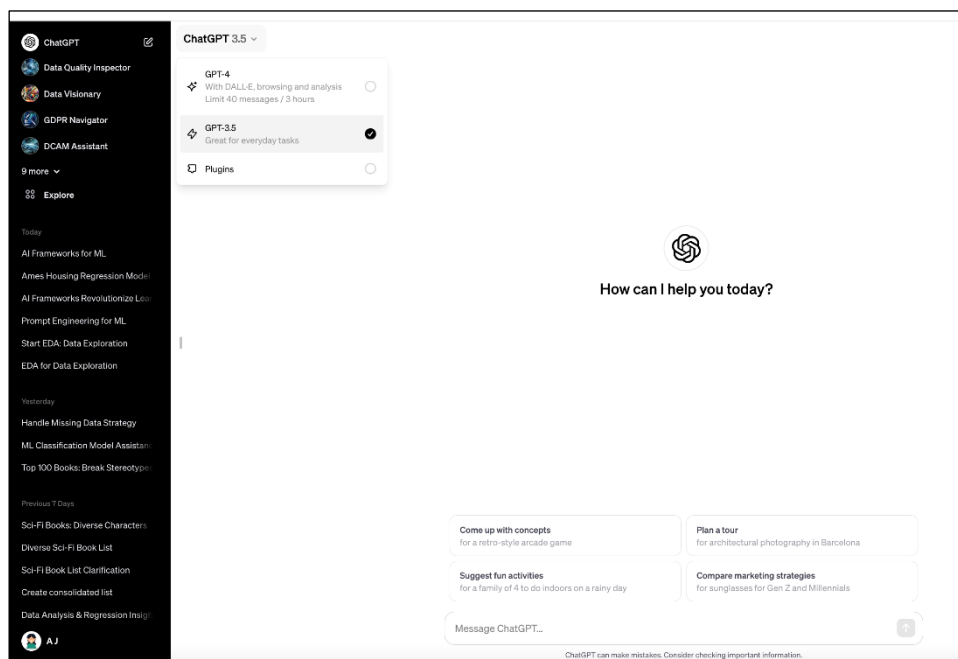
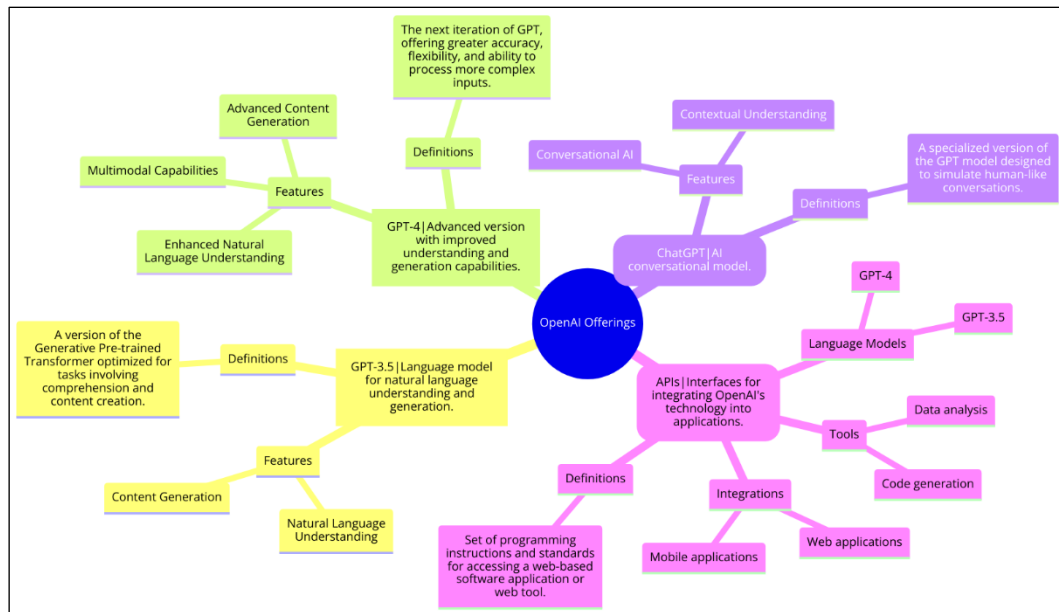


Chapter 3: Tools of the Trade: Introducing Our AI Assistants



Custom instructions

What would you like ChatGPT to know about you to provide better responses?

I am working on machine learning algorithms and data analysis

62/1500

How would you like ChatGPT to respond?

Behave as if you are an expert in the requested field.

For data preparation, use tasks like handling missing values, detecting outliers, feature encoding, scaling, etc.

- For EDA uncover insights through statistical analysis, data visualization, subgroup comparisons, etc.

1001/1500

Enable for new chats ☒

Cancel

Save

ChatGPT 3.5



You

Can you please explain the process of machine learning in bullet points to me ?



ChatGPT

Certainly! Here's an overview of the machine learning process in bullet points, following a systematic framework:

1. Problem Definition:

- Clearly define the problem you want to solve with machine learning.
- Determine whether it's a classification, regression, clustering, or other types of problem.

2. Data Collection:

- Gather relevant data that will be used to train and evaluate your machine learning model.
- Ensure the data is representative and of good quality.

3. Data Preprocessing:

- Handle missing values: Decide whether to remove, impute, or leave them as-is.
- Detect and handle outliers: Identify and deal with data points that deviate significantly from the norm.
- Feature encoding: Convert categorical variables into numerical representations using techniques like one-hot encoding or label encoding.
- Feature scaling: Normalize or standardize numerical features to have a consistent scale.

4. Exploratory Data Analysis (EDA):

- Conduct statistical analysis: Calculate descriptive statistics, correlations, and distributions.



Message ChatGPT...



ChatGPT can make mistakes. Consider checking important information.

Chapter 4: Build the Appearance of Our App with HTML and Copilot

Section 1

Lorem ipsum dolor sit amet consectetur adipiscing elit. Quisquam, voluptatum.



Section 2

Lorem ipsum dolor sit amet consectetur adipiscing elit. Quisquam, voluptatum.



Login

[Create User](#) [Forgot Password](#)

Products



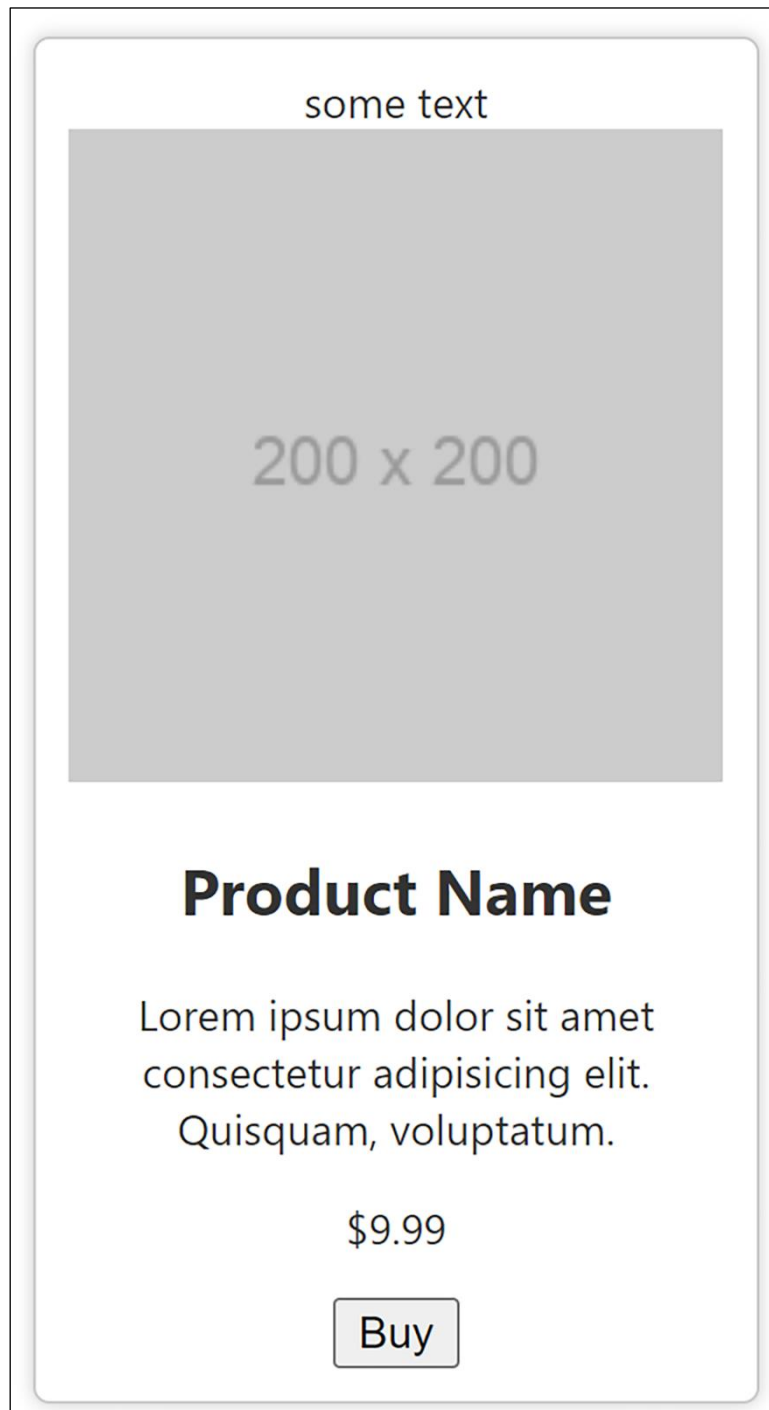
Product 1

Lorem ipsum dolor sit amet consectetur adipisicing elit. Quisquam, voluptatum.

[View Product](#)



Chapter 5: Style the App with CSS and Copilot



Product 2

Price: 200

Quantity: 2

Sum: 400

+

-

Product 3

Price: 300

Quantity: 3

Sum: 900

+

-

Chapter 6: Add Behavior with JavaScript

item1	Quantity: 1	Price: 10	Total: 10
item2	Quantity: 1	Price: 20	Total: 20
item3	Quantity: 1	Price: 30	Total: 30
Total price: 60			

Product 1

Price: 100

Quantity: 1

Sum: 100

+

-

Product 2

Price: 200

Quantity: 2

Sum: 400

+

-

Product 3

Price: 300

Quantity: 3

Sum: 900

+

-

Product 1

Price: 100

Quantity: 2

Sum: 200

+

-

Product 2

Price: 200

Quantity: 2

Sum: 400

+

-

Chapter 7: Support Multiple Viewports Using Responsive Web Layouts



Home

About

Contact

Gallery

Login

main
area

Home

About

Contact

Gallery

Login

main area

Home
About
Contact
Gallery
Login



Product 1

Description

Price

Add to cart



Product 2

Description

Price

Add to cart



Product 3

Description

Price

Add to cart



Product 4

Description

Price

Add to cart



Product 5

Description

Price

Add to cart



Product 6

Description

Price

Add to cart

Home

About

Contact

Gallery

Login



Product 1

Description

Price

Add to
cart



Product 2

Description

Price

Add to
cart



Product 3

Description

Price

Add to
cart



Home

About

Contact

Gallery

Login



Product 1

Description

Price

Memory Game

0

Selected:

Chapter 8: Build a Backend with Web APIs

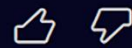


softchris

add code to respond with JSON
for all routes



GitHub Copilot



Sure, here's an example of how you can modify all routes to return a JSON response:



```
from flask import Flask
```

```
# create app
```

```
app = Flask(__name__)
```

```
# default route
```

```
@app.route('/')
```

```
def index():
```

```
    return jsonify({'me
```

```
# product catalog
```

```
@app.route('/products')
```

```
def products():
```

```
    products = [
```

```
        {'id': 1, 'name
```

```
        {'id': 2, 'name
```

```
        {'id': 3, 'name
```

```
    ]
```

```
    return jsonify(prod
```

```
# shopping cart
```

```
@app.route('/cart')
```

```
def cart():
```

```
    return jsonify({'me
```



A swagger API 0.0.1

[/apispec_1.json](#)

powered by Flasgger

[Terms of service](#)

cart

GET /cart ---

POST /cart/add ---

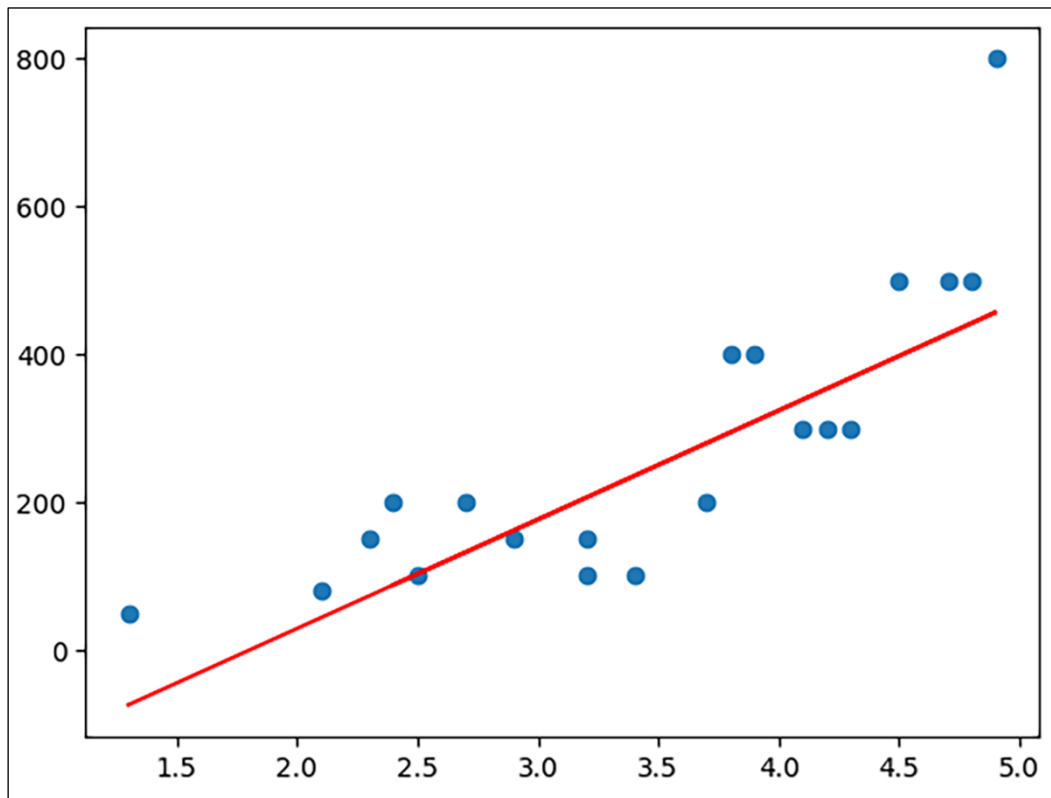
checkout

POST /checkout ---

products

GET /products ---

Chapter 9: Augment Web Apps with AI Services



Chapter 11: Data Exploration with ChatGPT

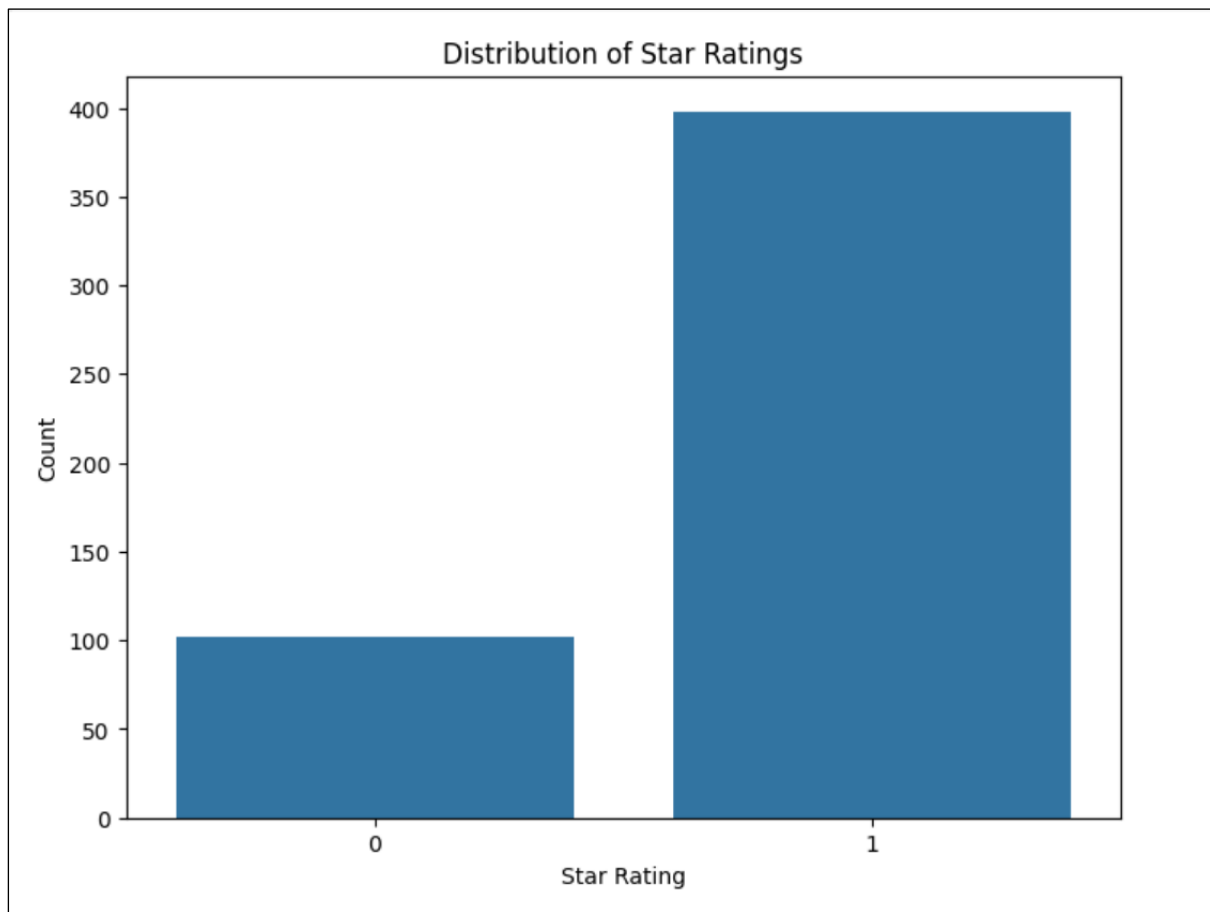
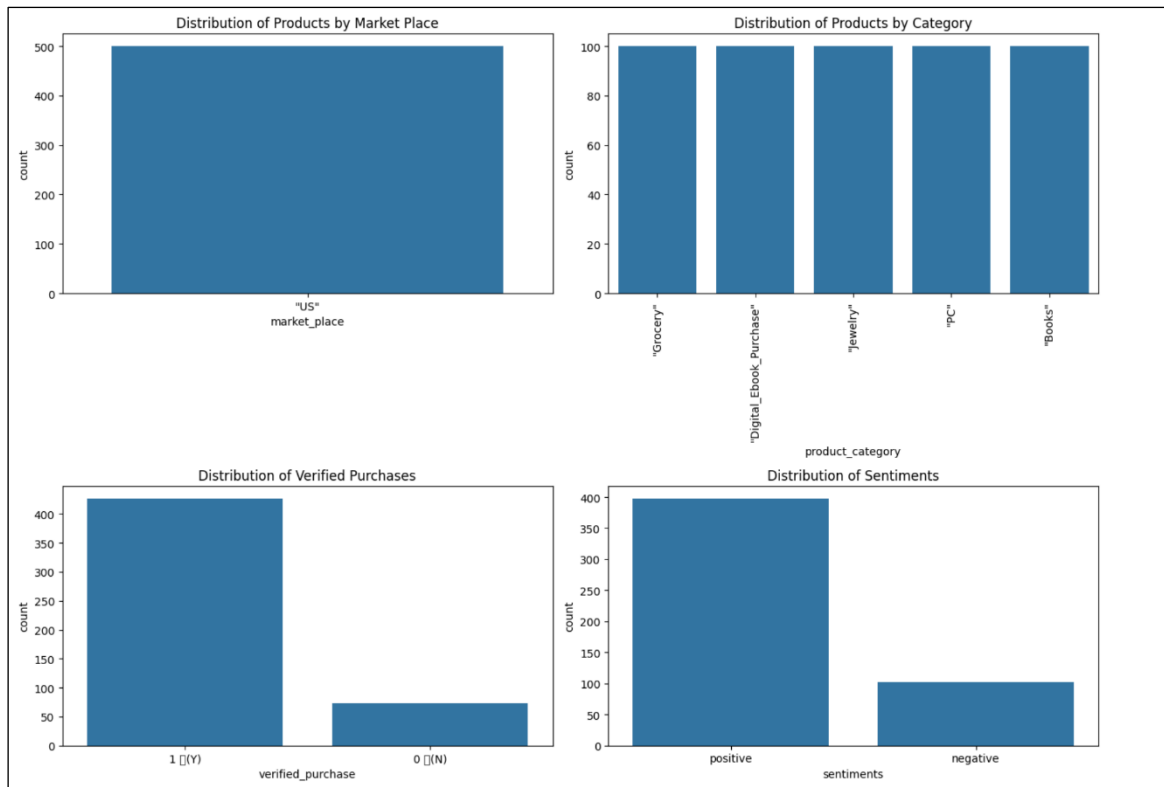
Permit this notebook to access your Google Drive files?

