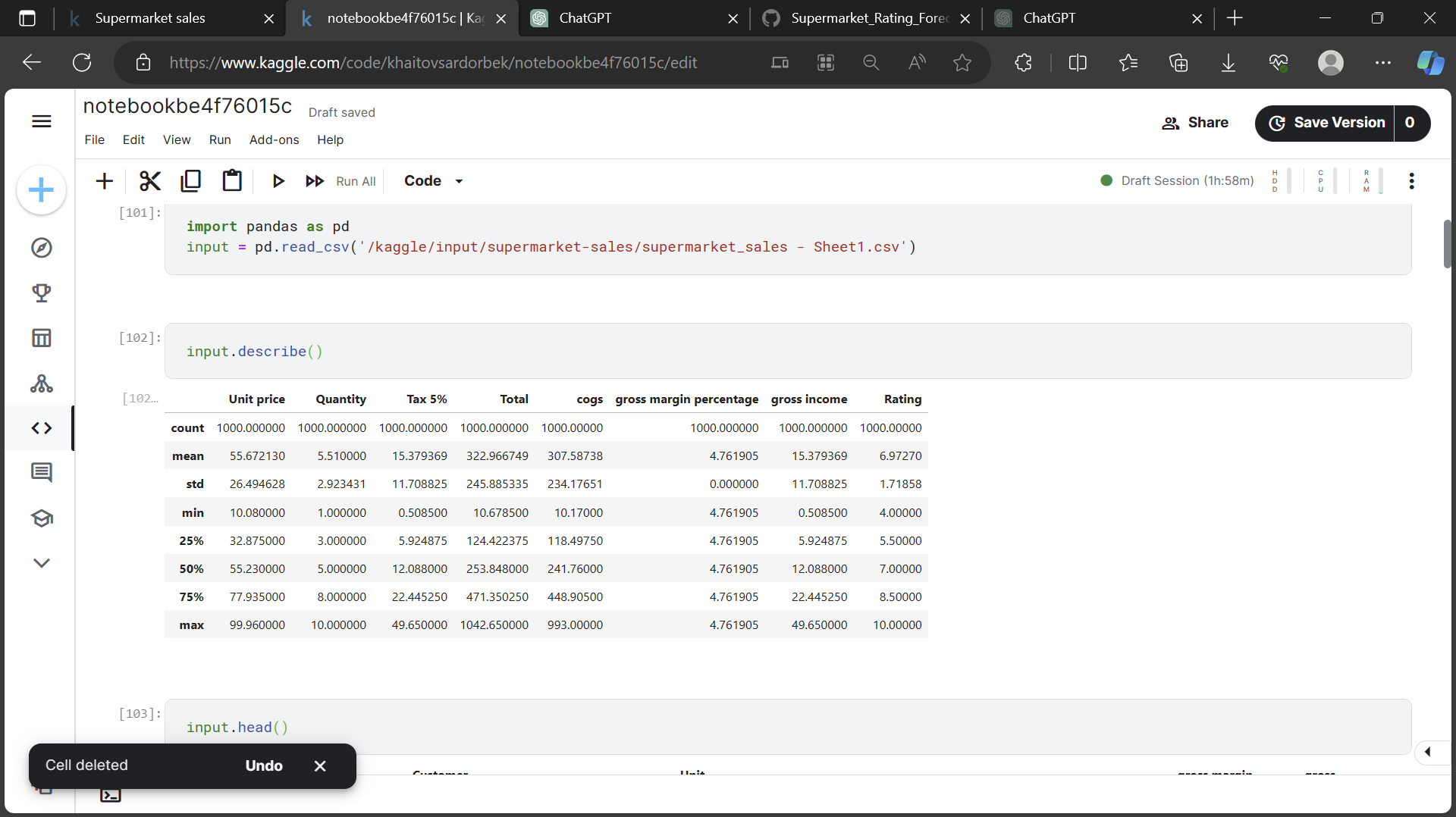
Codes and Results

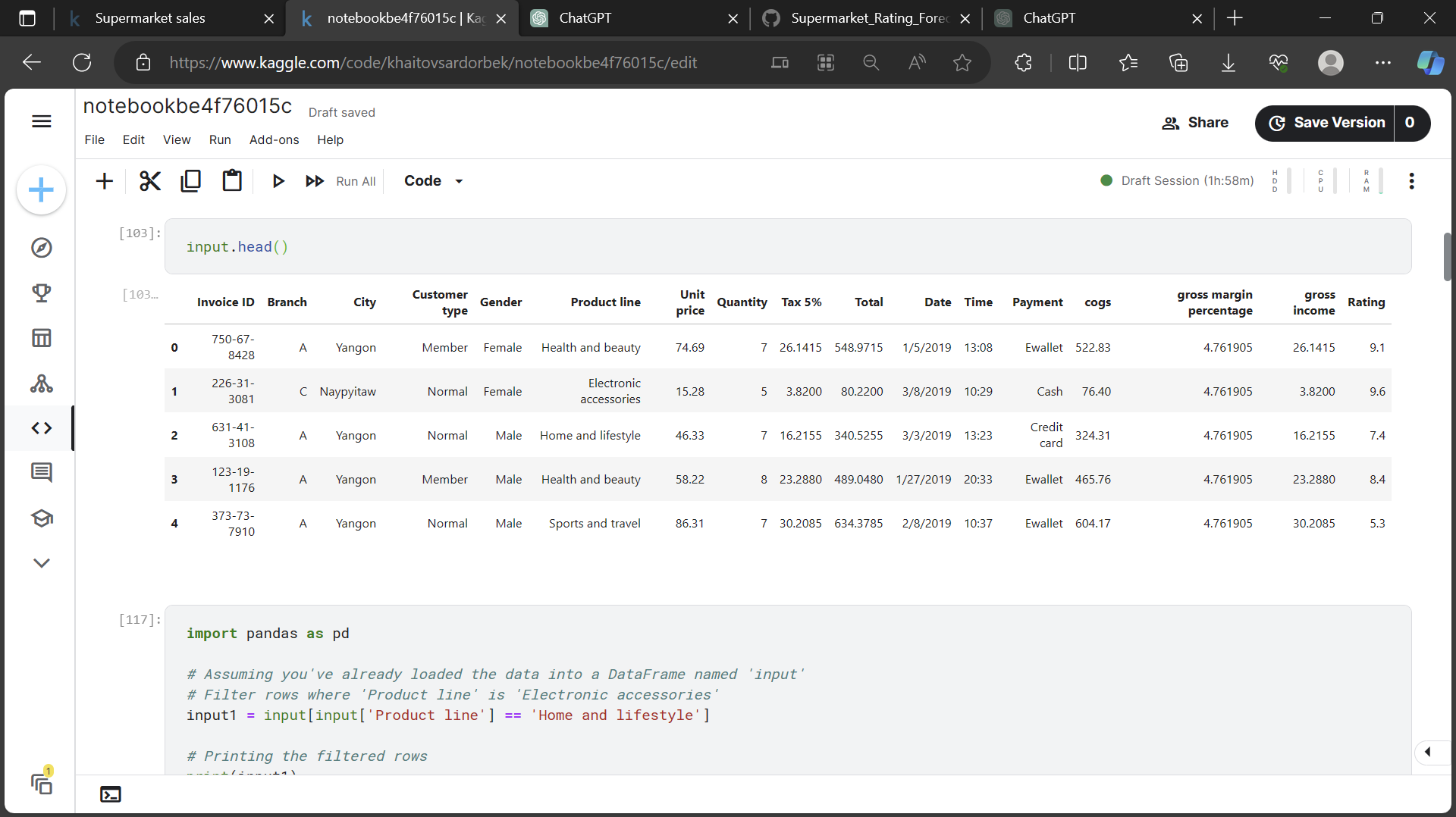
import pandas as pd

input = pd.read\_csv('/kaggle/input/supermarket-sales/supermarket\_sales - Sheet1.csv')

input.describe()



input.head()



