Jupyter notebooks. We will detail on how to run Jupyter notebooks on AWS instances later in this appendix.

Timing

We can use the `ExecuteTime` plugin to time the execution of each code cell in Jupyter notebooks. Use the following commands to install the plugin:

```
pip install jupyter_contrib_nbextensions
jupyter contrib nbextension install --user
jupyter nbextension enable execute_time/ExecuteTime
```

Summary

- Using the Jupyter Notebook tool, we can edit, run, and contribute to each section of the book.
- We can run Jupyter notebooks on remote servers using port forwarding.

Exercises

1. Edit and run the code in this book with the Jupyter Notebook on your local machine.
2. Edit and run the code in this book with the Jupyter Notebook *remotely* via port forwarding.

3. Compare the running time of the operations $\mathbf{A}^\top \mathbf{B}$ and $\mathbf{AB}$ for two square matrices in $\mathbb{R}^{1024\times 1024}$. Which one is faster?

Discussions

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from google.colab import files
uploaded = files.upload()
```

••• Choose Files No file chosen Cancel upload

```
# Load the datasets
customers = pd.read_csv("Customers.csv")
products = pd.read_csv("Products.csv")
transactions = pd.read_csv("Transactions.csv")


# Display the first few rows of each dataset
print("Customers Dataset:\n", customers.head())
print("\nProducts Dataset:\n", products.head())
print("\nTransactions Dataset:\n", transactions.head())


# Check for missing values
print("\nMissing values in Customers dataset:\n", customers.isnull().sum())
print("\nMissing values in Products dataset:\n", products.isnull().sum())
print("\nMissing values in Transactions dataset:\n", transactions.isnull().sum())


# Check the basic statistics of each dataset
print("\nCustomers Dataset Statistics:\n", customers.describe(include='all'))
print("\nProducts Dataset Statistics:\n", products.describe(include='all'))
print("\nTransactions Dataset Statistics:\n", transactions.describe(include='all'))


# Merge the datasets for combined analysis
merged_data = transactions.merge(customers, on="CustomerID", how="left")
merged_data = merged_data.merge(products, on="ProductID", how="left")


# Display the merged dataset
print("\nMerged Dataset:\n", merged_data.head())


# EDA: Overview of data
print("\nBasic Info of Merged Dataset:\n")
print(merged_data.info())
print("\nSummary Statistics:\n", merged_data.describe())


# Convert dates to datetime format
merged_data['TransactionDate'] = pd.to_datetime(merged_data['TransactionDate'])
```

```
customers['SignupDate'] = pd.to_datetime(customers['SignupDate'])

# Top 5 most purchased products
top_products = merged_data['ProductName'].value_counts().head(5)
print("\nTop 5 Most Purchased Products:\n", top_products)


# Total revenue generated by category
category_revenue = merged_data.groupby('Category')['TotalValue'].sum().sort_values(ascend
print("\nTotal Revenue by Category:\n", category_revenue)

# Most active regions
region_activity = merged_data['Region'].value_counts()
print("\nMost Active Regions:\n", region_activity)

# Monthly transaction trends
merged_data['Month'] = merged_data['TransactionDate'].dt.to_period('M')
monthly_trends = merged_data.groupby('Month')['TotalValue'].sum()
print("\nMonthly Transaction Trends:\n", monthly_trends)


# EDA Visualizations
plt.figure(figsize=(12, 6))
sns.barplot(x=top_products.index, y=top_products.values, palette="viridis")
plt.title("Top 5 Most Purchased Products")
plt.ylabel("Number of Purchases")
plt.xlabel("Product Name")
plt.xticks(rotation=45)
plt.show()

plt.figure(figsize=(12, 6))
category_revenue.plot(kind='bar', color='orange')
plt.title("Revenue by Category")
plt.ylabel("Total Revenue (USD)")
plt.xlabel("Category")
plt.show()

plt.figure(figsize=(12, 6))
region_activity.plot(kind='bar', color='skyblue')
plt.title("Most Active Regions")
plt.ylabel("Number of Transactions")
plt.xlabel("Region")
plt.show()

plt.figure(figsize=(12, 6))
monthly_trends.plot(kind='line', marker='o', color='green')
plt.title("Monthly Transaction Trends")
plt.ylabel("Total Revenue (USD)")
plt.xlabel("Month")
plt.xticks(rotation=45)
plt.show()


# Business Insights
print("\nBusiness Insights:")
print("1. The top 5 most purchased products contribute significantly to total sales, sugg
```

```
print("2. The highest revenue is generated from the 'Category X' (replace X with actual c
print("3. Customers from the 'Region Y' (replace Y with actual region) are the most activ
print("4. Monthly transaction trends reveal seasonal spikes in revenue, suggesting the po
```