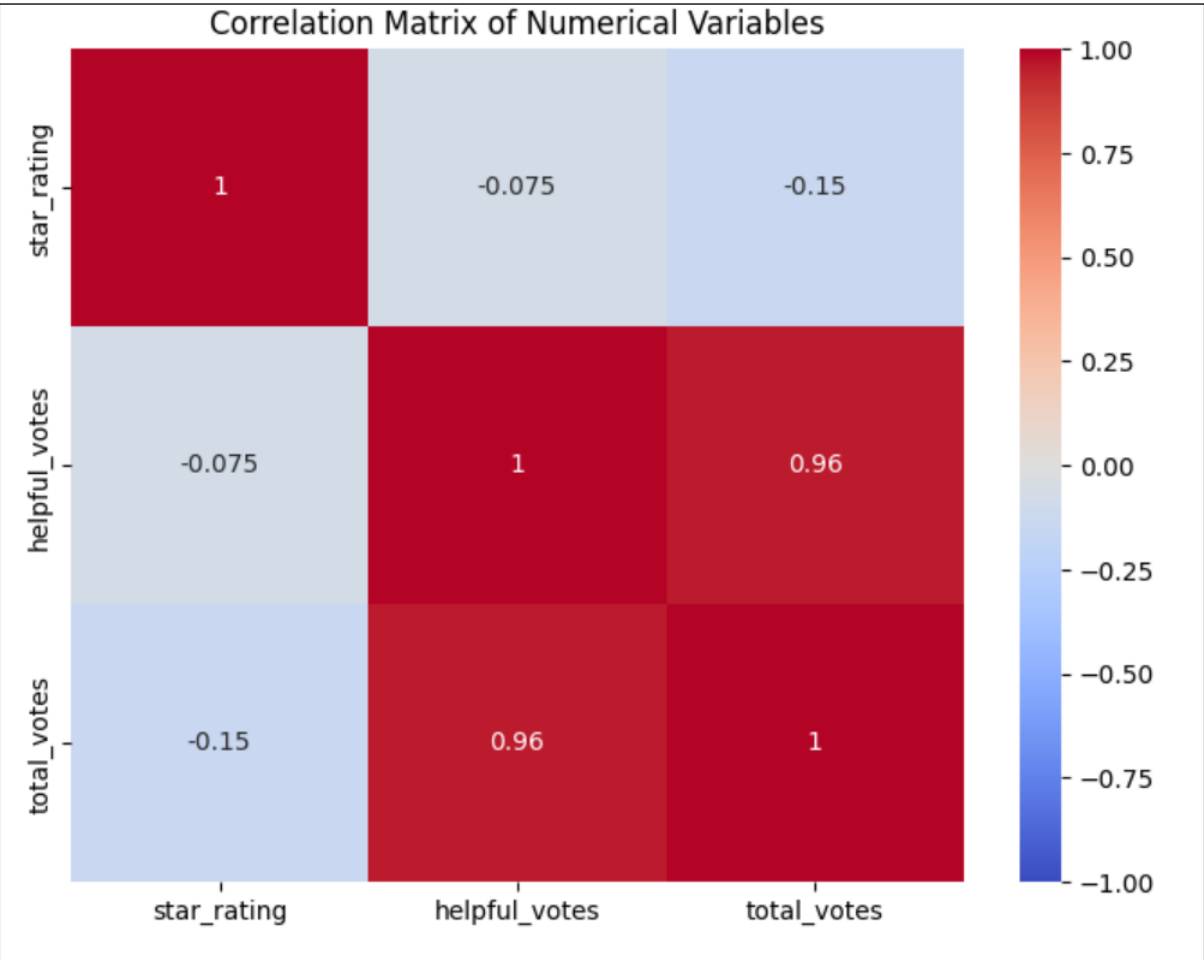
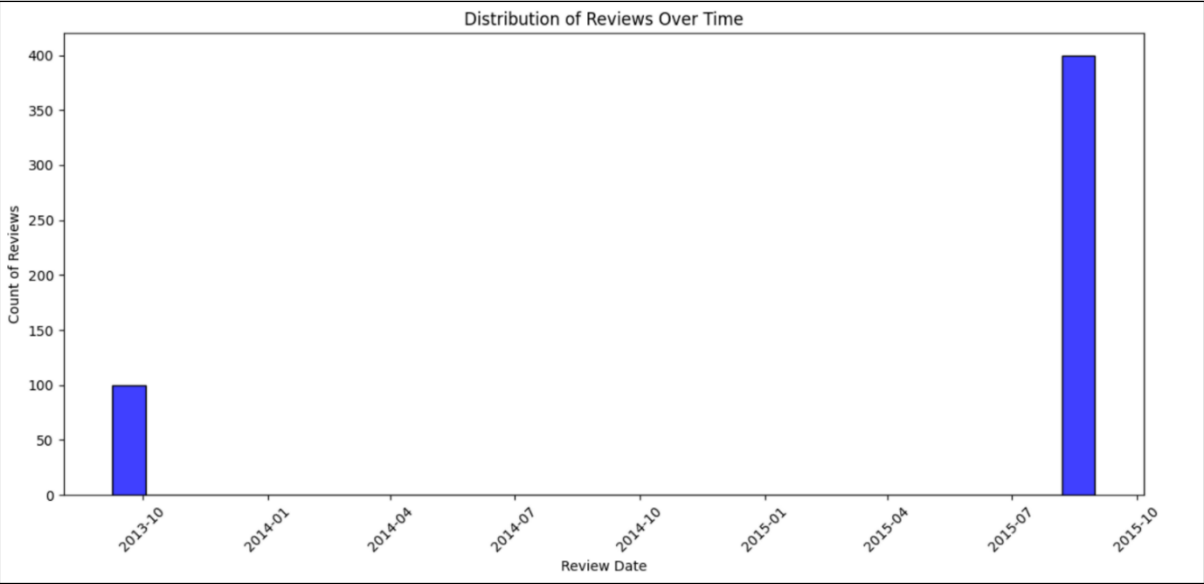
This notebook is requesting access to your Google Drive files. Granting access to Google Drive will permit code executed in the notebook to modify files in your Google Drive. Make sure to review notebook code prior to allowing this access.

No thanks

Connect to Google Drive

Mounted at /content/drive													
	market_place	customer_id	review_id	product_id	product_parent	product_title	product_category	star_rating	helpful_votes	total_votes	verified_purchase	review_headline	review_text
0	"US"	"42521656"	"R26MV8D0KG6QI6"	"B000SAQCWC"	"159713740"	"The Cravings Place Chocolate Chunk Cookie Mix..."	"Grocery"	1	0	0	1 \t(Y)	"Using these for years - love them."	"As all whee...
1	"US"	"12049833"	"R10F8GP57AQ1A0"	"B00509LVIQ"	"138680402"	"Mauna Loa Macadamias, 11 Ounce Packages"	"Grocery"	1	0	0	1 \t(Y)	"Wonderful"	"My nut. I crunch..."
2	"US"	"107642"	"R3VDC1QB6MC4ZZ"	"B00KHESL0"	"252021703"	"Organic Matcha Green Tea Powder - 100% Pure M..."	"Grocery"	1	0	0	0 \t(N)	"Five Stars"	"This tea te... griff..."
3	"US"	"6042304"	"R12FA3DCF8F9ER"	"B000F8JLIC"	"752728342"	"15oz Raspberry Lyons Designer Dessert Syrup S..."	"Grocery"	1	0	0	1 \t(Y)	"Five Stars"	"My brand is a gr..."
4	"US"	"18123821"	"RTWHVNV6X4CNJ"	"B004ZWR9RQ"	"552138758"	"Strice Spark Kinetic Fruit Sugar Free Gum, 14..."	"Grocery"	1	0	0	1 \t(Y)	"Five Stars"	





ChatGPT 4o ▾

Model ⓘ

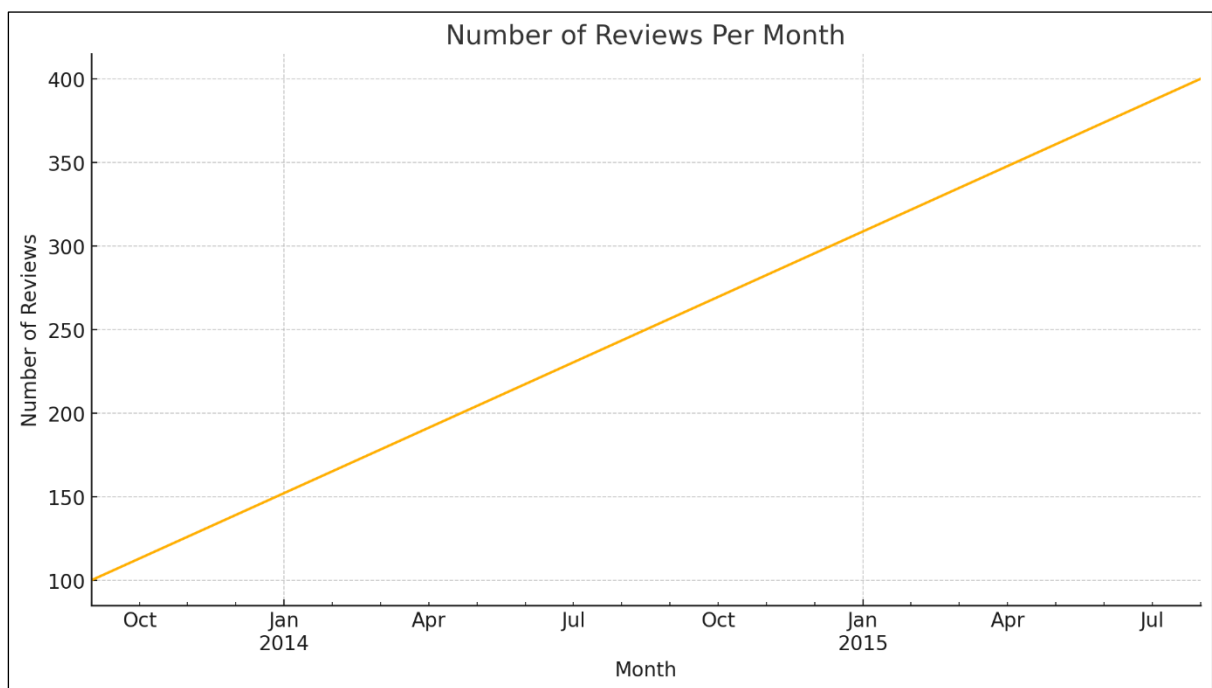
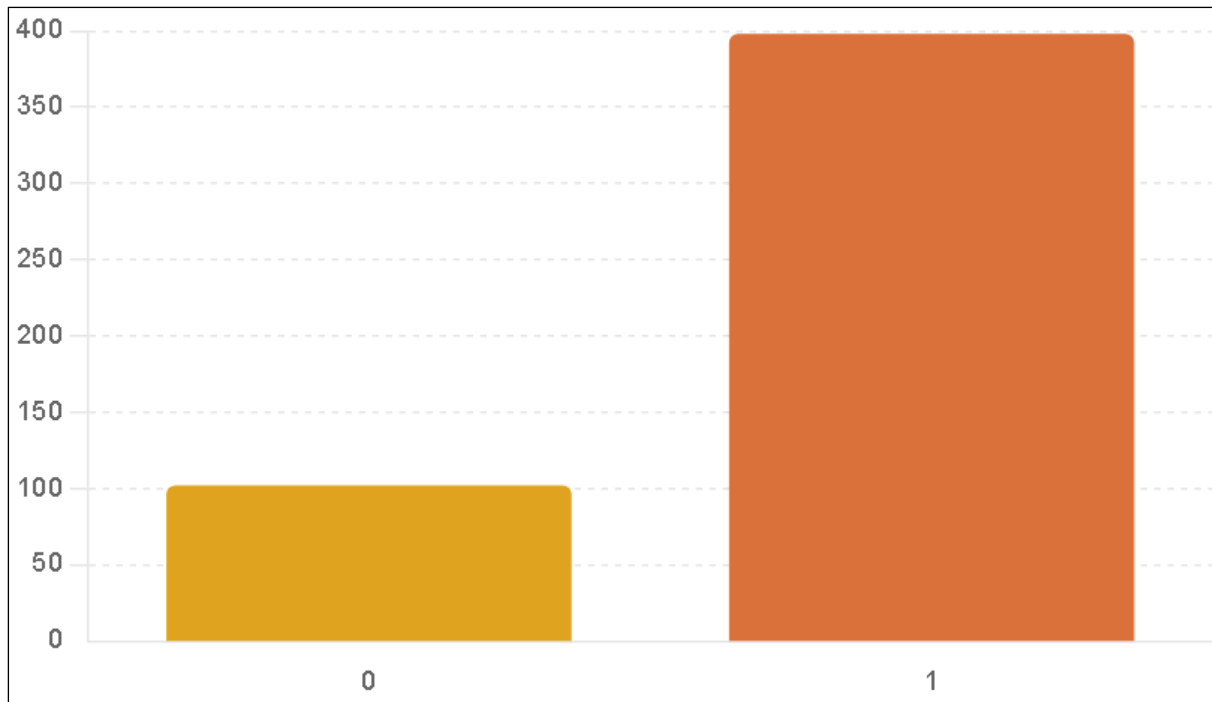
 **GPT-4o**
Newest and most advanced model

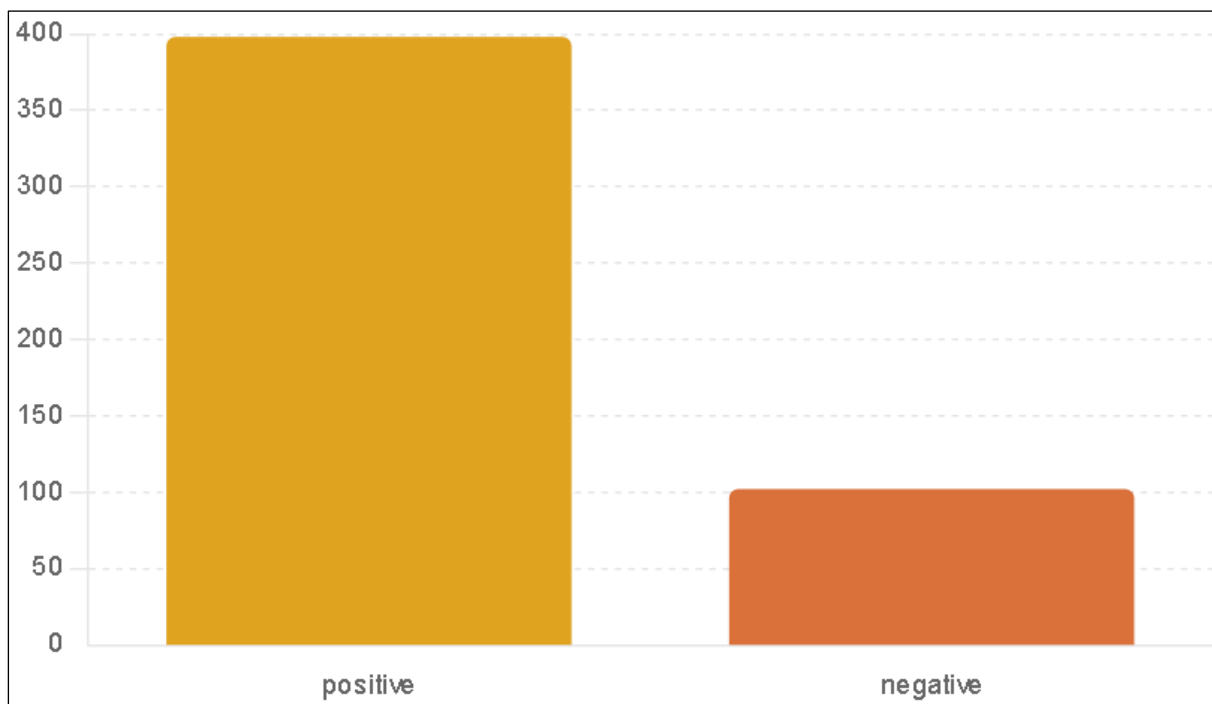
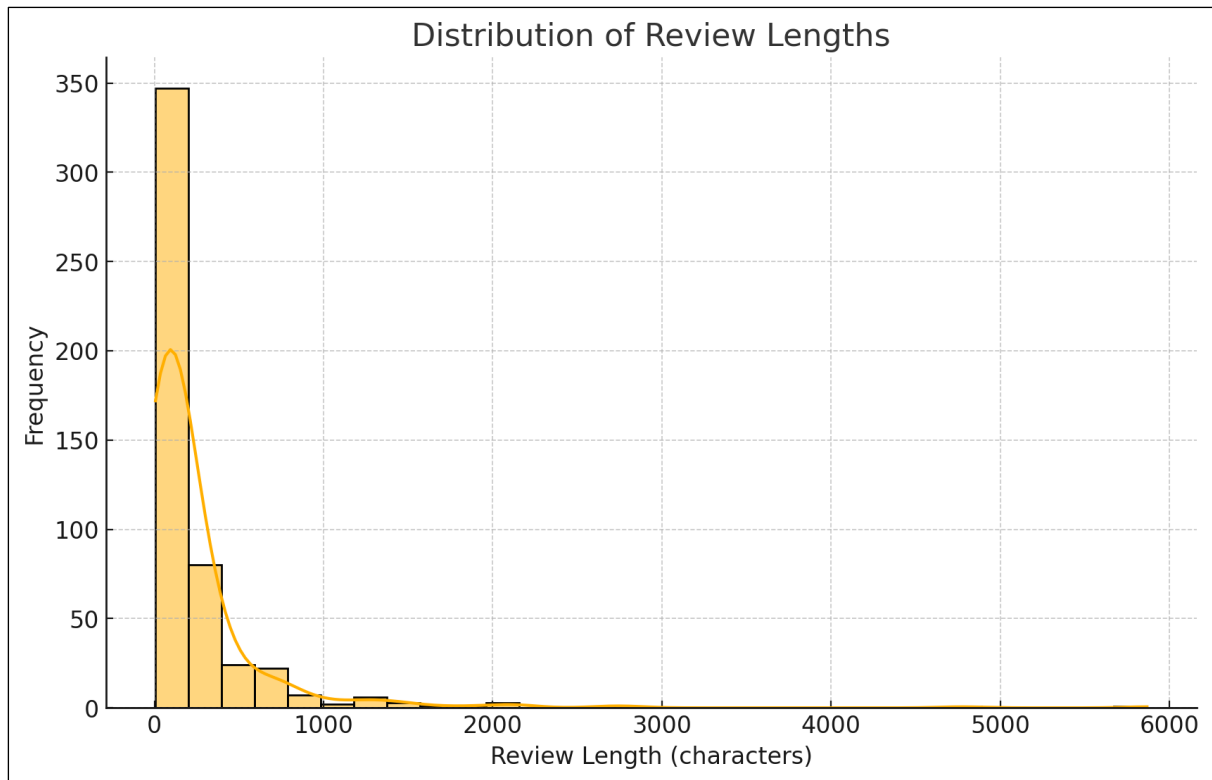
 **GPT-4**
Advanced model for complex tasks