import pandas as pd

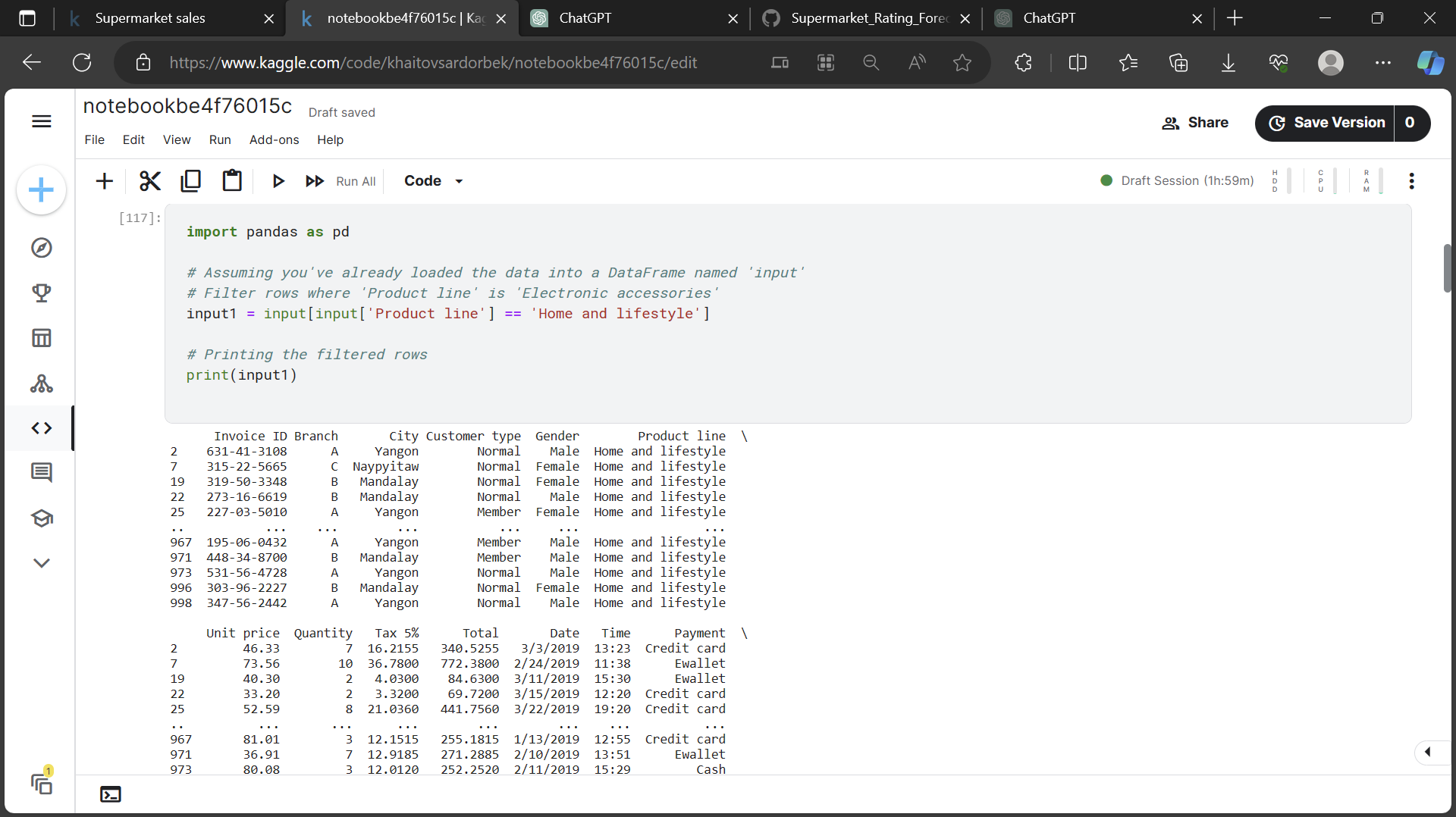
# Assuming you've already loaded the data into a DataFrame named 'input'

# Filter rows