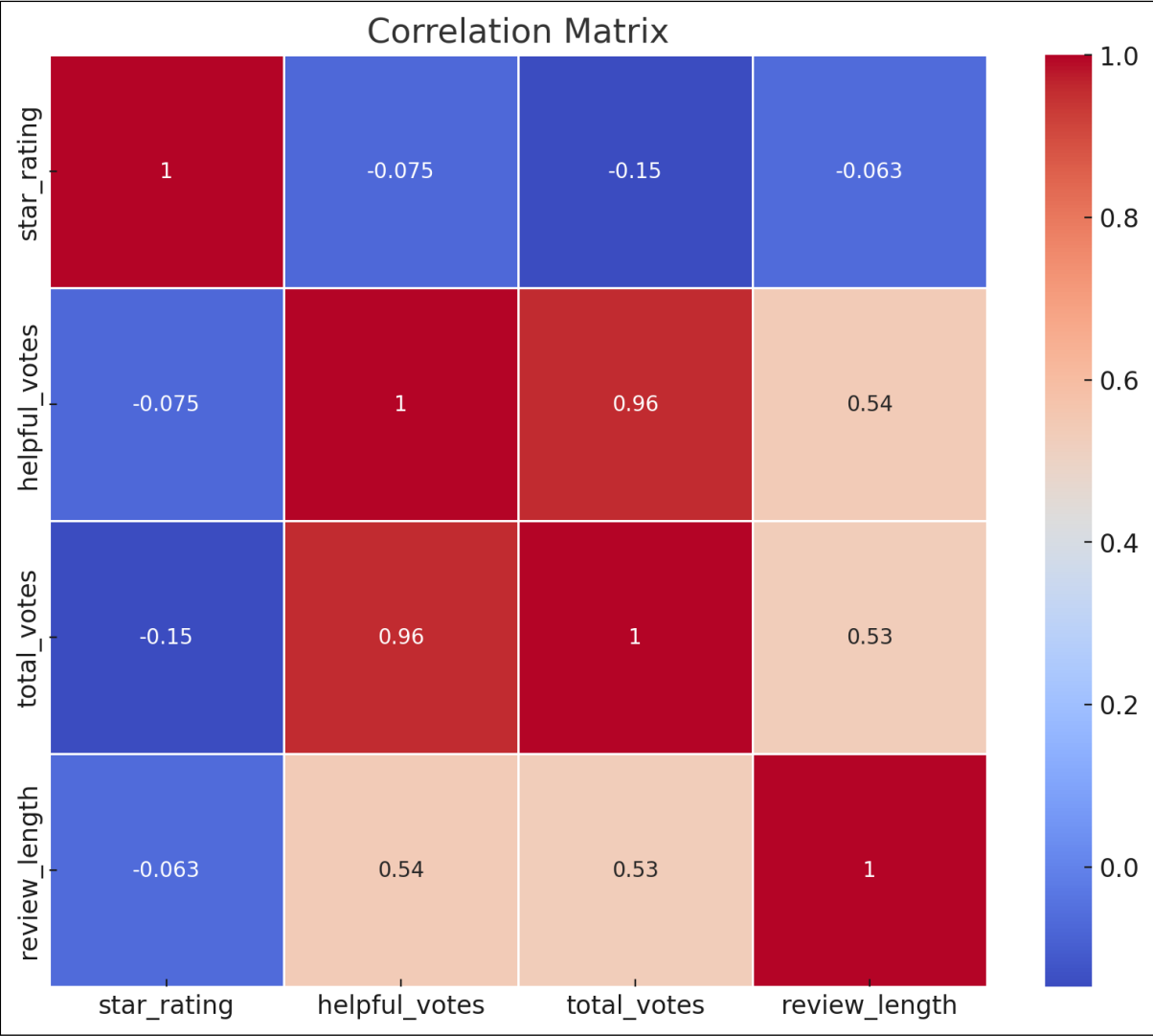
 **GPT-3.5**
Great for everyday tasks

 Temporary chat 

Summary Statistics				
☐		star_rating	helpful_votes	total_votes
1	count	500.0	500.0	500.0
2	mean	0.796	0.764	1.068
3	std	0.4033725558806342	2.8433207602892407	3.6984771821567137
4	min	0.0	0.0	0.0
5	25%	1.0	0.0	0.0
6	50%	1.0	0.0	0.0
7	75%	1.0	1.0	1.0
8	max	1.0	36.0	47.0







Chapter 12: Building a Classification Model with ChatGPT

```
✓ 2s ▶ 1 # Importing necessary libraries
      2 import pandas as pd
      3 from google.colab import drive
      4
      5 # Mount Google Drive
      6 drive.mount('/content/drive')
      7
      8 # Path to the dataset
      9 data_path = '/content/drive/My Drive/Chapter05/amazon_product_review_data.csv'
     10
     11 # Load the dataset into a DataFrame
     12 df = pd.read_csv(data_path)
```

Mounted at /content/drive

product_title	product_category	star_rating	helpful_votes	total_votes	verified_purchase	review_headline	review_body	review_date	sentiments	Cleaned_Review
"The Cravings Place Chocolate Chunk Cookie Mix..."	"Grocery"	1	0	0	1 \t(Y)	"Using these for years - love them."	"As a family allergic to wheat, dairy, eggs, n..."	2015-08-31	positive	" family allergic wheat dairy eggs nuts sever..."
"Mauna Loa Macadamias, 11 Ounce Packages"	"Grocery"	1	0	0	1 \t(Y)	"Wonderful"	"My favorite nut. Creamy, crunchy, salty, and ..."	2015-08-31	positive	" favorite nut creamy crunchy salty slightly ..."
"Organic Matcha Green Tea Powder - 100% Pure M..."	"Grocery"	1	0	0	0 \t(N)	"Five Stars"	"This green tea tastes so good! My girlfriend ..."	2015-08-31	positive	" green tea tastes good girlfriend loves ..."
"15oz Raspberry Lyons Designer Dessert Syrup S..."	"Grocery"	1	0	0	1 \t(Y)	"Five Stars"	"I love Melissa's brand but this is a great se..."	2015-08-31	positive	" love melissa's brand great second ca n't g..."
"Stride Spark Kinetic Fruit Sugar Free Gum, 14..."	"Grocery"	1	0	0	1 \t(Y)	"Five Stars"	"good"	2015-08-31	positive	" good ..."

Accuracy: 0.86

Classification Report:

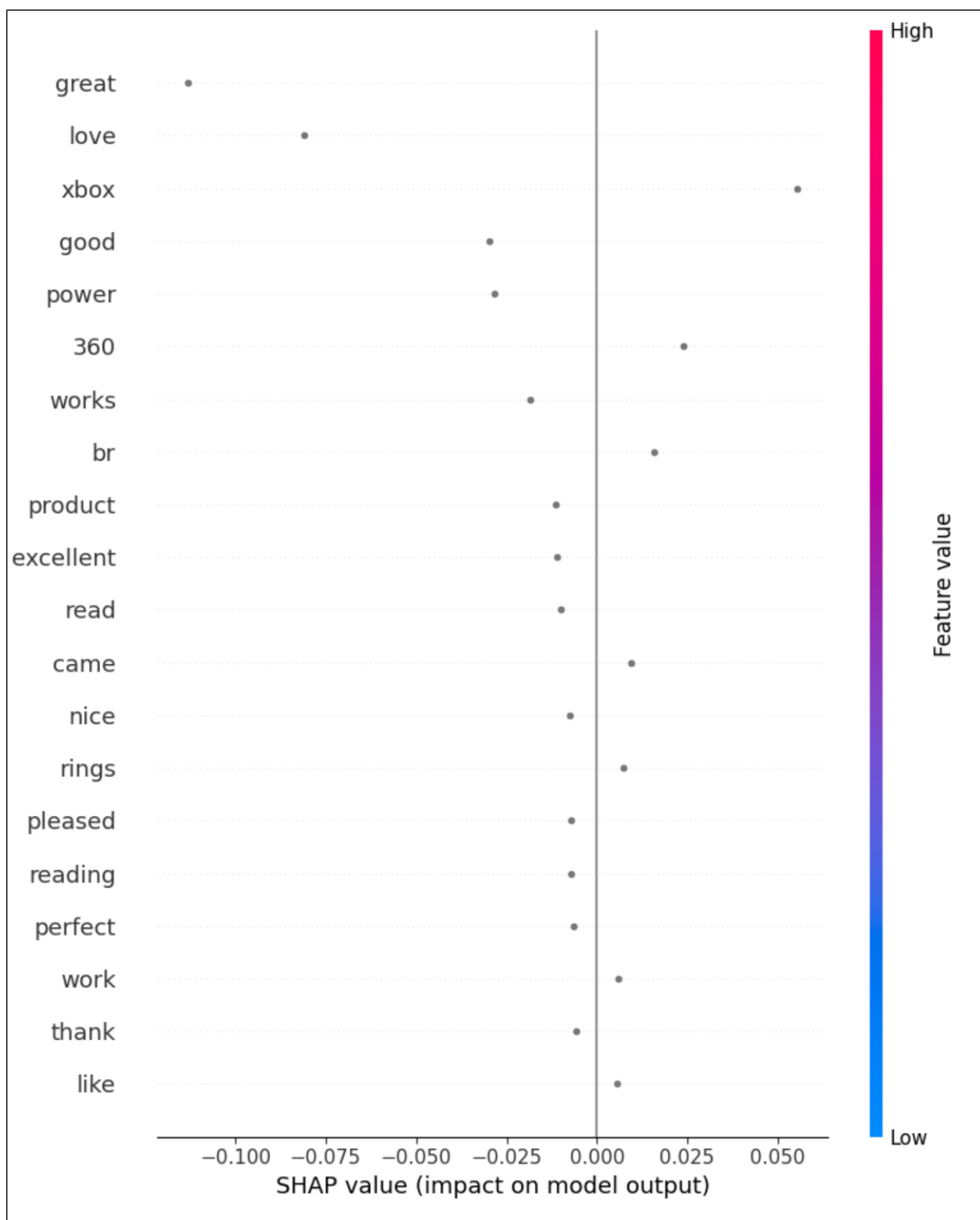
	precision	recall	f1-score	support
negative	0.00	0.00	0.00	14
positive	0.86	1.00	0.92	86
accuracy			0.86	100
macro avg	0.43	0.50	0.46	100
weighted avg	0.74	0.86	0.80	100

```
[ ] 1 !pip install shap
```

```
[6] 1 # Importing necessary libraries
2 import shap
3
4 # Initialize the SHAP explainer with the logistic regression model and training data
5 explainer = shap.Explainer(lr_model, X_train_tfidf)
6
7 # Explain the model's predictions for a specific instance (e.g., the first instance in the test set)
8 instance_index = 0 # You can choose any instance index from the test set
9 shap_values = explainer.shap_values(X_test_tfidf[instance_index])
10
11 # Visualize the SHAP values
12 shap.summary_plot(shap_values, features=X_test_tfidf[instance_index], feature_names=tfidf_vectorizer.get_feature_names())
```

```
-----
AttributeError                                Traceback (most recent call last)
<ipython-input-6-673ab3e1b84b> in <cell line: 12>()
    10
    11 # Visualize the SHAP values
--> 12 shap.summary_plot(shap_values, features=X_test_tfidf[instance_index], feature_names=tfidf_vectorizer.get_feature_names())

AttributeError: 'TfidfVectorizer' object has no attribute 'get_feature_names'
```



y=positive top features

Weight?	Feature
+1.799	x1269
+1.660	x1674
+1.176	<BIAS>
+0.969	x1249
+0.812	x3146
+0.739	x993
+0.711	x2041
+0.676	x706
... 2027 more positive ...	
... 1138 more negative ...	
-0.584	x2340
-0.592	x365
-0.625	x2342
-0.626	x2600
-0.633	x1286
-0.633	x1426
-0.658	x248
-0.682	x364
-0.764	x1005
-0.860	x488
-0.900	x1930
-0.943	x1929

y=positive top features

Weight?	Feature
---------	---------

+1.799	great
--------	-------

+1.660	love
--------	------

+1.176	<BIAS>
--------	--------

+0.969	good
--------	------

+0.812	works
--------	-------

+0.739	excellent
--------	-----------

+0.711	perfect
--------	---------

+0.676	cute
--------	------

... 2027 more positive ...

... 1138 more negative ...

-0.584	right
--------	-------

-0.592	bracelet
--------	----------

-0.625	rings
--------	-------

-0.626	sort
--------	------

-0.633	gum
--------	-----

-0.633	impressive
--------	------------

-0.658	back
--------	------

-0.682	br
--------	----

-0.764	expected
--------	----------

-0.860	cheap
--------	-------

-0.900	okay
--------	------

-0.943	ok
--------	----



Accuracy with balanced class weights: 0.81

Classification Report with balanced class weights:

	precision	recall	f1-score	support
negative	0.27	0.21	0.24	14
positive	0.88	0.91	0.89	86
accuracy			0.81	100
macro avg	0.57	0.56	0.57	100
weighted avg	0.79	0.81	0.80	100

Best Hyperparameters: {'C': 0.01, 'solver': 'liblinear'}
Accuracy with tuned model: 0.86

Classification Report with tuned model:

	precision	recall	f1-score	support
negative	0.00	0.00	0.00	14
positive	0.86	1.00	0.92	86
accuracy			0.86	100
macro avg	0.43	0.50	0.46	100
weighted avg	0.74	0.86	0.80	100

```
1 from sklearn.feature_extraction.text import CountVectorizer
2
3 # Initialize CountVectorizer
4 bow_vectorizer = CountVectorizer(max_features=5000) # You can adjust max_features as needed
5
6 # Transform text data into bag-of-words representation
7 X_train_bow = bow_vectorizer.fit_transform(X_train)
8 X_test_bow = bow_vectorizer.transform(X_test)
9
10 # Train logistic regression model with bag-of-words representation
11 lr_model_bow = LogisticRegression()
12 lr_model_bow.fit(X_train_bow, y_train)
13
14 # Evaluate model performance
15 y_pred_bow = lr_model_bow.predict(X_test_bow)
16 accuracy_bow = accuracy_score(y_test, y_pred_bow)
17 print("Accuracy with bag-of-words representation:", accuracy_bow)
18 # Add other evaluation metrics if needed
19
```

Accuracy with bag-of-words representation: 0.85

```

1  from sklearn.feature_extraction.text import TfidfVectorizer
2
3  # Initialize TfidfVectorizer
4  tfidf_vectorizer = TfidfVectorizer(max_features=5000) # You can adjust max_features as needed
5
6  # Transform text data into TF-IDF representation
7  X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
8  X_test_tfidf = tfidf_vectorizer.transform(X_test)
9
10 # Train logistic regression model with TF-IDF representation
11 lr_model_tfidf = LogisticRegression()
12 lr_model_tfidf.fit(X_train_tfidf, y_train)
13
14 # Evaluate model performance
15 y_pred_tfidf = lr_model_tfidf.predict(X_test_tfidf)
16 accuracy_tfidf = accuracy_score(y_test, y_pred_tfidf)
17 print("Accuracy with TF-IDF representation:", accuracy_tfidf)
18 # Add other evaluation metrics if needed
19

```

Accuracy with TF-IDF representation: 0.86

```

1  # Assuming you have pre-trained word embeddings loaded (e.g., GloVe)
2  # Convert text data into word embeddings representation
3  # You may need to tokenize and pad sequences before applying word embeddings
4
5  # Train logistic regression model with word embeddings representation
6  lr_model_word_embeddings = LogisticRegression()
7  lr_model_word_embeddings.fit(X_train_word_embeddings, y_train)
8
9  # Evaluate model performance
10 y_pred_word_embeddings = lr_model_word_embeddings.predict(X_test_word_embeddings)
11 accuracy_word_embeddings = accuracy_score(y_test, y_pred_word_embeddings)
12 print("Accuracy with word embeddings representation:", accuracy_word_embeddings)
13 # Add other evaluation metrics if needed
14

```

```

-----
NameError                                Traceback (most recent call last)
<ipython-input-20-8a483831a6f9> in <cell line: 7>()
      5 # Train logistic regression model with word embeddings representation
      6 lr_model_word_embeddings = LogisticRegression()
----> 7 lr_model_word_embeddings.fit(X_train_word_embeddings, y_train)
      8
      9 # Evaluate model performance

NameError: name 'X_train_word_embeddings' is not defined

```

```

43
44 # Convert tokenized text data into word embeddings representation
45 X_train_word_embeddings = WordEmbeddingsTransformer(w2v_model).fit_transform(TokenizerTransformer().fit_transform(X_train))
46 X_test_word_embeddings = WordEmbeddingsTransformer(w2v_model).transform(TokenizerTransformer().transform(X_test))
47
48 # Train logistic regression model with word embeddings representation
49 lr_model_word_embeddings = LogisticRegression()
50 lr_model_word_embeddings.fit(X_train_word_embeddings, y_train)
51
52 # Evaluate model performance
53 y_pred_word_embeddings = lr_model_word_embeddings.predict(X_test_word_embeddings)
54 accuracy_word_embeddings = accuracy_score(y_test, y_pred_word_embeddings)
55 print("Accuracy with word embeddings representation (Word2Vec):", accuracy_word_embeddings)
56 # Add other evaluation metrics if needed
57

```

Accuracy with word embeddings representation (Word2Vec): 0.86

Chapter 13: Building a Regression Model for Customer Spend with ChatGPT

ChatGPT

Sure, let's proceed step by step. First, we need to mount Google Drive in our Jupyter Notebook environment so we can access the dataset.

Here's how you can do it:

```
python Copy code  
  
from google.colab import drive  
  
# Mount Google Drive  
drive.mount('/content/drive')
```

Explanation:

- `from google.colab import drive`: This line imports the `drive` module from the `google.colab` package.
- `drive.mount('/content/drive')`: This command mounts your Google Drive at the specified path `/content/drive`. When you run this cell, it will prompt you to authenticate and give you a link to follow to get an authorization code. Once you enter the code, your Google Drive will be mounted.