where 'Product line' is 'Electronic accessories'

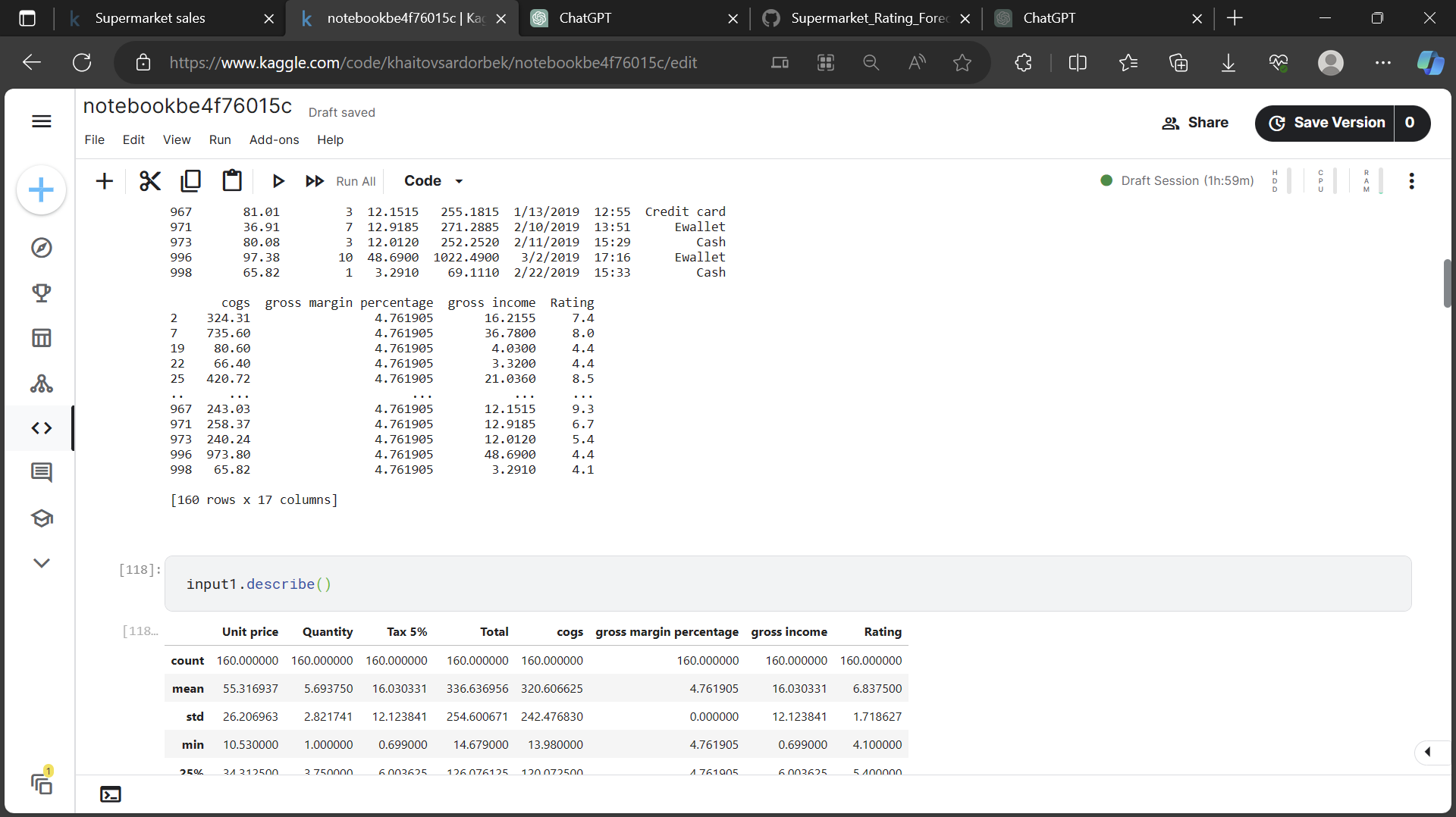
input1 = input[input['Product line'] == 'Home and lifestyle']

# Printing the filtered rows

print(input1)



input1.describe()



import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming 'input1' is your DataFrame

# Data Overview

print(input1.head())

print(input1.describe())

# Data Cleaning (if needed)

# Check for missing values: input1.isnull().sum()

# Handle missing values if any

# EDA

# Example: Plotting histograms

numeric\_cols = ['Unit price', 'Quantity', 'Total']

for col in numeric\_cols:

plt.figure(figsize=(6, 4))

sns.histplot(input1[col], kde=True)

plt.title(f'Distribution of {col}')

plt.show()

# Feature Engineering

# Extract information from Date and Time columns if needed

# Visualization

# Example: Correlation matrix

plt.figure(figsize=(8, 6))

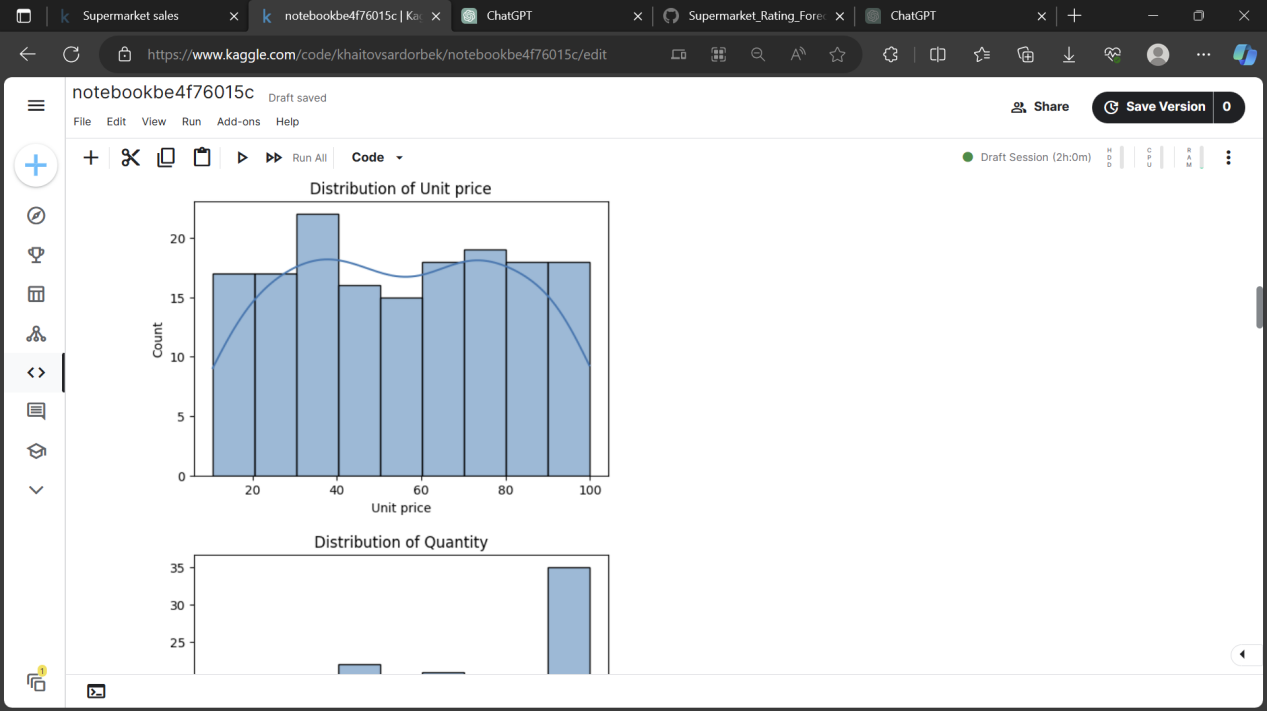
sns.heatmap(input1.corr(), annot=True, cmap='coolwarm', fmt='.2f')