Let me know if you have any questions before we proceed.



```
[1] 1  from google.colab import drive  
    2  # Mount Google Drive  
    3  drive.mount('/content/drive')
```

Mounted at /content/drive

```
[2] 1 import pandas as pd
2
3 # Define the file path
4 file_path = '/content/drive/My Drive/Chapter05/Ecommerce Customers.csv'
5
6 # Load the dataset into a DataFrame
7 ecommerce_data = pd.read_csv(file_path)
8
9 # Display the first few rows of the DataFrame
10 print(ecommerce_data.head())
```

```

      Email \
0  mstephenson@fernandez.com
1  hduke@hotmail.com
2  pallen@yahoo.com
3  riverarebecca@gmail.com
4  mstephens@daavidson-herman.com

      Address      Avatar \
0  835 Frank Tunnel\nWrightmouth, MI 82180-9605  Violet
1  4547 Archer Common\nDiazchester, CA 06566-8576  DarkGreen
2  24645 Valerie Unions Suite 582\nCobbborough, D...  Bisque
3  1414 David Throughway\nPort Jason, OH 22070-1220  SaddleBrown
4  14023 Rodriguez Passage\nPort Jacobville, PR 3...  MediumAquaMarine

      Avg. Session Length  Time on App  Time on Website  Length of Membership \
0          34.497268      12.655651      39.577668          4.082621
1          31.926272      11.109461      37.268959          2.664034
2          33.000915      11.330278      37.110597          4.104543
3          34.305557      13.717514      36.721283          3.120179
4          33.330673      12.795189      37.536653          4.446308

      Yearly Amount Spent
0          587.951054
1          392.204933
2          487.547505
3          581.852344
4          599.406092
```

```
[ ] 1 # Extract input features (X) and target variable (y)
2 X = ecommerce_data[['Avg. Session Length', 'Time on App', 'Time on Website', 'Length of Membership']]
3 y = ecommerce_data['Yearly Amount Spent']
```

```
[ ] 1 from sklearn.model_selection import train_test_split
2
3 # Split the data into training and testing sets (80% train, 20% test)
4 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
5
6 # Display the shapes of the training and testing sets
7 print("Training set shape - X_train:", X_train.shape, "y_train:", y_train.shape)
8 print("Testing set shape - X_test:", X_test.shape, "y_test:", y_test.shape)
```

```
Training set shape - X_train: (400, 4) y_train: (400,)
Testing set shape - X_test: (100, 4) y_test: (100,)
```

```
[ ] 1 from sklearn.linear_model import LinearRegression
2
3 # Initialize the linear regression model
4 model = LinearRegression()
```

```
[ ] 1 # Train the linear regression model using the training data
    2 model.fit(X_train, y_train)
```

LinearRegression
LinearRegression()

```
[ ] 1 # Make predictions on the testing data
    2 y_pred = model.predict(X_test)
    3
    4 # Evaluate the model's performance
    5 from sklearn.metrics import mean_squared_error, r2_score
    6
    7 # Calculate Mean Squared Error (MSE)
    8 mse = mean_squared_error(y_test, y_pred)
    9
   10 # Calculate R-squared (coefficient of determination)
   11 r_squared = r2_score(y_test, y_pred)
   12
   13 # Print the evaluation metrics
   14 print("Mean Squared Error (MSE):", mse)
   15 print("R-squared (Coefficient of Determination):", r_squared)
```

Mean Squared Error (MSE): 109.8637411839399
R-squared (Coefficient of Determination): 0.9778130629184127

```
[ ] 1 from sklearn.linear_model import RidgeCV
    2
    3 # Initialize the Ridge regression model with cross-validation
    4 ridge_model = RidgeCV(alphas=[0.1, 1.0, 10.0], cv=5)
    5
    6 # Train the Ridge regression model
    7 ridge_model.fit(X_train, y_train)
    8
    9 # Make predictions on the testing data
   10 ridge_y_pred = ridge_model.predict(X_test)
   11
   12 # Evaluate the model's performance
   13 ridge_mse = mean_squared_error(y_test, ridge_y_pred)
   14 ridge_r_squared = ridge_model.score(X_test, y_test)
   15
   16 # Print the evaluation metrics
   17 print("Ridge Regression - Mean Squared Error (MSE):", ridge_mse)
   18 print("Ridge Regression - R-squared (Coefficient of Determination):", ridge_r_squared)
   19 print("Ridge Regression - Best Alpha:", ridge_model.alpha_)
   20
```

Ridge Regression - Mean Squared Error (MSE): 109.82024999312038
Ridge Regression - R-squared (Coefficient of Determination): 0.9778218459464064
Ridge Regression - Best Alpha: 0.1

```
[ ] 1 from sklearn.linear_model import LassoCV
2
3 # Initialize the Lasso regression model with cross-validation
4 lasso_model = LassoCV(alphas=[0.1, 1.0, 10.0], cv=5)
5
6 # Train the Lasso regression model
7 lasso_model.fit(X_train, y_train)
8
9 # Make predictions on the testing data
10 lasso_y_pred = lasso_model.predict(X_test)
11
12 # Evaluate the model's performance
13 lasso_mse = mean_squared_error(y_test, lasso_y_pred)
14 lasso_r_squared = lasso_model.score(X_test, y_test)
15
16 # Print the evaluation metrics
17 print("Lasso Regression - Mean Squared Error (MSE):", lasso_mse)
18 print("Lasso Regression - R-squared (Coefficient of Determination):", lasso_r_squared)
19 print("Lasso Regression - Best Alpha:", lasso_model.alpha_)
```

```
Lasso Regression - Mean Squared Error (MSE): 109.77575439252612
Lasso Regression - R-squared (Coefficient of Determination): 0.9778308318145387
Lasso Regression - Best Alpha: 0.1
```

```
[ ] 1 import numpy as np
2 import pandas as pd
3
4 # Set random seed for reproducibility
5 np.random.seed(42)
6
7 # Number of samples
8 n_samples = 500
9
10 # Generate synthetic input features
11 avg_session_length = np.random.normal(loc=30, scale=5, size=n_samples)
12 time_on_app = np.random.normal(loc=15, scale=5, size=n_samples)
13 time_on_website = np.random.normal(loc=20, scale=7, size=n_samples)
14 length_of_membership = np.random.normal(loc=3, scale=1, size=n_samples)
15
16 # Generate synthetic target variable (linear combination of input features with noise)
17 yearly_amount_spent = 100 + (avg_session_length * 10) + (time_on_app * 20) + (time_on_website * 15) + (length_of_membership * 30) + np.random.normal(loc=0, scale=50, size=n_samples)
18
19 # Create a DataFrame from the synthetic dataset
20 synthetic_data = pd.DataFrame({
21     'Avg. Session Length': avg_session_length,
22     'Time on App': time_on_app,
23     'Time on Website': time_on_website,
24     'Length of Membership': length_of_membership,
25     'Yearly Amount Spent': yearly_amount_spent
26 })
27
28 # Display the first few rows of the synthetic dataset
29 print(synthetic_data.head())
30
31 # Save the synthetic dataset to a CSV file
32 synthetic_data.to_csv('synthetic_ecommerce_data.csv', index=False)
33
```

	Avg. Session Length	Time on App	Time on Website	Length of Membership \
0	22.482571	19.630868	29.795488	2.778361
1	29.308678	24.547083	26.472436	2.448814
2	33.239443	6.007162	20.417413	2.181801
3	37.615149	17.814846	15.471443	2.996626
4	28.829233	11.746787	24.887563	2.822915

	Yearly Amount Spent
0	1343.977702
1	1347.353481
2	924.621896
3	1139.820745
4	986.755258

```
[ ] 1 import pandas as pd
2 from sklearn.model_selection import train_test_split
3 from sklearn.linear_model import LinearRegression
4 from sklearn.metrics import mean_squared_error, r2_score
5
6 # Load the synthetic dataset
7 synthetic_data = pd.read_csv('synthetic_ecommerce_data.csv')
8
9 # Extract input features (X) and target variable (y)
10 X = synthetic_data[['Avg. Session Length', 'Time on App', 'Time on Website', 'Length of Membership']]
11 y = synthetic_data['Yearly Amount Spent']
12
13 # Split the dataset into training and testing sets (80% train, 20% test)
14 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
15
16 # Initialize the linear regression model
17 model = LinearRegression()
18
19 # Train the linear regression model using the training data
20 model.fit(X_train, y_train)
21
22 # Make predictions on the testing data
23 y_pred = model.predict(X_test)
24
25 # Evaluate the model's performance
26 mse = mean_squared_error(y_test, y_pred)
27 r_squared = r2_score(y_test, y_pred)
28
29 # Print the evaluation metrics
30 print("Mean Squared Error (MSE):", mse)
31 print("R-squared (Coefficient of Determination):", r_squared)
```

Mean Squared Error (MSE): 2103.188278720931
R-squared (Coefficient of Determination): 0.9196954387778045

ChatGPT

 Error analyzing ▾

It seems there was an internal issue when trying to run the code for Lasso and Ridge regression. Unfortunately, I cannot provide a specific stack trace or error details because the issue seems to be internal to the system. However, I can help troubleshoot common causes for issues in regression models with cross-validation:

1. **Data Scaling:** Both Lasso and Ridge regression can be sensitive to the scale of the input features. It's often recommended to standardize the data before applying these methods.
2. **Choice of Alphas:** The choice of alpha values (regularization strength) might not be suitable for the data. It can be helpful to test a wider range of values or use a logarithmic scale for choosing alphas.
3. **Sparse Data:** If your data is sparse, certain solvers used by Lasso and Ridge might not perform optimally. Ensuring the data format and solver compatibility is crucial.
4. **Library Issues:** Occasionally, issues might arise from the underlying libraries or their dependencies. Ensuring that all libraries are up to date can help mitigate these problems.

Would you like me to adjust the code by including data scaling, try a different range of alpha values, or check for other potential issues? Alternatively, I can attempt to run the code again to see if the issue persists. [\[↩\]](#)

```

1] 1 # Import necessary libraries
2   import pandas as pd
3   from sklearn.model_selection import train_test_split
4   from sklearn.linear_model import LinearRegression, LassoCV, RidgeCV
5   from sklearn.metrics import mean_absolute_error, mean_squared_error
6   from sklearn.preprocessing import StandardScaler
7
8   # Mount Google Drive (specific to Google Colab)
9   from google.colab import drive
10  drive.mount('/content/drive')
11
12  # Load the dataset from Google Drive
13  file_path = '/content/drive/My Drive/Chapter05/Ecommerce Customers.csv'
14  data = pd.read_csv(file_path)
15
16  # Select only the numerical columns for features and the target column
17  features = data[['Avg. Session Length', 'Time on App', 'Time on Website', 'Length of Membership']]
18  target = data['Yearly Amount Spent']
19
20  # Splitting the dataset into training and testing sets (80% train, 20% test)
21  X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
22
23  # Standardizing the data
24  scaler = StandardScaler()
25  X_train_scaled = scaler.fit_transform(X_train)
26  X_test_scaled = scaler.transform(X_test)
27
28  # Initialize and fit the Linear Regression model
29  model = LinearRegression()
30  model.fit(X_train_scaled, y_train)
31
32  # Lasso Regression with Cross-Validation
33  lasso_cv = LassoCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)
34  lasso_cv.fit(X_train_scaled, y_train)
35  lasso_cv_score = lasso_cv.score(X_test_scaled, y_test)
36  lasso_cv_mae = mean_absolute_error(y_test, lasso_cv.predict(X_test_scaled))
37
38  # Ridge Regression with Cross-Validation
39  ridge_cv = RidgeCV(alphas=[0.01, 0.1, 1.0, 10.0], cv=5)
40  ridge_cv.fit(X_train_scaled, y_train)
41  ridge_cv_score = ridge_cv.score(X_test_scaled, y_test)
42  ridge_cv_mae = mean_absolute_error(y_test, ridge_cv.predict(X_test_scaled))
43
44  # Print results
45  print(f'Linear Regression Test MAE: {mean_absolute_error(y_test, model.predict(X_test_scaled))}')
46  print(f'Lasso CV Score: {lasso_cv_score}, MAE: {lasso_cv_mae}')
47  print(f'Ridge CV Score: {ridge_cv_score}, MAE: {ridge_cv_mae}')
48
Mounted at /content/drive
Linear Regression Test MAE: 8.558441885315247
Lasso CV Score: 0.9778155618342972, MAE: 8.557805798317295
Ridge CV Score: 0.9778139996945001, MAE: 8.558241935489011

```

 Talking to Data Analyst

 Message Data Analyst

ChatGPT can make mistakes. Consider checking important information.

Chapter 14: Building an MLP Model for Fashion-MNIST with ChatGPT



e + Text

```
19 # Step 3: Flatten images into vectors
20 def flatten_images(train_data, test_data):
21     # Reshape images to vectors
22     train_data = train_data.reshape(train_data.shape[0], -1)
23     test_data = test_data.reshape(test_data.shape[0], -1)
24     return train_data, test_data
25
26 # Step 4: Encode categorical labels
27 def encode_labels(train_labels, test_labels):
28     # Initialize LabelEncoder
29     label_encoder = LabelEncoder()
30     # Fit LabelEncoder on training labels and transform both training and testing labels
31     train_labels = label_encoder.fit_transform(train_labels)
32     test_labels = label_encoder.transform(test_labels)
33     return train_labels, test_labels
34
35 # Apply data pre-processing steps
36 x_train, x_test = normalize_images(x_train, x_test)
37 x_train, x_test = flatten_images(x_train, x_test)
38 y_train, y_test = encode_labels(y_train, y_test)
39
40 # Print the shapes of pre-processed data
41 print("x_train shape:", x_train.shape)
42 print("y_train shape:", y_train.shape)
43 print("x_test shape:", x_test.shape)
44 print("y_test shape:", y_test.shape)
45
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz>
29515/29515 [=====] - 0s 0us/step
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz>
26421880/26421880 [=====] - 1s 0us/step
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz>
5148/5148 [=====] - 0s 0us/step
Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz>
4422102/4422102 [=====] - 0s 0us/step
x_train shape: (60000, 784)
y_train shape: (60000,)
x_test shape: (10000, 784)
y_test shape: (10000,)

```
1 # Step 5: Split dataset into training, validation, and testing sets
2 def split_dataset(train_data, train_labels, test_data, test_labels, validation_size=0.1, test_size=0.1, random_state=42):
3     # Split training set into training and validation sets
4     x_train, x_val, y_train, y_val = train_test_split(train_data, train_labels, test_size=validation_size, random_state=random_state)
5     # Split testing set into testing and validation sets
6     x_test, x_val, y_test, y_val = train_test_split(test_data, test_labels, test_size=test_size, random_state=random_state)
7     return x_train, y_train, x_val, y_val, x_test, y_test
8
9 # Apply data splitting
10 x_train, y_train, x_val, y_val, x_test, y_test = split_dataset(x_train, y_train, x_test, y_test)
11
12 # Print the shapes of split datasets
13 print("Training set:")
14 print("x_train shape:", x_train.shape)
15 print("y_train shape:", y_train.shape)
16 print("\nValidation set:")
17 print("x_val shape:", x_val.shape)
18 print("y_val shape:", y_val.shape)
19 print("\nTesting set:")
20 print("x_test shape:", x_test.shape)
21 print("y_test shape:", y_test.shape)
22
```

Training set:
x_train shape: (54000, 784)
y_train shape: (54000,)

Validation set:
x_val shape: (1000, 784)
y_val shape: (1000,)

Testing set:
x_test shape: (9000, 784)
y_test shape: (9000,)

```

1  from tensorflow.keras.models import Sequential
2  from tensorflow.keras.layers import Dense
3
4  # Step 6: Define a single-layer MLP model
5  def create_single_layer_mlp(input_shape, num_classes):
6      model = Sequential()
7      # Add a single hidden layer with ReLU activation
8      model.add(Dense(128, activation='relu', input_shape=input_shape))
9      # Add output layer with softmax activation for multi-class classification
10     model.add(Dense(num_classes, activation='softmax'))
11     return model
12
13 # Define input shape and number of classes
14 input_shape = x_train.shape[1:]
15 num_classes = len(np.unique(y_train))
16
17 # Create the single-layer MLP model
18 model = create_single_layer_mlp(input_shape, num_classes)
19
20 # Print model summary
21 model.summary()
22

```

Model: "sequential"

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 128)	100480
dense_1 (Dense)	(None, 10)	1290
Total params: 101770 (397.54 KB)		
Trainable params: 101770 (397.54 KB)		
Non-trainable params: 0 (0.00 Byte)		

```
[4] 1 from tensorflow.keras.utils import to_categorical
2
3 # Step 7: Model Training
4 def train_model(model, x_train, y_train, x_val, y_val, batch_size=32, epochs=10):
5     # Convert class vectors to binary class matrices (one-hot encoding)
6     y_train = to_categorical(y_train)
7     y_val = to_categorical(y_val)
8     # Compile the model with categorical cross-entropy loss and Adam optimizer
9     model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
10    # Train the model on the training data
11    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_data=(x_val, y_val))
12    return history
13
14 # Train the model
15 history = train_model(model, x_train, y_train, x_val, y_val, batch_size=32, epochs=10)
16
```

Epoch 1/10
1688/1688 [=====] - 6s 3ms/step - loss: 0.5038 - accuracy: 0.8231 - val_loss: 0.4535 - val_accuracy: 0.8210
Epoch 2/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.3781 - accuracy: 0.8646 - val_loss: 0.3870 - val_accuracy: 0.8550
Epoch 3/10
1688/1688 [=====] - 6s 3ms/step - loss: 0.3392 - accuracy: 0.8765 - val_loss: 0.3674 - val_accuracy: 0.8690
Epoch 4/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.3127 - accuracy: 0.8854 - val_loss: 0.3454 - val_accuracy: 0.8780
Epoch 5/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.2960 - accuracy: 0.8910 - val_loss: 0.3372 - val_accuracy: 0.8680
Epoch 6/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.2803 - accuracy: 0.8958 - val_loss: 0.3497 - val_accuracy: 0.8720
Epoch 7/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.2666 - accuracy: 0.8997 - val_loss: 0.3657 - val_accuracy: 0.8650
Epoch 8/10
1688/1688 [=====] - 6s 3ms/step - loss: 0.2554 - accuracy: 0.9039 - val_loss: 0.3464 - val_accuracy: 0.8700
Epoch 9/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.2459 - accuracy: 0.9079 - val_loss: 0.3492 - val_accuracy: 0.8700
Epoch 10/10
1688/1688 [=====] - 5s 3ms/step - loss: 0.2382 - accuracy: 0.9106 - val_loss: 0.3363 - val_accuracy: 0.8800

```
282/282 [=====] - 2s 5ms/step
-----
AxisError                                Traceback (most recent call last)
<ipython-input-5-02193580c831> in <cell line: 23>()
    21
    22 # Evaluate the model
--> 23 accuracy, precision, recall, cm = evaluate_model(model, x_test, y_test)
    24
    25 # Print evaluation metrics

2 frames
/usr/local/lib/python3.10/dist-packages/numpy/core/fromnumeric.py in _wrapfunc(obj, method, *args, **kws)
    57
    58     try:
--> 59         return bound(*args, **kws)
    60     except TypeError:
    61         # A TypeError occurs if the object does have such a method in its