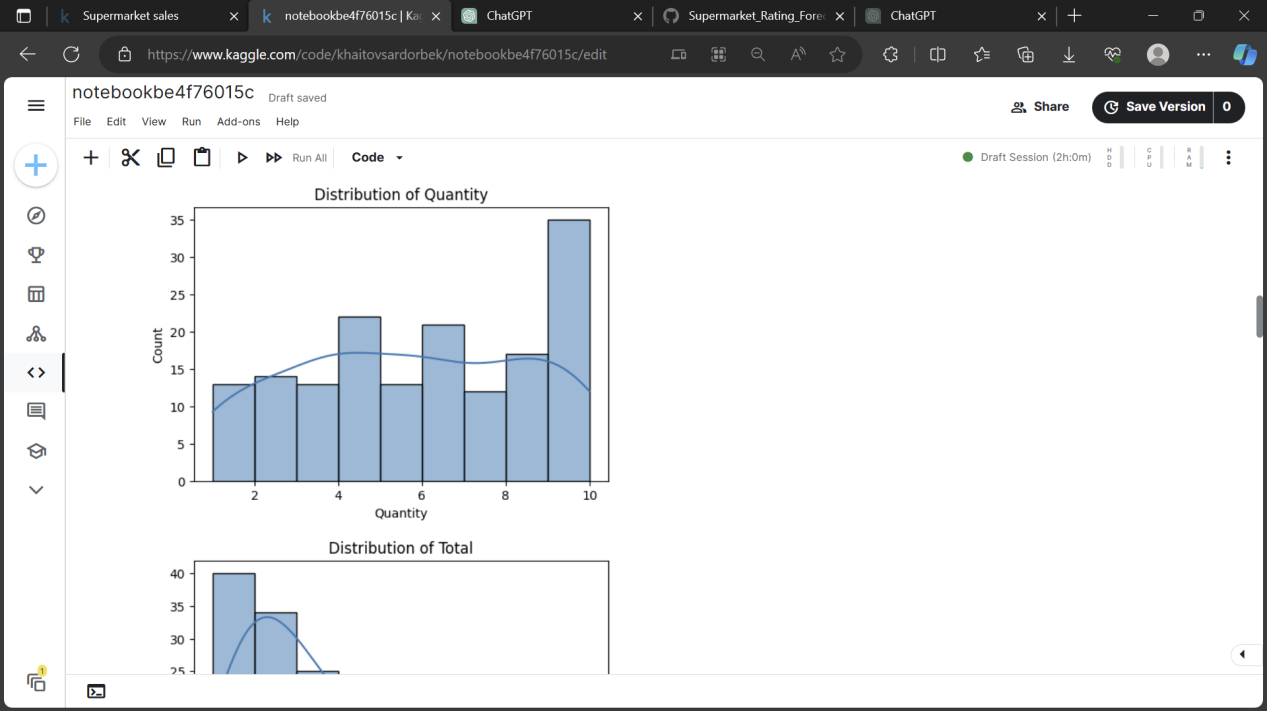
plt.title('Correlation Matrix')

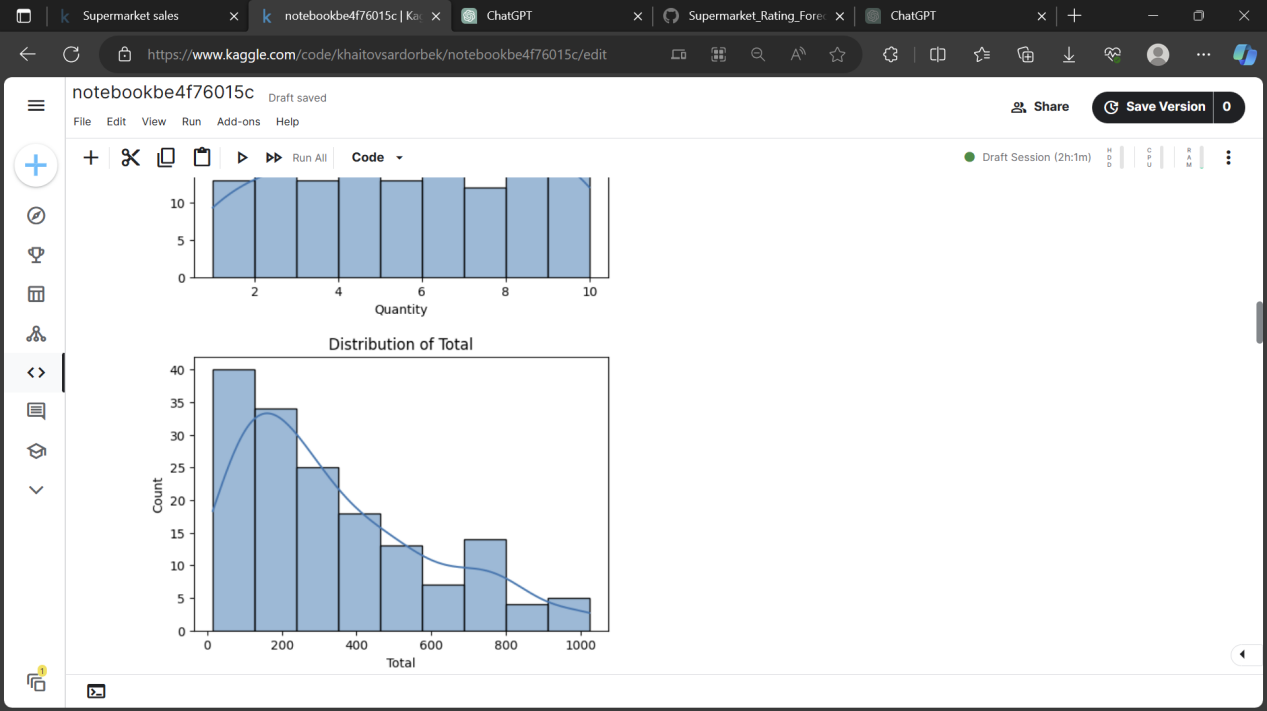
plt.show()

# Insights

# Provide insights or recommendations based on the analysis







import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Assuming 'input1' is your DataFrame

# Selecting only numeric columns for correlation matrix

numeric\_cols = input1.select\_dtypes(include=['float64', 'int64']).columns

# Compute correlation matrix for numeric columns

correlation\_matrix = input1[numeric\_cols].corr()

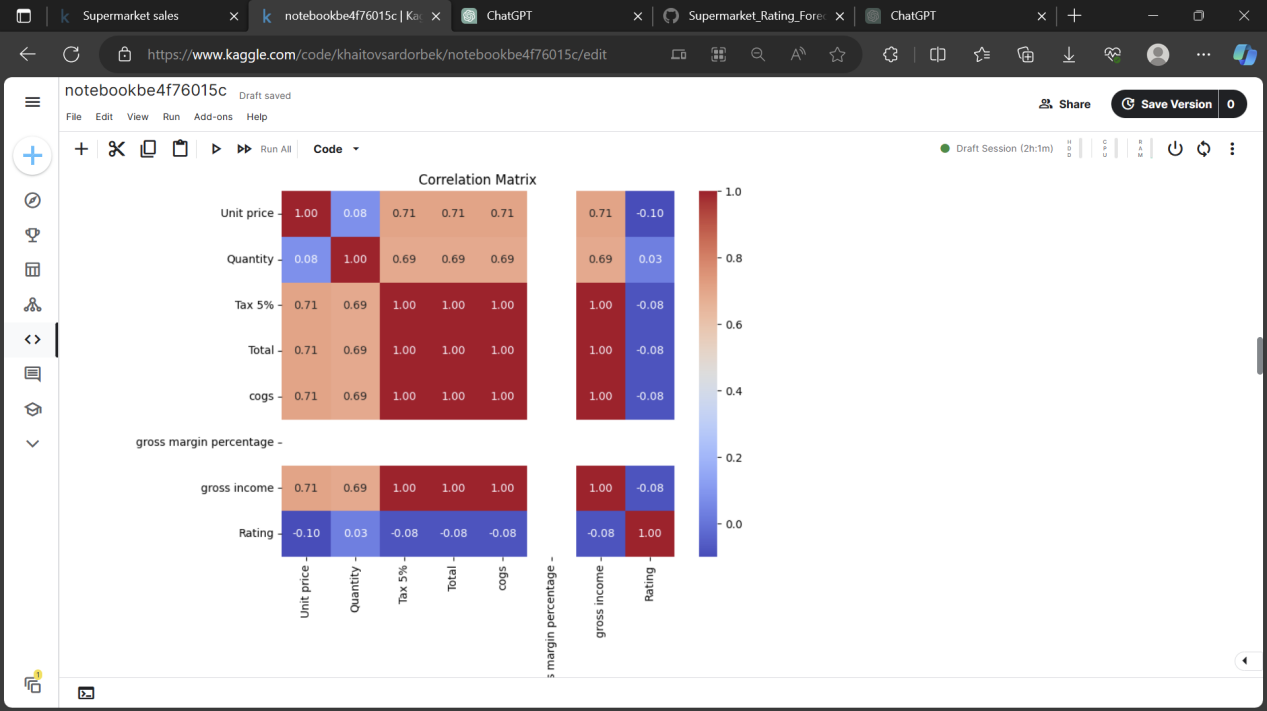
# Visualization - Correlation matrix

plt.figure(figsize=(8, 6))

sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', fmt='.2f')

plt.title('Correlation Matrix')

plt.show()



import pandas as pd

from sklearn.model\_selection import train\_test\_split

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import GRU, Dense

from tensorflow.keras.preprocessing.sequence import pad\_sequences

# Selecting the columns of interest

selected\_columns = ['Quantity', 'Tax 5%', 'Total', 'cogs', 'gross margin percentage', 'gross income', 'Rating']

X = input1[selected\_columns] # Features

# Target variable is 'Unit price'

y = input1['Unit price']

# Splitting the data into train and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Building the GRU model

model = Sequential()

model.add(Dense(64, input\_shape=(X\_train.shape[1],))) # Dense layer

model.add(Dense(32)) # Additional Dense layer (optional

model.add(Dense(1)) # Output layer for regression

# Compile the model

model.compile(optimizer='adam', loss='mse') # Using mean squared error for regression

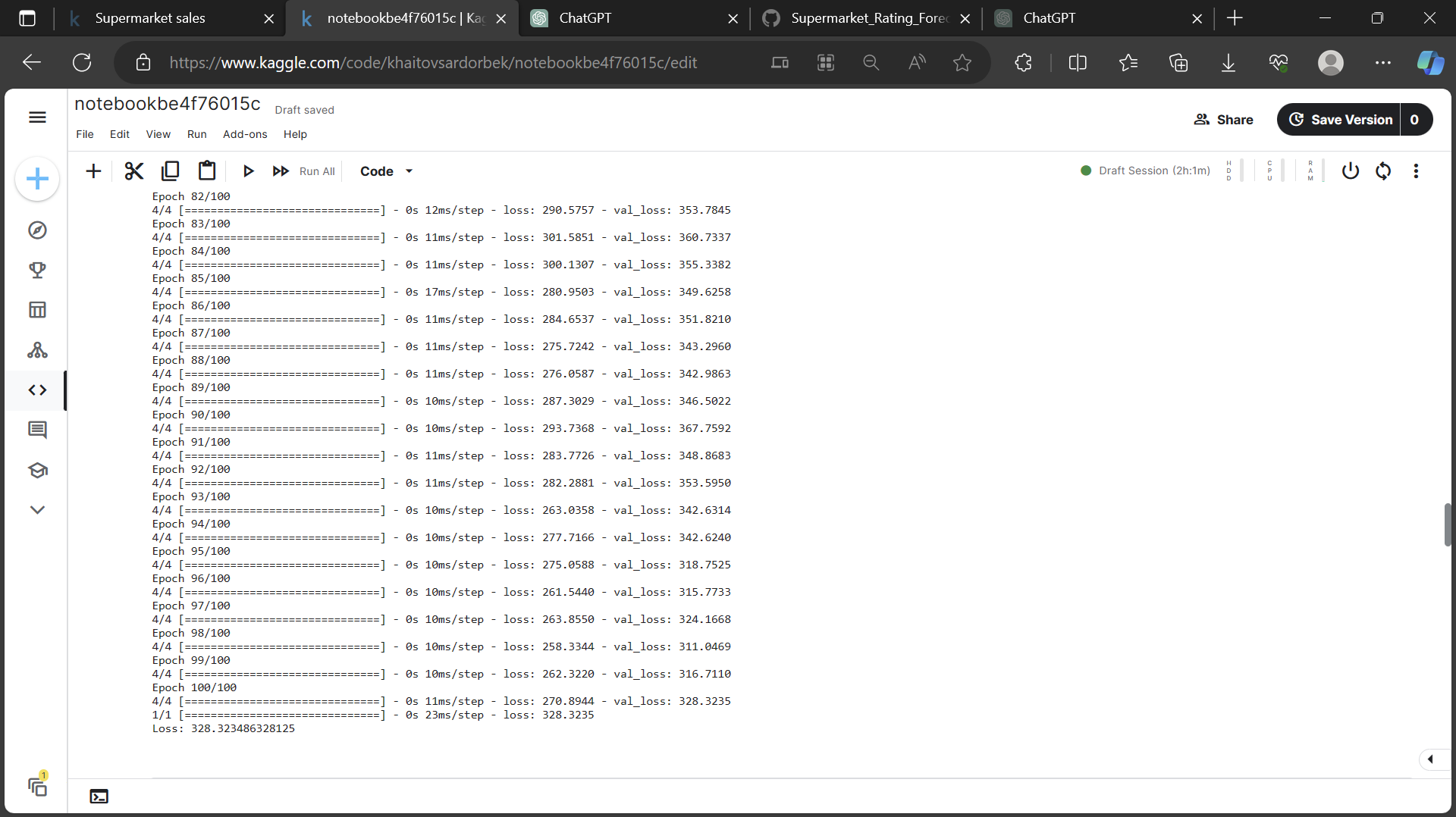
# Train the model

model.fit(X\_train, y\_train, epochs=100, batch\_size=32, validation\_data=(X\_test, y\_test))