AxisError: axis 1 is out of bounds for array of dimension 1
```

✓
0s



```
1 print("Shape of y_test:", y_test.shape)
```

```
Shape of y_test: (9000,)
```

282/282 [=====] - 1s 3ms/step

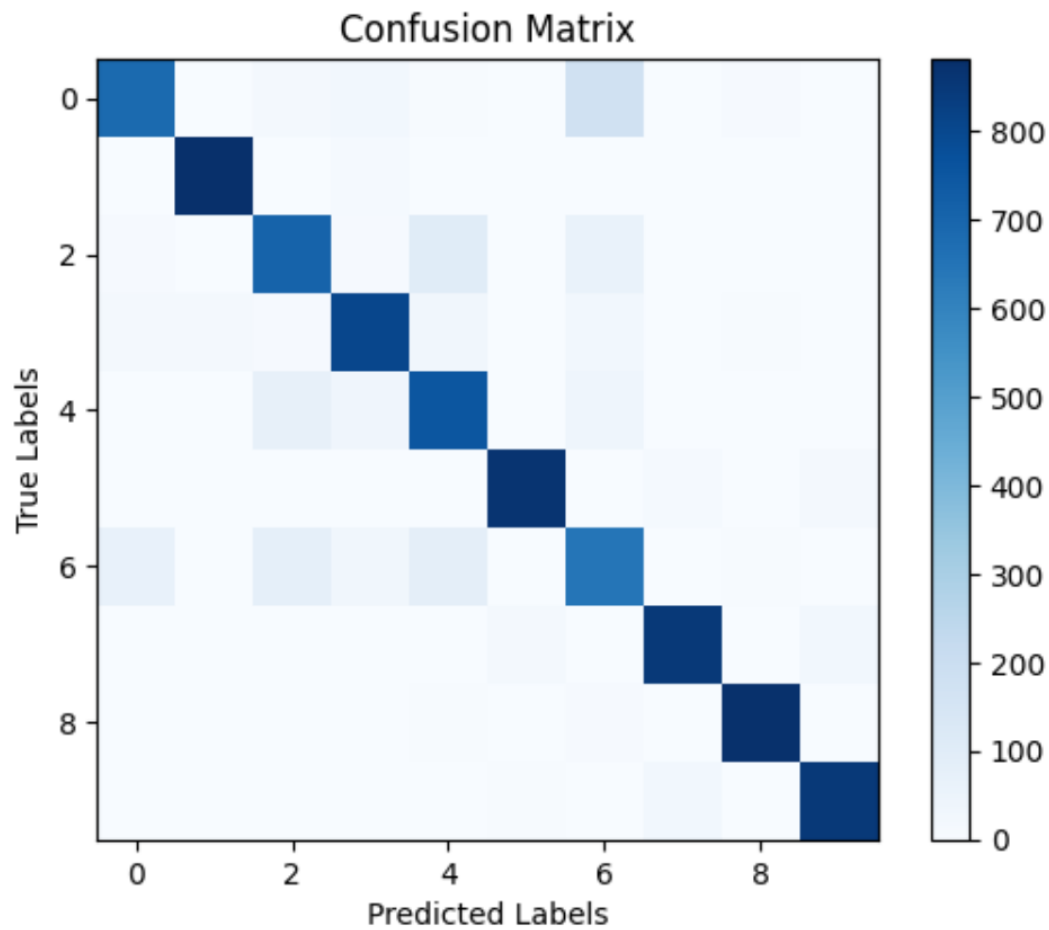
Accuracy: 0.8795555555555555

Precision: 0.8817019102604995

Recall: 0.8795555555555555

Confusion Matrix:

```
[[683  2 15 25  5  3 175  0  8  0]
 [  0 881  1 12  3  0  3  0  1  0]
 [  7  1 707  9 103  1  60  0  1  0]
 [ 15 15 10 804 28  1 27  0  6  0]
 [  0  3 72 37 753  0 40  0  1  0]
 [  0  0  0  0  0 866  0 11  0 17]
 [ 63  1 79 29 83  0 646  0  6  0]
 [  0  0  0  0  0 20  0 850  0 25]
 [  3  0  2  3  5  3  8  3 877  0]
 [  0  0  0  0  0  5  1 27  0 849]]
```



```

1 # Evaluate two-layer MLP model
2 two_layer_accuracy, _, _, _ = evaluate_model(two_layer_model, x_test, y_test)
3
4 # Evaluate three-layer MLP model
5 three_layer_accuracy, _, _, _ = evaluate_model(three_layer_model, x_test, y_test)
6
7 # Print model comparison
8 print("Single-layer MLP Accuracy:", accuracy)
9 print("Two-layer MLP Accuracy:", two_layer_accuracy)
10 print("Three-layer MLP Accuracy:", three_layer_accuracy)
11

```

```

282/282 [=====] - 2s 7ms/step
282/282 [=====] - 1s 4ms/step
Single-layer MLP Accuracy: 0.8795555555555555
Two-layer MLP Accuracy: 0.8806666666666667
Three-layer MLP Accuracy: 0.8785555555555555

```

```

1 # Evaluate models with different batch sizes
2 accuracy_batch_32, _, _, _ = evaluate_model(model_batch_32, x_test, y_test)
3 accuracy_batch_64, _, _, _ = evaluate_model(model_batch_64, x_test, y_test)
4 accuracy_batch_128, _, _, _ = evaluate_model(model_batch_128, x_test, y_test)
5
6 # Print model comparison
7 print("Single-layer MLP Accuracy (Batch Size = 32):", accuracy_batch_32)
8 print("Single-layer MLP Accuracy (Batch Size = 64):", accuracy_batch_64)
9 print("Single-layer MLP Accuracy (Batch Size = 128):", accuracy_batch_128)

```

```

⇒ 282/282 [=====] - 1s 3ms/step
282/282 [=====] - 1s 4ms/step
282/282 [=====] - 1s 3ms/step
Single-layer MLP Accuracy (Batch Size = 32): 0.88
Single-layer MLP Accuracy (Batch Size = 64): 0.8723333333333333
Single-layer MLP Accuracy (Batch Size = 128): 0.8797777777777778

```

```

1 # Evaluate models with different number of neurons
2 accuracy_neurons_64, _, _, _ = evaluate_model(model_neurons_64, x_test, y_test)
3 accuracy_neurons_128, _, _, _ = evaluate_model(model_neurons_128, x_test, y_test)
4 accuracy_neurons_256, _, _, _ = evaluate_model(model_neurons_256, x_test, y_test)
5
6 # Print model comparison
7 print("Single-layer MLP Accuracy (Neurons = 64):", accuracy_neurons_64)
8 print("Single-layer MLP Accuracy (Neurons = 128):", accuracy_neurons_128)
9 print("Single-layer MLP Accuracy (Neurons = 256):", accuracy_neurons_256)

```

```

⇒ 282/282 [=====] - 0s 2ms/step
282/282 [=====] - 0s 2ms/step
282/282 [=====] - 0s 1ms/step
Single-layer MLP Accuracy (Neurons = 64): 0.8773333333333333
Single-layer MLP Accuracy (Neurons = 128): 0.8754444444444445
Single-layer MLP Accuracy (Neurons = 256): 0.8813333333333333

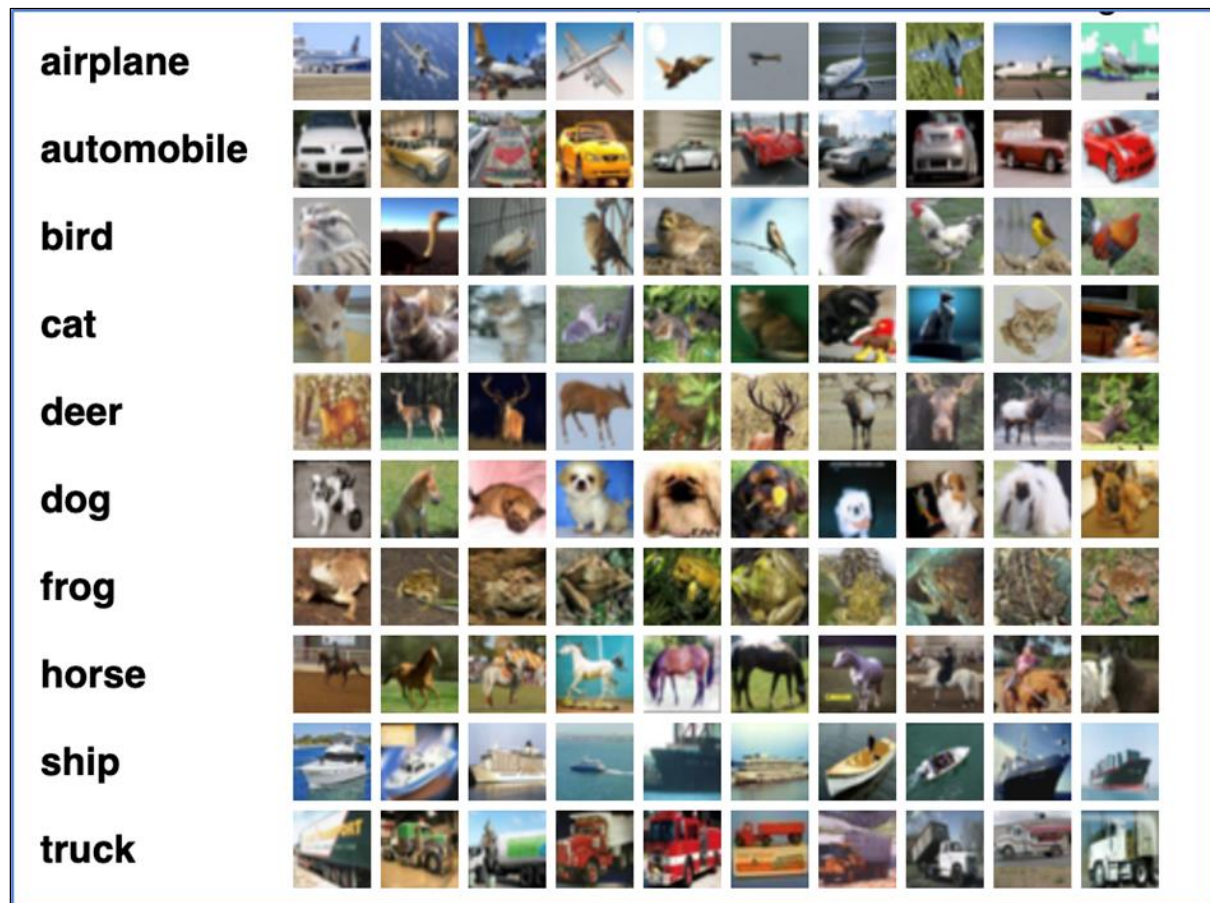
```



```
1 # Evaluate models with different optimizers
2 accuracy_sgd, _, _ = evaluate_model(model_sgd, x_test, y_test)
3 accuracy_adam, _, _ = evaluate_model(model_adam, x_test, y_test)
4 accuracy_rmsprop, _, _ = evaluate_model(model_rmsprop, x_test, y_test)
5
6 # Print model comparison
7 print("Single-layer MLP Accuracy (SGD):", accuracy_sgd)
8 print("Single-layer MLP Accuracy (Adam):", accuracy_adam)
9 print("Single-layer MLP Accuracy (RMSprop):", accuracy_rmsprop)
10
```

```
282/282 [=====] - 1s 2ms/step
282/282 [=====] - 1s 3ms/step
282/282 [=====] - 0s 1ms/step
Single-layer MLP Accuracy (SGD): 0.8492222222222222
Single-layer MLP Accuracy (Adam): 0.8816666666666667
Single-layer MLP Accuracy (RMSprop): 0.8663333333333333
```

Chapter 15: Building a CNN Model for CIFAR-10 with ChatGPT



✓ 56s



```

1 # Step 1: Preprocessing the image data
2
3 # Import necessary libraries
4 import numpy as np
5 import tensorflow as tf
6 from tensorflow.keras.datasets import cifar10
7 from tensorflow.keras.utils import to_categorical
8 from skimage.transform import resize
9
10 # Load CIFAR-10 dataset
11 (x_train, y_train), (x_test, y_test) = cifar10.load_data()
12
13 # Normalize pixel values to range [0, 1]
14 x_train = x_train.astype('float32') / 255.0
15 x_test = x_test.astype('float32') / 255.0
16
17 # Resize images to a standardized size (e.g., 32x32 pixels)
18 standardized_size = (32, 32)
19 x_train_resized = np.array([resize(image, standardized_size) for image in x_train])
20 x_test_resized = np.array([resize(image, standardized_size) for image in x_test])
21
22 # Display shape of datasets after preprocessing
23 print("Shape of x_train:", x_train_resized.shape)
24 print("Shape of x_test:", x_test_resized.shape)

```

Downloading data from <https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz>
 170498071/170498071 [=====] - 3s 0us/step
 Shape of x_train: (50000, 32, 32, 3)
 Shape of x_test: (10000, 32, 32, 3)

```

1 # Step 2: Model Construction
2
3 # Import necessary libraries
4 from tensorflow.keras.models import Sequential
5 from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
6
7 # Define the baseline CNN model architecture
8 model = Sequential([
9     # Add a convolutional layer with 32 filters, kernel size of (3, 3), and ReLU activation function
10     Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
11     # Add a max pooling layer with pool size of (2, 2)
12     MaxPooling2D((2, 2)),
13     # Flatten the output of the convolutional layer
14     Flatten(),
15     # Add a fully connected layer with 64 neurons and ReLU activation function
16     Dense(64, activation='relu'),
17     # Add the output layer with 10 neurons (one for each class) and softmax activation function
18     Dense(10, activation='softmax')
19 ])
20
21 # Compile the model
22 model.compile(optimizer='adam',
23               loss='sparse_categorical_crossentropy',
24               metrics=['accuracy'])
25
26 # Display model summary
27 model.summary()

```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
flatten (Flatten)	(None, 7200)	0
dense (Dense)	(None, 64)	460864
dense_1 (Dense)	(None, 10)	650

Total params: 462410 (1.76 MB)
 Trainable params: 462410 (1.76 MB)
 Non-trainable params: 0 (0.00 Byte)

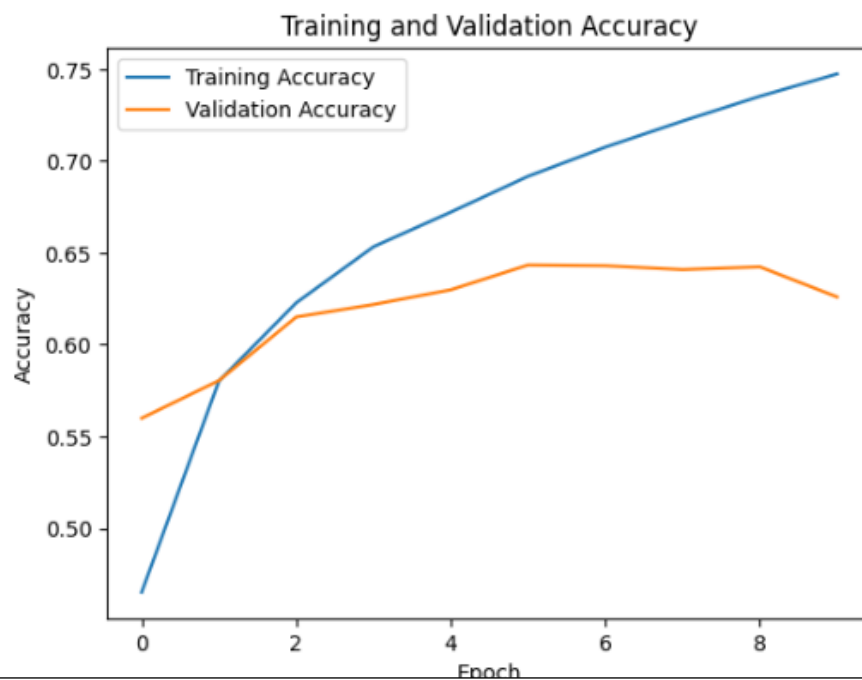
```
[ ] 1 # Step 3: Model Training
    2
    3 # Train the model
    4 history = model.fit(x_train_resized, y_train,
    5                     epochs=10,
    6                     batch_size=32,
    7                     validation_data=(x_test_resized, y_test))
```

```
Epoch 1/10
1563/1563 [=====] - 40s 25ms/step - loss: 1.5051 - accuracy: 0.4653 - val_loss: 1.2554 - val_accuracy: 0.5600
Epoch 2/10
1563/1563 [=====] - 39s 25ms/step - loss: 1.2019 - accuracy: 0.5802 - val_loss: 1.1988 - val_accuracy: 0.5805
Epoch 3/10
1563/1563 [=====] - 39s 25ms/step - loss: 1.0894 - accuracy: 0.6228 - val_loss: 1.1065 - val_accuracy: 0.6151
Epoch 4/10
1563/1563 [=====] - 41s 26ms/step - loss: 1.0049 - accuracy: 0.6533 - val_loss: 1.1028 - val_accuracy: 0.6219
Epoch 5/10
1563/1563 [=====] - 38s 24ms/step - loss: 0.9461 - accuracy: 0.6722 - val_loss: 1.0833 - val_accuracy: 0.6299
Epoch 6/10
1563/1563 [=====] - 41s 26ms/step - loss: 0.8892 - accuracy: 0.6917 - val_loss: 1.0305 - val_accuracy: 0.6434
Epoch 7/10
1563/1563 [=====] - 40s 25ms/step - loss: 0.8401 - accuracy: 0.7076 - val_loss: 1.0453 - val_accuracy: 0.6430
Epoch 8/10
1563/1563 [=====] - 39s 25ms/step - loss: 0.7990 - accuracy: 0.7217 - val_loss: 1.0580 - val_accuracy: 0.6409
Epoch 9/10
1563/1563 [=====] - 40s 26ms/step - loss: 0.7597 - accuracy: 0.7352 - val_loss: 1.0681 - val_accuracy: 0.6424
Epoch 10/10
1563/1563 [=====] - 39s 25ms/step - loss: 0.7248 - accuracy: 0.7474 - val_loss: 1.1223 - val_accuracy: 0.6260
```

```
[ ] 1 # Step 4: Model Evaluation and Visualization
2
3 import matplotlib.pyplot as plt
4
5 # Evaluate the model on the testing dataset
6 test_loss, test_accuracy = model.evaluate(x_test_resized, y_test, verbose=0)
7
8 print("Test Loss:", test_loss)
9 print("Test Accuracy:", test_accuracy)
10
11 # Plot training history
12 plt.plot(history.history['accuracy'], label='Training Accuracy')
13 plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
14 plt.title('Training and Validation Accuracy')
15 plt.xlabel('Epoch')
16 plt.ylabel('Accuracy')
17 plt.legend()
18 plt.show()
```



Test Loss: 1.1222519874572754
Test Accuracy: 0.6259999871253967



Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_1 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_3 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_4 (Conv2D)	(None, 4, 4, 64)	36928
max_pooling2d_3 (MaxPooling2D)	(None, 2, 2, 64)	0
flatten_1 (Flatten)	(None, 256)	0
dense_2 (Dense)	(None, 64)	16448
dense_3 (Dense)	(None, 10)	650

=====
Total params: 73418 (286.79 KB)
Trainable params: 73418 (286.79 KB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/10
1563/1563 [=====] - 10s 5ms/step - loss: 1.5440 - accuracy: 0.4342 - val_loss: 1.2825 - val_accuracy: 0.5462
Epoch 2/10
1563/1563 [=====] - 10s 6ms/step - loss: 1.1861 - accuracy: 0.5767 - val_loss: 1.0853 - val_accuracy: 0.6187
Epoch 3/10
1563/1563 [=====] - 11s 7ms/step - loss: 1.0438 - accuracy: 0.6310 - val_loss: 0.9983 - val_accuracy: 0.6543
Epoch 4/10
1563/1563 [=====] - 9s 6ms/step - loss: 0.9431 - accuracy: 0.6677 - val_loss: 0.9449 - val_accuracy: 0.6732
Epoch 5/10
1563/1563 [=====] - 9s 5ms/step - loss: 0.8770 - accuracy: 0.6923 - val_loss: 0.9397 - val_accuracy: 0.6711
Epoch 6/10
1563/1563 [=====] - 8s 5ms/step - loss: 0.8180 - accuracy: 0.7140 - val_loss: 0.8753 - val_accuracy: 0.6972
Epoch 7/10
1563/1563 [=====] - 8s 5ms/step - loss: 0.7749 - accuracy: 0.7282 - val_loss: 0.8775 - val_accuracy: 0.6969
Epoch 8/10
1563/1563 [=====] - 8s 5ms/step - loss: 0.7288 - accuracy: 0.7435 - val_loss: 0.9046 - val_accuracy: 0.6924
Epoch 9/10
1563/1563 [=====] - 8s 5ms/step - loss: 0.6930 - accuracy: 0.7582 - val_loss: 0.8837 - val_accuracy: 0.6962
Epoch 10/10
1563/1563 [=====] - 8s 5ms/step - loss: 0.6588 - accuracy: 0.7706 - val_loss: 0.8475 - val_accuracy: 0.7174
Test Loss (Modified Model with 64 Filters): 0.8474552035331726
Test Accuracy (Modified Model with 64 Filters): 0.7174000144004822

```
ValueError                                Traceback (most recent call last)
<ipython-input-12-d1ac82c289a7> in <cell line: 2>()
----> 1 # Define the modified model with two consecutive layers of 128 filters added to the 64-layer model
      2 modified_model_64_to_128 = Sequential([
      3     Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
      4     MaxPooling2D((2, 2)),
      5     Conv2D(64, (3, 3), activation='relu'),

~
~
~
2 frames
/usr/local/lib/python3.10/dist-packages/tensorflow/python/framework/ops.py in _create_c_op(graph, node_def, inputs, control_inputs, op_def, extract_traceback)
   1018     except errors.InvalidArgumentError as e:
   1019         # Convert to ValueError for backwards compatibility.
-> 1020         raise ValueError(e.message)
   1021
   1022     # Record the current Python stack trace as the creating stacktrace of this

ValueError: Exception encountered when calling layer "conv2d_38" (type Conv2D).