# Evaluate the model

loss = model.evaluate(X\_test, y\_test)

print(f"Loss: {loss}")



# Evaluate the model

loss = model.evaluate(X\_test, y\_test)

print(f"Loss: {loss}")

# Get model predictions

predictions = model.predict(X\_test).flatten()

# Visualize actual vs predicted values

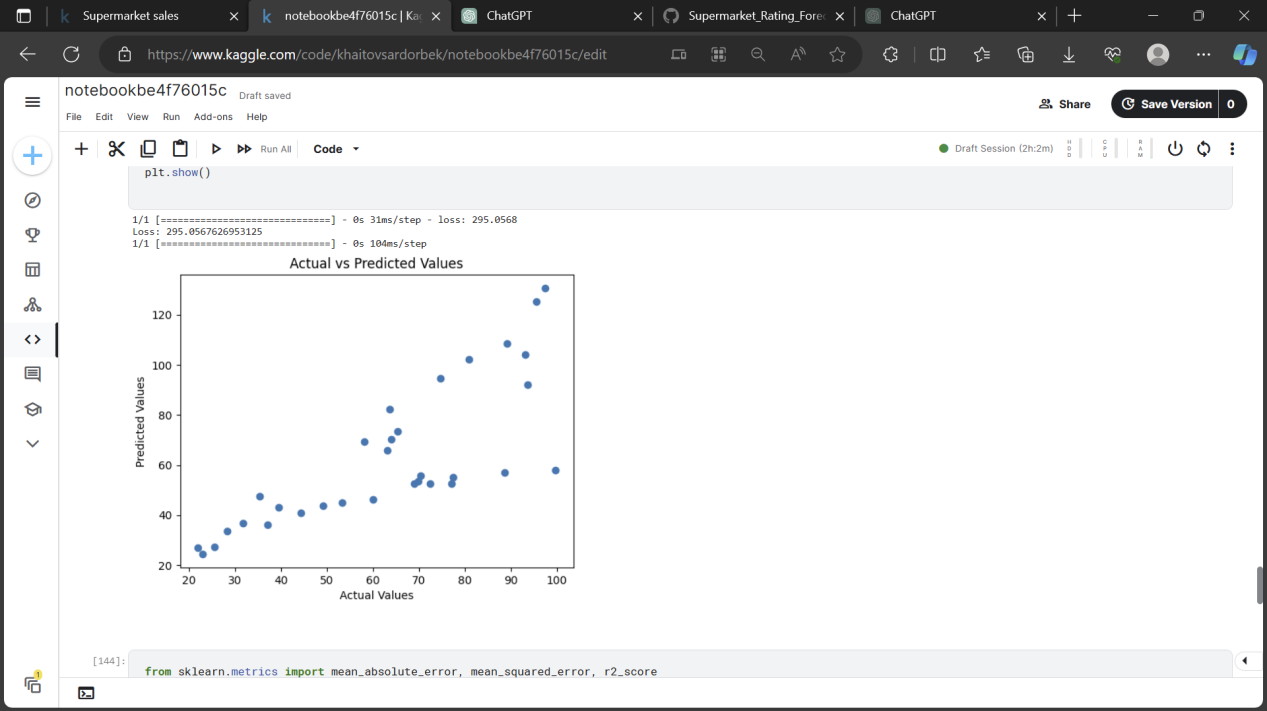
plt.scatter(y\_test, predictions)

plt.xlabel('Actual Values')

plt.ylabel('Predicted Values')

plt.title('Actual vs Predicted Values')

plt.show()



import matplotlib.pyplot as plt

import seaborn as sns

# Plotting for training set

plt.figure(figsize=(8, 6))

sns.scatterplot(x=y\_train, y=train\_predictions.flatten())

plt.xlabel('True Values')

plt.ylabel('Predictions')

plt.title('Training Set: True Values vs Predictions')

plt.show()

# Plotting for testing set

plt.figure(figsize=(8, 6))

sns.scatterplot(x=y\_test, y=test\_predictions.flatten())

plt.xlabel('True Values')

plt.ylabel('Predictions')

plt.title('Testing Set: True Values vs Predictions')

plt.show()