Negative dimension size caused by subtracting 3 from 2 for '{{node conv2d_38/Conv2D}} = Conv2D[DT=DT_FLOAT, data_format="NHWC", dilations=[1, 1, 1, 1], explicit_paddings=[], padding="VALID", strides=[1, 1, 1, 1], use_cudnn_on_gpu=true](Placeholder, conv2d_38/Conv2D/ReadVariableOp)' with input shapes: [?,2,2,64], [3,3,64,128].

Call arguments received by layer "conv2d_38" (type Conv2D):
  * inputs=tf.Tensor(shape=(None, 2, 2, 64), dtype=float32)
```

Model: "sequential_9"

Layer (type)	Output Shape	Param #
conv2d_40 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_34 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_41 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_35 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_42 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_43 (Conv2D)	(None, 4, 4, 128)	73856
conv2d_44 (Conv2D)	(None, 4, 4, 128)	147584
max_pooling2d_36 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_9 (Flatten)	(None, 512)	0
dense_18 (Dense)	(None, 64)	32832
dense_19 (Dense)	(None, 10)	650

=====
Total params: 311242 (1.19 MB)
Trainable params: 311242 (1.19 MB)
Non-trainable params: 0 (0.00 Byte)

Epoch 1/10
1563/1563 [=====] - 12s 7ms/step - loss: 1.5531 - accuracy: 0.4194 - val_loss: 1.3012 - val_accuracy: 0.5338
Epoch 2/10
1563/1563 [=====] - 10s 6ms/step - loss: 1.1590 - accuracy: 0.5874 - val_loss: 1.1143 - val_accuracy: 0.5969
Epoch 3/10
1563/1563 [=====] - 9s 6ms/step - loss: 0.9821 - accuracy: 0.6528 - val_loss: 0.9471 - val_accuracy: 0.6691
Epoch 4/10
1563/1563 [=====] - 10s 6ms/step - loss: 0.8647 - accuracy: 0.6960 - val_loss: 0.9182 - val_accuracy: 0.6881
Epoch 5/10
1563/1563 [=====] - 10s 6ms/step - loss: 0.7730 - accuracy: 0.7271 - val_loss: 0.9124 - val_accuracy: 0.6931
Epoch 6/10
1563/1563 [=====] - 10s 6ms/step - loss: 0.6967 - accuracy: 0.7541 - val_loss: 0.8611 - val_accuracy: 0.7045
Epoch 7/10
1563/1563 [=====] - 9s 6ms/step - loss: 0.6228 - accuracy: 0.7804 - val_loss: 1.0315 - val_accuracy: 0.6705
Epoch 8/10
1563/1563 [=====] - 10s 6ms/step - loss: 0.5574 - accuracy: 0.8050 - val_loss: 0.9043 - val_accuracy: 0.7114
Epoch 9/10
1563/1563 [=====] - 10s 7ms/step - loss: 0.4964 - accuracy: 0.8224 - val_loss: 0.9639 - val_accuracy: 0.6958
Epoch 10/10
1563/1563 [=====] - 9s 6ms/step - loss: 0.4393 - accuracy: 0.8438 - val_loss: 1.0033 - val_accuracy: 0.6993
Test Loss (Modified Model with 64 to 128 Filters): 1.0032662153244019
Test Accuracy (Modified Model with 64 to 128 Filters): 0.6992999911308289

Model: "sequential_10"

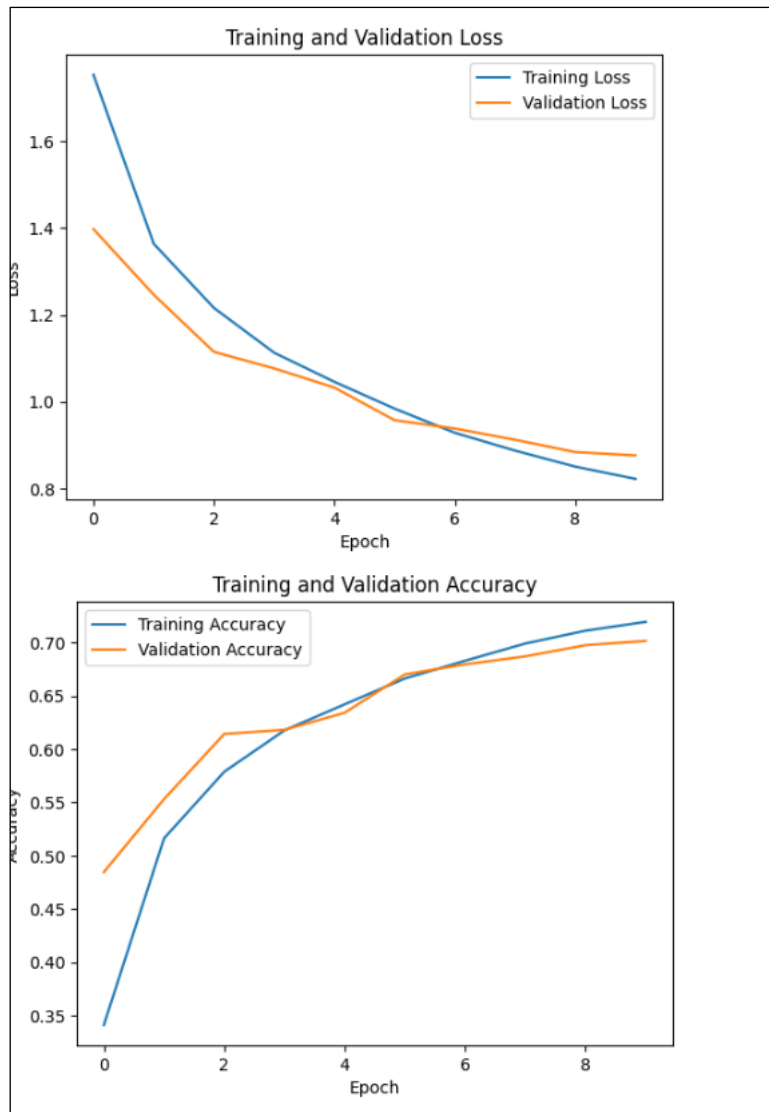
Layer (type)	Output Shape	Param #
conv2d_45 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_37 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_46 (Conv2D)	(None, 13, 13, 64)	18496
max_pooling2d_38 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_47 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_48 (Conv2D)	(None, 4, 4, 128)	73856
dropout (Dropout)	(None, 4, 4, 128)	0
conv2d_49 (Conv2D)	(None, 4, 4, 128)	147584
dropout_1 (Dropout)	(None, 4, 4, 128)	0
max_pooling2d_39 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_10 (Flatten)	(None, 512)	0
dense_20 (Dense)	(None, 64)	32832
dropout_2 (Dropout)	(None, 64)	0
dense_21 (Dense)	(None, 10)	650

=====
Total params: 311242 (1.19 MB)

Trainable params: 311242 (1.19 MB)

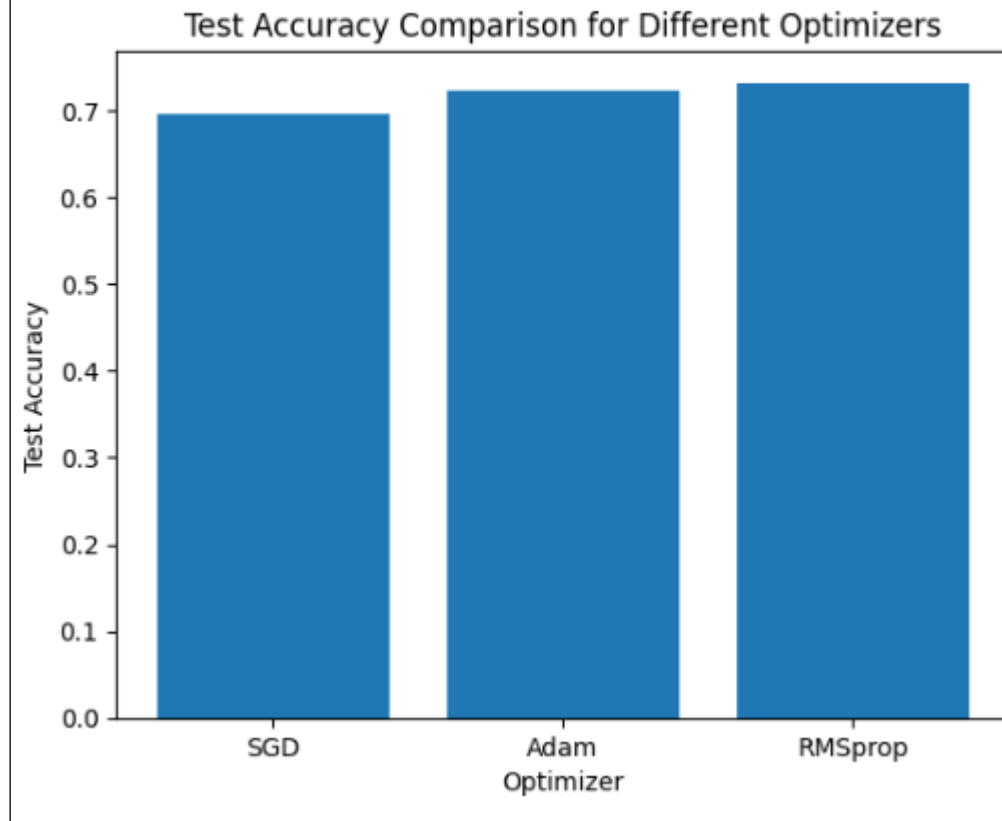
Non-trainable params: 0 (0.00 Byte)

Epoch 1/10
1563/1563 [=====] - 15s 7ms/step - loss: 1.7530 - accuracy: 0.3412 - val_loss: 1.3975 - val_accuracy: 0.4846
Epoch 2/10
1563/1563 [=====] - 11s 7ms/step - loss: 1.3638 - accuracy: 0.5163 - val_loss: 1.2464 - val_accuracy: 0.5531
Epoch 3/10
1563/1563 [=====] - 12s 7ms/step - loss: 1.2159 - accuracy: 0.5788 - val_loss: 1.1147 - val_accuracy: 0.6142
Epoch 4/10
1563/1563 [=====] - 11s 7ms/step - loss: 1.1127 - accuracy: 0.6176 - val_loss: 1.0766 - val_accuracy: 0.6181
Epoch 5/10
1563/1563 [=====] - 11s 7ms/step - loss: 1.0457 - accuracy: 0.6420 - val_loss: 1.0325 - val_accuracy: 0.6341
Epoch 6/10
1563/1563 [=====] - 10s 7ms/step - loss: 0.9839 - accuracy: 0.6662 - val_loss: 0.9573 - val_accuracy: 0.6701
Epoch 7/10
1563/1563 [=====] - 11s 7ms/step - loss: 0.9283 - accuracy: 0.6829 - val_loss: 0.9381 - val_accuracy: 0.6795
Epoch 8/10
1563/1563 [=====] - 11s 7ms/step - loss: 0.8872 - accuracy: 0.6990 - val_loss: 0.9123 - val_accuracy: 0.6870
Epoch 9/10
1563/1563 [=====] - 11s 7ms/step - loss: 0.8503 - accuracy: 0.7111 - val_loss: 0.8838 - val_accuracy: 0.6975
Epoch 10/10
1563/1563 [=====] - 12s 8ms/step - loss: 0.8220 - accuracy: 0.7194 - val_loss: 0.8761 - val_accuracy: 0.7015
Test Loss (Modified Model with Dropout): 0.876133382320404
Test Accuracy (Modified Model with Dropout): 0.7014999985694885



batch_normalization (Batch Normalization)	(None, 30, 30, 32)	128
max_pooling2d_3 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_6 (Conv2D)	(None, 13, 13, 64)	18496
batch_normalization_1 (Batch Normalization)	(None, 13, 13, 64)	256
max_pooling2d_4 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_7 (Conv2D)	(None, 4, 4, 64)	36928
conv2d_8 (Conv2D)	(None, 4, 4, 128)	73856
batch_normalization_2 (Batch Normalization)	(None, 4, 4, 128)	512
max_pooling2d_5 (MaxPooling2D)	(None, 2, 2, 128)	0
conv2d_9 (Conv2D)	(None, 2, 2, 128)	147584
batch_normalization_3 (Batch Normalization)	(None, 2, 2, 128)	512
max_pooling2d_6 (MaxPooling2D)	(None, 1, 1, 128)	0
flatten_1 (Flatten)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
batch_normalization_4 (Batch Normalization)	(None, 64)	256
dense_3 (Dense)	(None, 10)	650
=====		
Total params: 288330 (1.10 MB)		
Trainable params: 287498 (1.10 MB)		
Non-trainable params: 832 (3.25 KB)		
Epoch 1/10		
1563/1563 [=====] - 21s 8ms/step - loss: 1.3927 - accuracy: 0.5045 - val_loss: 1.4828 - val_accuracy: 0.5232		
Epoch 2/10		
1563/1563 [=====] - 11s 7ms/step - loss: 1.0016 - accuracy: 0.6489 - val_loss: 0.9775 - val_accuracy: 0.6596		
Epoch 3/10		
1563/1563 [=====] - 12s 7ms/step - loss: 0.8231 - accuracy: 0.7152 - val_loss: 1.0615 - val_accuracy: 0.6423		
Epoch 4/10		
1563/1563 [=====] - 12s 8ms/step - loss: 0.7145 - accuracy: 0.7522 - val_loss: 0.9972 - val_accuracy: 0.6483		
Epoch 5/10		
1563/1563 [=====] - 12s 7ms/step - loss: 0.6202 - accuracy: 0.7861 - val_loss: 0.8076 - val_accuracy: 0.7213		
Epoch 6/10		
1563/1563 [=====] - 12s 7ms/step - loss: 0.5490 - accuracy: 0.8106 - val_loss: 1.0868 - val_accuracy: 0.6462		
Epoch 7/10		
1563/1563 [=====] - 12s 8ms/step - loss: 0.4774 - accuracy: 0.8349 - val_loss: 0.7828 - val_accuracy: 0.7444		
Epoch 8/10		
1563/1563 [=====] - 12s 7ms/step - loss: 0.4215 - accuracy: 0.8518 - val_loss: 0.9065 - val_accuracy: 0.7209		
Epoch 9/10		
1563/1563 [=====] - 12s 8ms/step - loss: 0.3664 - accuracy: 0.8712 - val_loss: 0.9365 - val_accuracy: 0.7152		
Epoch 10/10		
1563/1563 [=====] - 12s 8ms/step - loss: 0.3223 - accuracy: 0.8865 - val_loss: 0.9293 - val_accuracy: 0.7268		
Test Loss (Modified Model with Batch Normalization): 0.9293251037597656		
Test Accuracy (Modified Model with Batch Normalization): 0.726800245094299		

Test Accuracy (Optimizer: SGD): 0.6955999732017517
Test Accuracy (Optimizer: Adam): 0.7224000096321106
Test Accuracy (Optimizer: RMSprop): 0.7318999767303467



```
ValueError                                Traceback (most recent call last)
<ipython-input-7-84ffdc3d4264> in <cell line: 55>()
    53
    54 # Compile the model
--> 55 optimizer = SGD(momentum=0.9, decay=0.000125)
    56 model_davidnet.compile(optimizer=optimizer,
    57                        loss='sparse_categorical_crossentropy',

~
~
~
3 frames
/usr/local/lib/python3.10/dist-packages/keras/src/optimizers/optimizer.py in _process_kwargs(self, kwargs)
    133     for k in kwargs:
    134         if k in legacy_kwargs:
--> 135             raise ValueError(
    136                 f"{k} is deprecated in the new Keras optimizer, please "
    137                 "check the docstring for valid arguments, or use the "

ValueError: decay is deprecated in the new Keras optimizer, please check the docstring for valid arguments, or use the legacy optimizer, e.g., tf.keras.optimizers.legacy.SGD.
```

```
Epoch 13/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9439 - val_accuracy: 0.7734
Learning rate for epoch 14 is 0.0002621440000000001
Epoch 14/24
98/98 [=====] - 31s 319ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9443 - val_accuracy: 0.7731
Learning rate for epoch 15 is 0.00010485760000000006
Epoch 15/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9443 - val_accuracy: 0.7733
Learning rate for epoch 16 is 4.1943040000000025e-05
Epoch 16/24
98/98 [=====] - 32s 326ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9442 - val_accuracy: 0.7733
Learning rate for epoch 17 is 1.677721600000001e-05
Epoch 17/24
98/98 [=====] - 32s 326ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9443 - val_accuracy: 0.7731
Learning rate for epoch 18 is 6.7108864000000044e-06
Epoch 18/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9443 - val_accuracy: 0.7733
Learning rate for epoch 19 is 2.684354560000002e-06
Epoch 19/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.9444 - val_accuracy: 0.7734
Learning rate for epoch 20 is 1.073741824000001e-06
Epoch 20/24
98/98 [=====] - 31s 319ms/step - loss: 0.0034 - accuracy: 0.9999 - val_loss: 0.9441 - val_accuracy: 0.7732
Learning rate for epoch 21 is 4.294967296000004e-07
Epoch 21/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.9442 - val_accuracy: 0.7733
Learning rate for epoch 22 is 1.7179869184000014e-07
Epoch 22/24
98/98 [=====] - 31s 318ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9443 - val_accuracy: 0.7733
Learning rate for epoch 23 is 6.871947673600007e-08
Epoch 23/24
98/98 [=====] - 32s 326ms/step - loss: 0.0035 - accuracy: 0.9999 - val_loss: 0.9442 - val_accuracy: 0.7732
Learning rate for epoch 24 is 2.748779069440003e-08
Epoch 24/24
98/98 [=====] - 32s 326ms/step - loss: 0.0035 - accuracy: 1.0000 - val_loss: 0.9443 - val_accuracy: 0.7734
```

Chapter 16: Unsupervised Learning: Clustering and PCA

Column names and data types:

```
InvoiceNo      object
StockCode      object
Description    object
Quantity       int64
InvoiceDate    object
UnitPrice      float64
CustomerID     float64
Country        object
dtype: object
```

Missing values per column:

```
InvoiceNo      0
StockCode      0
Description     1454
Quantity       0
InvoiceDate    0
UnitPrice      0
CustomerID     135080
Country        0
dtype: int64
```

Summary Statistics:

	count	unique	top	freq	\
InvoiceNo	541909	25900	573585	1114	
StockCode	541909	4070	85123A	2313	
Description	540455	4223	WHITE HANGING HEART T-LIGHT HOLDER	2369	
Quantity	541909.0	NaN	NaN	NaN	
InvoiceDate	541909	23260	10/31/2011 14:41	1114	
UnitPrice	541909.0	NaN	NaN	NaN	
CustomerID	406829.0	NaN	NaN	NaN	
Country	541909	38	United Kingdom	495478	

	mean	std	min	25%	50%	75%	\
InvoiceNo	NaN	NaN	NaN	NaN	NaN	NaN	
StockCode	NaN	NaN	NaN	NaN	NaN	NaN	
Description	NaN	NaN	NaN	NaN	NaN	NaN	
Quantity	9.55225	218.081158	-80995.0	1.0	3.0	10.0	
InvoiceDate	NaN	NaN	NaN	NaN	NaN	NaN	
UnitPrice	4.611114	96.759853	-11062.06	1.25	2.08	4.13	
CustomerID	15287.69057	1713.600303	12346.0	13953.0	15152.0	16791.0	
Country	NaN	NaN	NaN	NaN	NaN	NaN	

	max	InvoiceNo
NaN		
StockCode	NaN	
Description	NaN	
Quantity	80995.0	
InvoiceDate	NaN	
UnitPrice	38970.0	
CustomerID	18287.0	
Country	NaN	

Shape of the cleaned DataFrame: (397924, 4)

First few rows of the cleaned DataFrame:

	InvoiceNo	Quantity	UnitPrice	CustomerID
0	536365	6	2.55	17850.0
1	536365	6	3.39	17850.0
2	536365	8	2.75	17850.0
3	536365	6	3.39	17850.0
4	536365	6	3.39	17850.0

Customer Segmentation DataFrame:

	CustomerID	TotalAmount	TotalTransactions
0	12346.0	77183.60	1
1	12347.0	4310.00	7
2	12348.0	1797.24	4
3	12349.0	1757.55	1
4	12350.0	334.40	1

Shape of DataFrame before removing outliers: (4339, 2)

Shape of DataFrame after removing outliers: (3912, 2)

Customer Segmentation DataFrame with Cluster Assignments:

	TotalAmount	TotalTransactions	Cluster
2	1797.24	4	3
3	1757.55	1	3
4	334.40	1	5
5	2506.04	8	2
6	89.00	1	5

C:\Users\marin\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.9_qbz5n2kf

Data columns (total 8 columns):

#	Column	Non-Null Count	Dtype
0	InvoiceNo	541909 non-null	object
1	StockCode	541909 non-null	object
2	Description	540455 non-null	object
3	Quantity	541909 non-null	int64
4	InvoiceDate	541909 non-null	object
5	UnitPrice	541909 non-null	float64
6	CustomerID	406829 non-null	float64
7	Country	541909 non-null	object

dtypes: float64(2), int64(1), object(5)

memory usage: 33.1+ MB

None

	InvoiceNo	StockCode	Description	Quantity	\
0	536365	85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	
1	536365	71053	WHITE METAL LANTERN	6	
2	536365	84406B	CREAM CUPID HEARTS COAT HANGER	8	
3	536365	84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	
4	536365	84029E	RED WOOLLY HOTTIE WHITE HEART.	6	

	InvoiceDate	UnitPrice	CustomerID	Country
0	12/1/2010 8:26	2.55	17850.0	United Kingdom
1	12/1/2010 8:26	3.39	17850.0	United Kingdom
2	12/1/2010 8:26	2.75	17850.0	United Kingdom
3	12/1/2010 8:26	3.39	17850.0	United Kingdom
4	12/1/2010 8:26	3.39	17850.0	United Kingdom

[nltk_data] Downloading package punkt to /root/nltk_data...

[nltk_data] Unzipping tokenizers/punkt.zip.

[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Unzipping corpora/stopwords.zip.

<ipython-input-4-e54dfbc360ba>:24: SettingWithCopyWarning:

A value is trying to be set on a copy of a slice from a DataFrame.

Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: <https://pandas.pydata.org/pandas-docs/stable/>

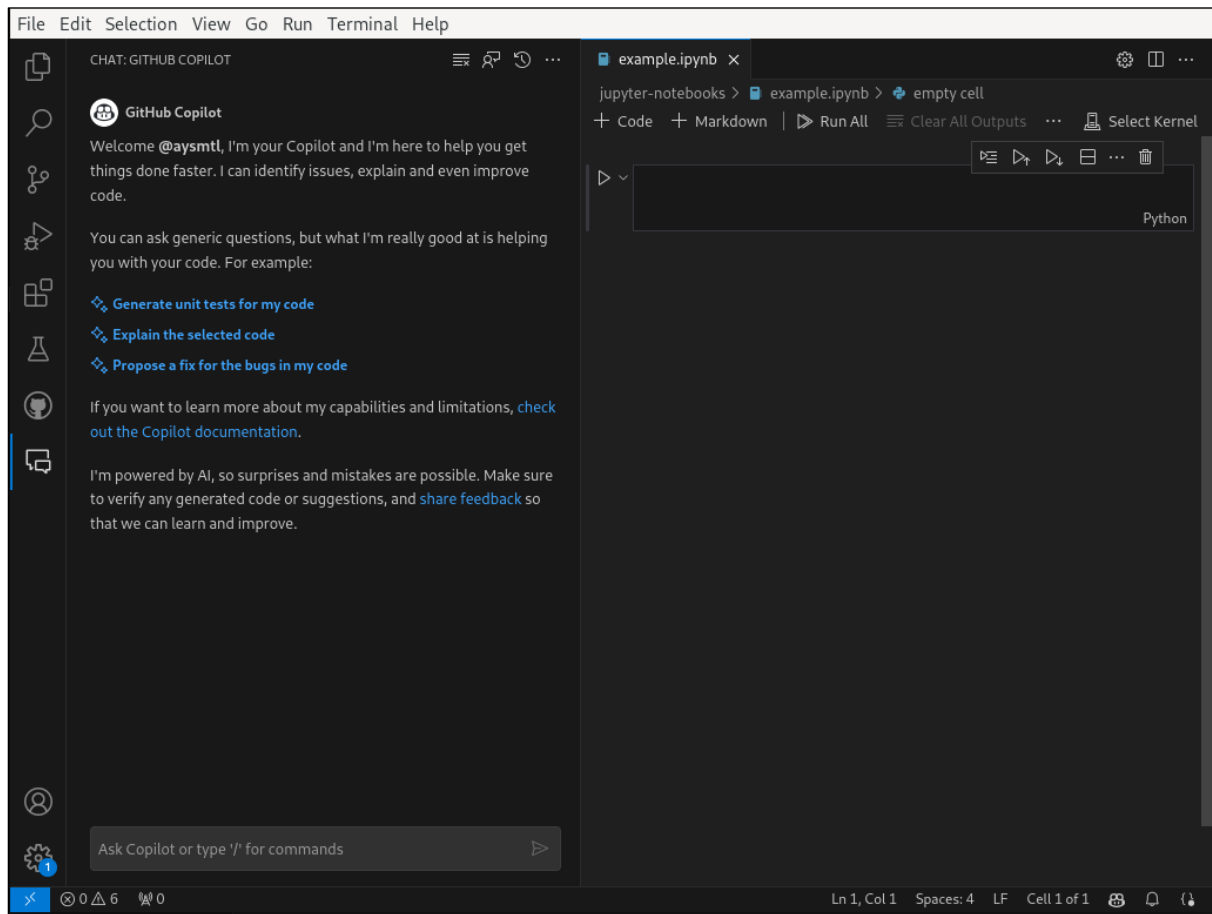
df['Description'] = df['Description'].str.lower()

	00	10	12	16	20	20713	23343	24	30cm	36	...	world	woven	\
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	0.0	0.0	

	wrap	wreath	writing	wrong	wrongly	xmas	yellow	zinc
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

[5 rows x 1000 columns]

Chapter 17: Machine Learning with Copilot



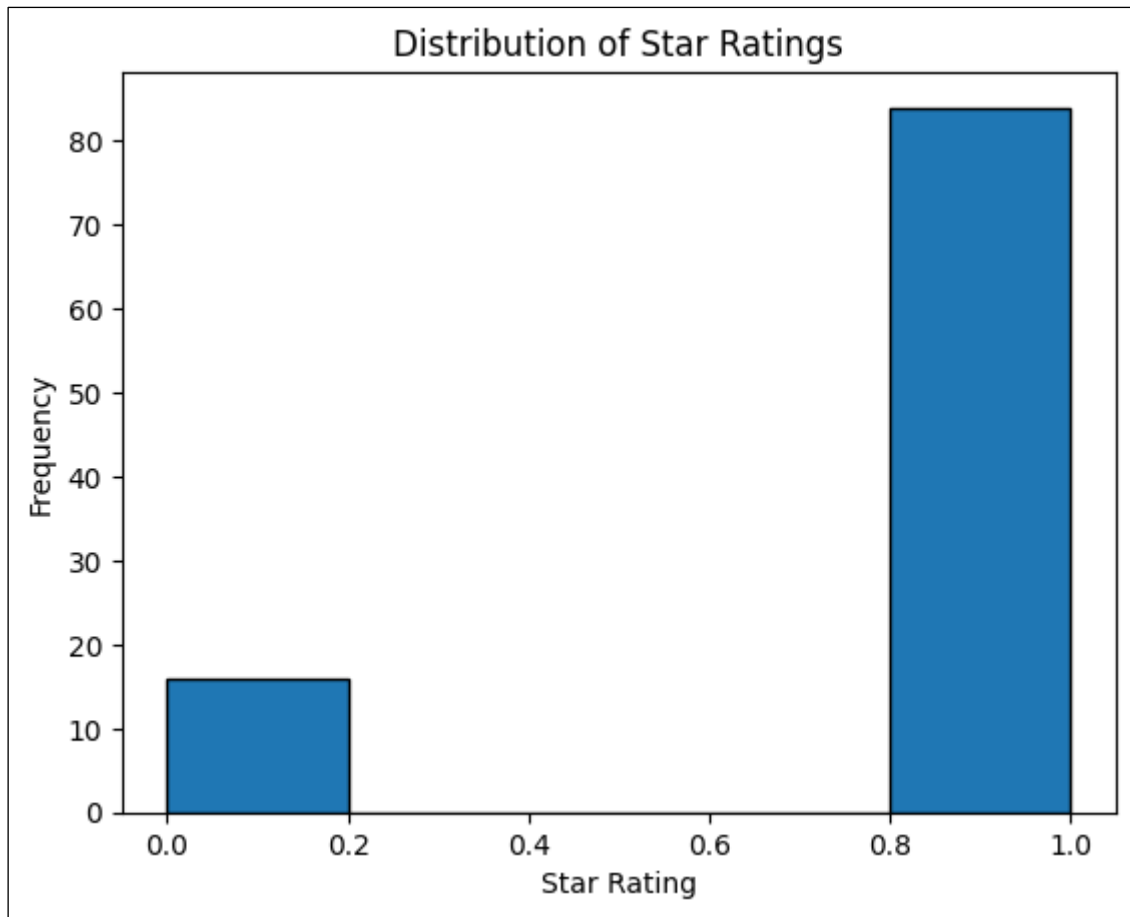
I am performing data exploration for the AwesomeShop e-commerce project. The dataset contains information about various products and their reviews. I want to gain insights into the data, identify patterns, and understand the characteristics of the reviews. Can you provide me with some exploratory analysis techniques and code snippets to help me uncover interesting insights from the dataset? The dataset for the AwesomeShop E-commerce project contains information about different products and their reviews. It includes the following columns:

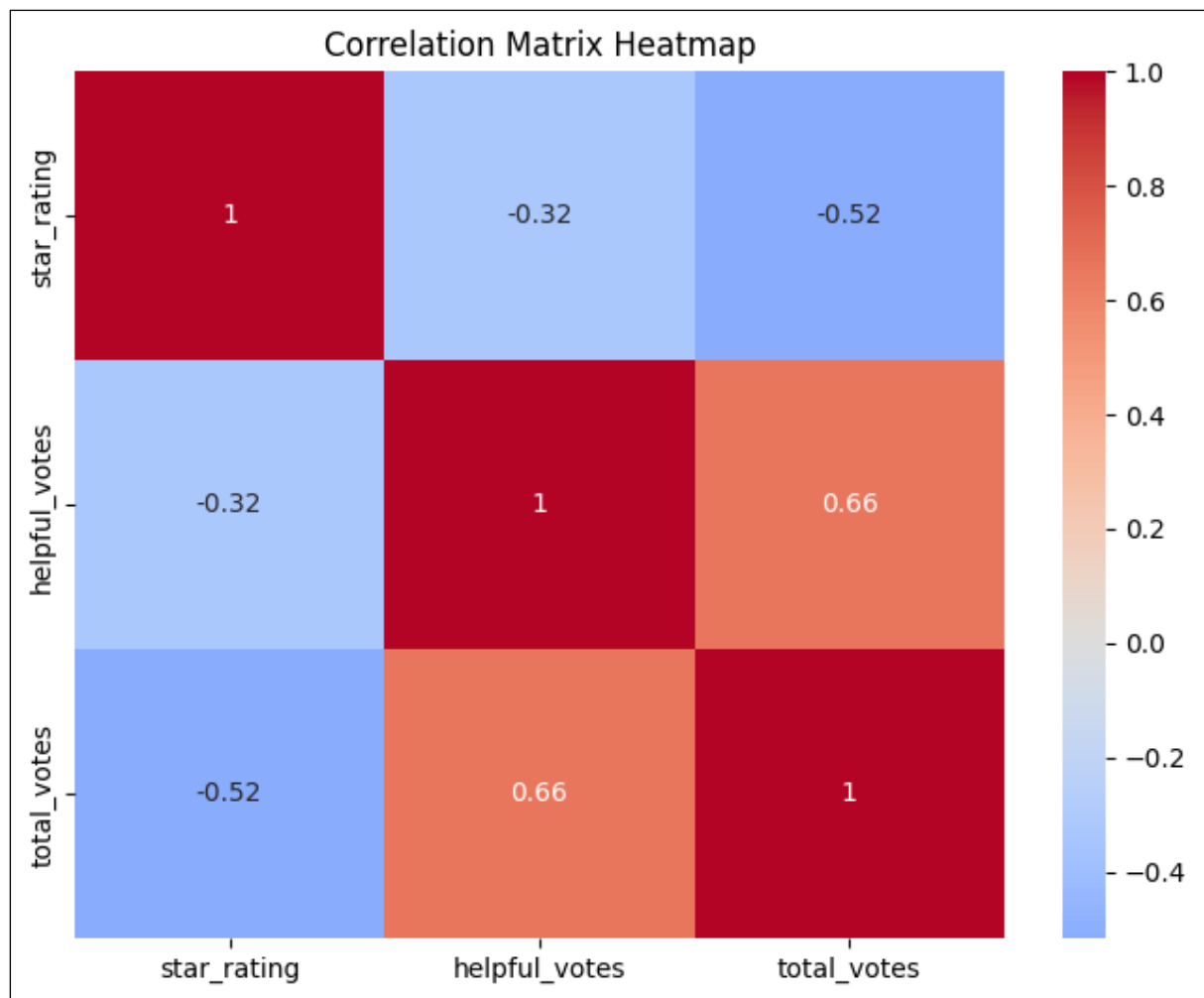
1. marketplace (string): Location of the product
2. customer_id (string): Unique ID of the customer
3. review_id (string): Reviewing ID
4. product_id (string): Unique ID of the product
5. product_parent (string): Parent product
6. product_title (string): Title of the product reviewed
7. product_category (string): Different product categories
8. star_rating (int): Rating of the product out of 5
9. helpful_votes (int): Number of helpful votes for the product
10. total_votes (int): Total number of votes for the product
11. review_headline (string): Heading of the review
12. review_body (string): Content of the review
13. review_date (string): Date at which the product is reviewed
14. sentiment (string): Sentiment of the review (positive or negative)

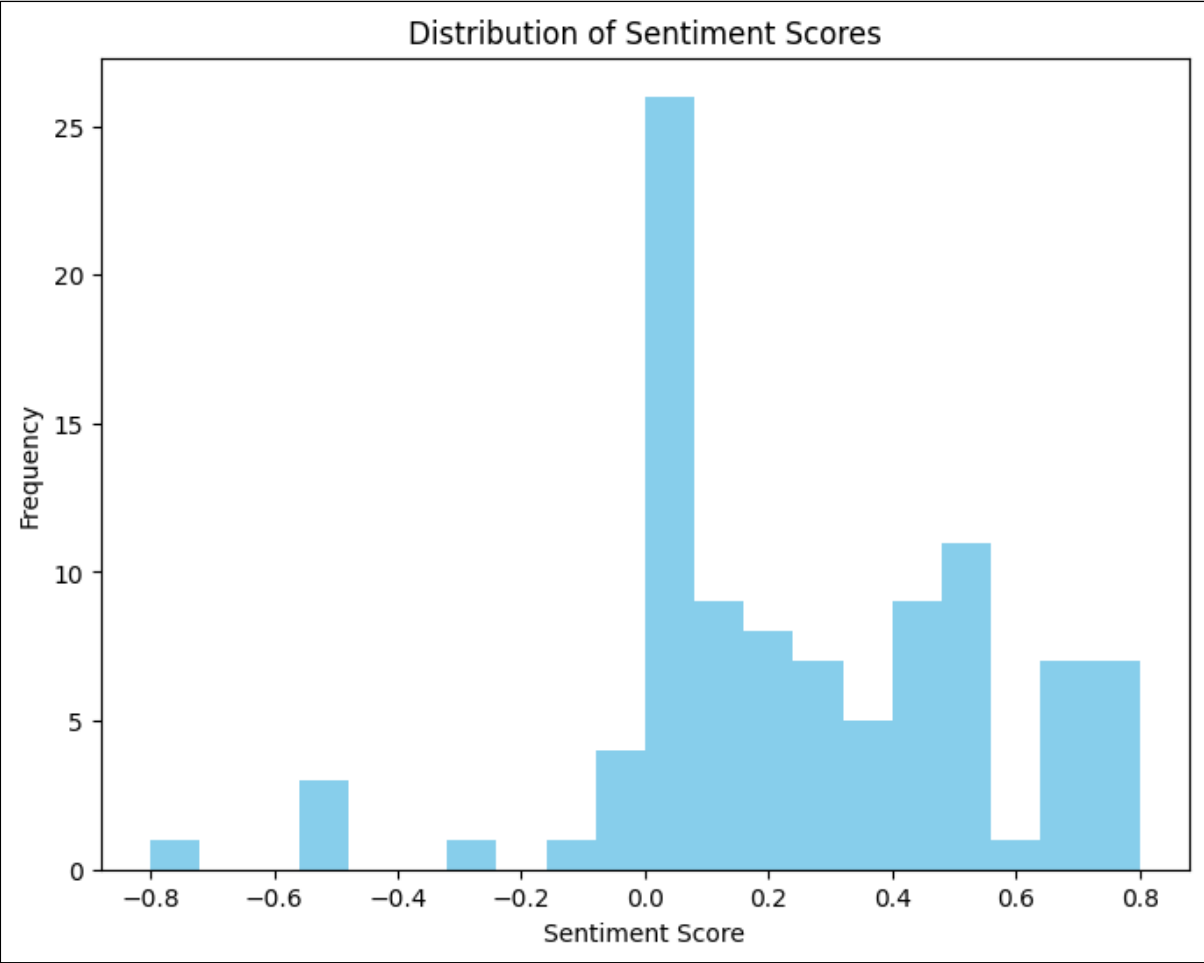
1. Load the Dataset: Read the dataset into a pandas DataFrame to work with the data effectively.
2. Inspect the Data: Display the first few rows of the DataFrame to get a glimpse of the data. Check the column names, data types, and any missing values.
3. Summary Statistics: Calculate summary statistics such as mean, median, minimum, maximum, and quartiles for numeric columns. This helps in understanding the distribution and range of values.
4. Explore Categorical Variables: Analyze the unique values and their frequencies for categorical variables like marketplace, product_category, and sentiment. Visualizations such as bar plots can be helpful for this analysis.
5. Distribution of Ratings: Plot a histogram or bar plot to visualize the distribution of star_ratings. This helps in understanding the overall sentiment of the reviews.
6. Temporal Analysis: Analyze the temporal aspect of the data by examining the review_date column. Explore trends, seasonality, or any patterns over time.
7. Review Length Analysis: Analyze the length of review_body to understand the amount of information provided in the reviews. Calculate descriptive statistics like mean, median, and maximum length.
8. Correlation Analysis: Investigate the correlation between numeric variables using correlation matrices or scatter plots. This helps in

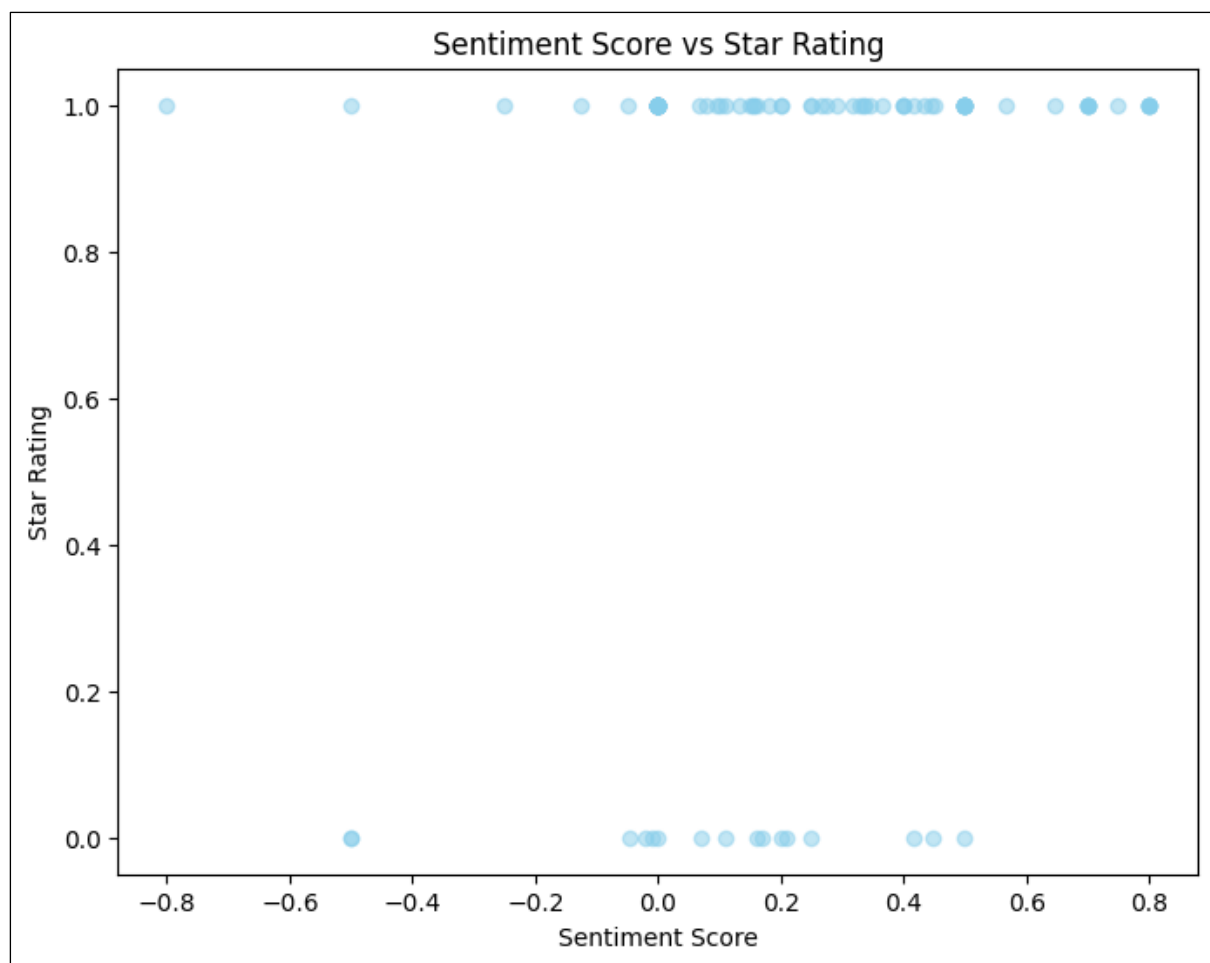
-

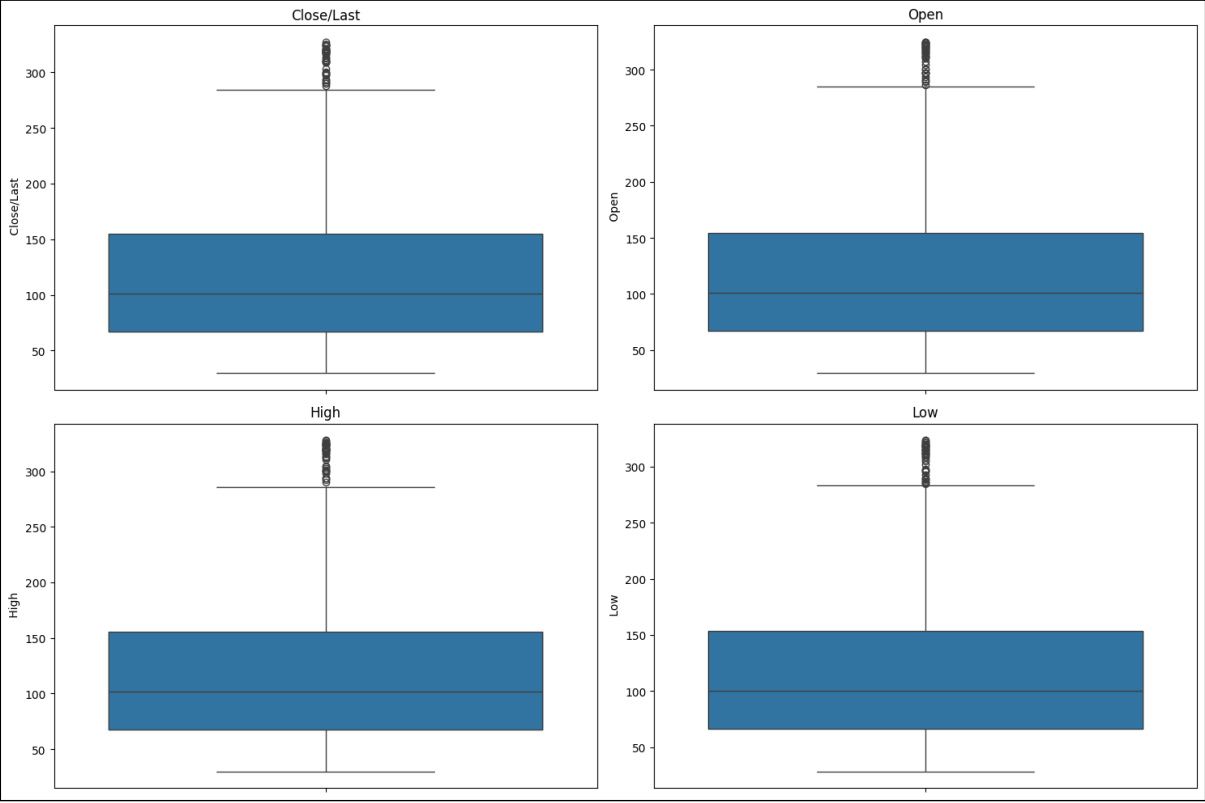
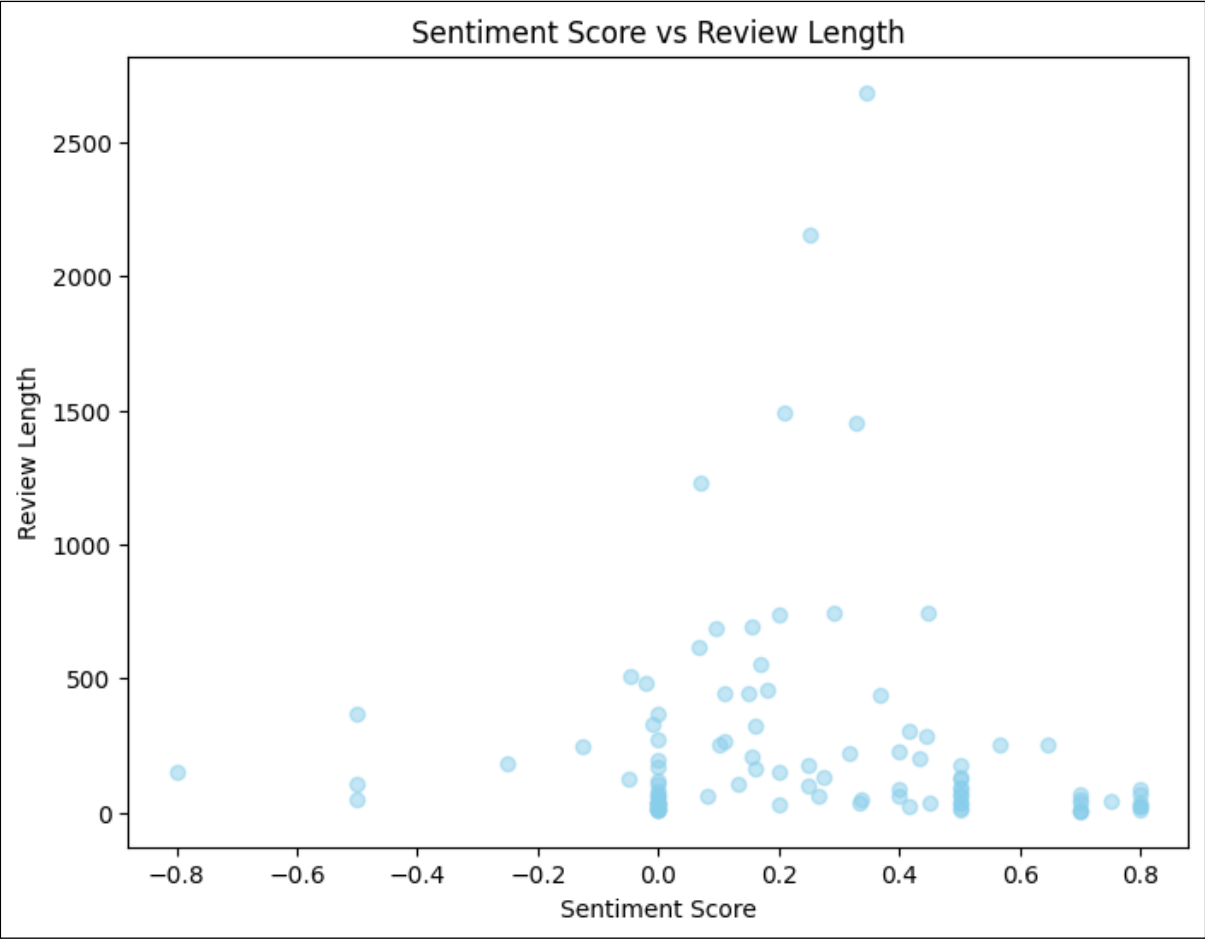


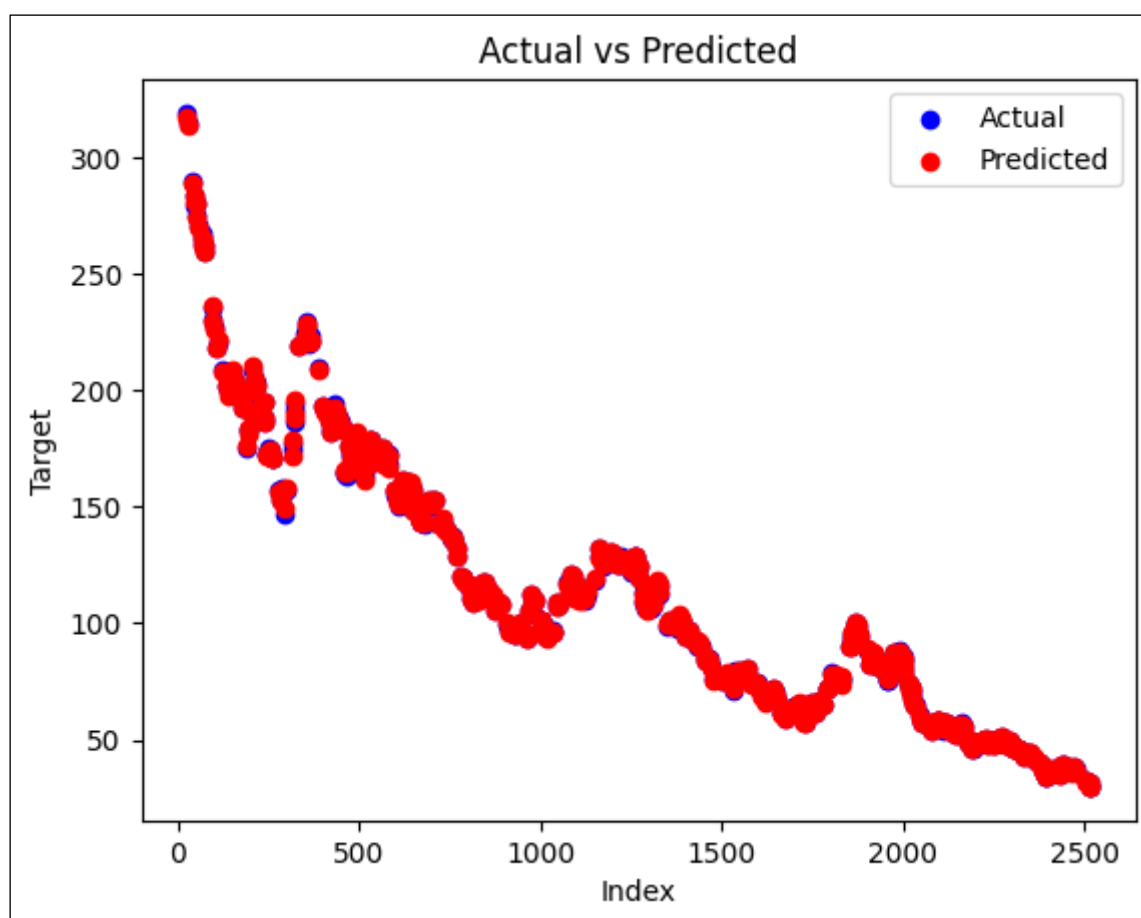
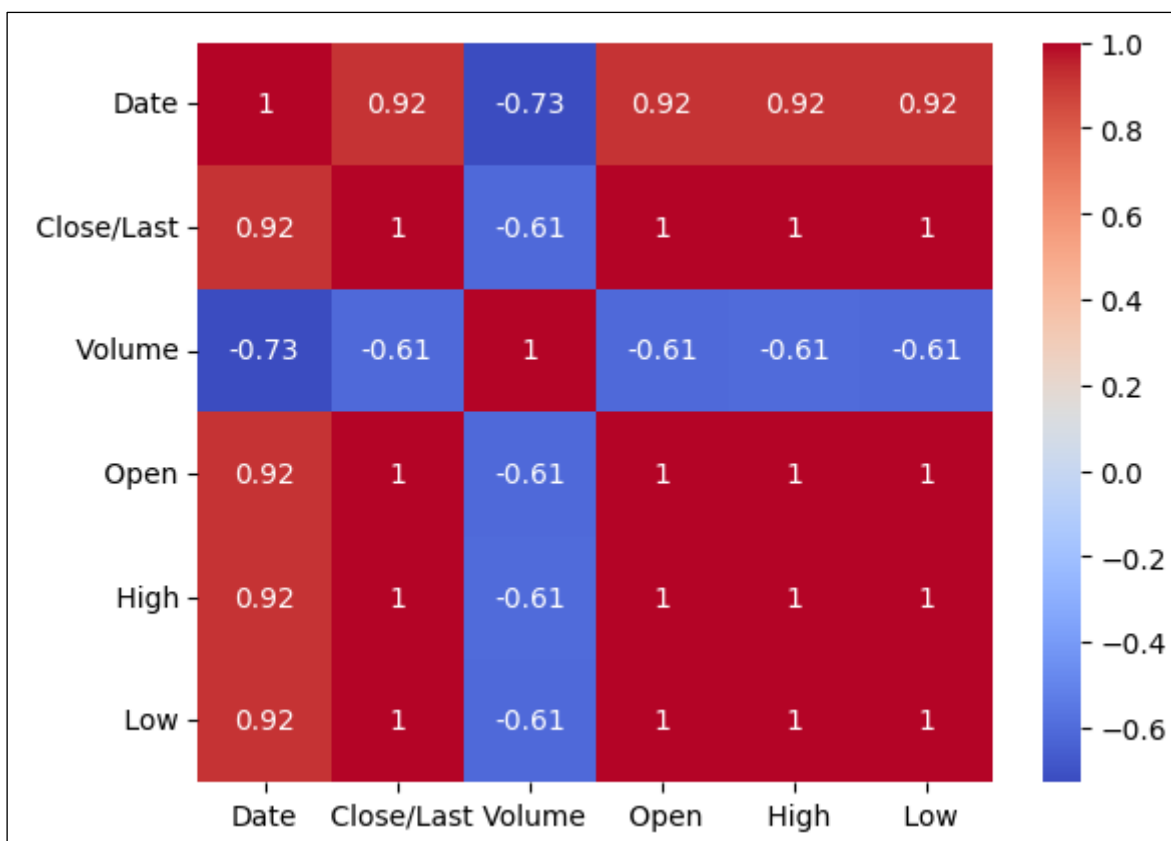


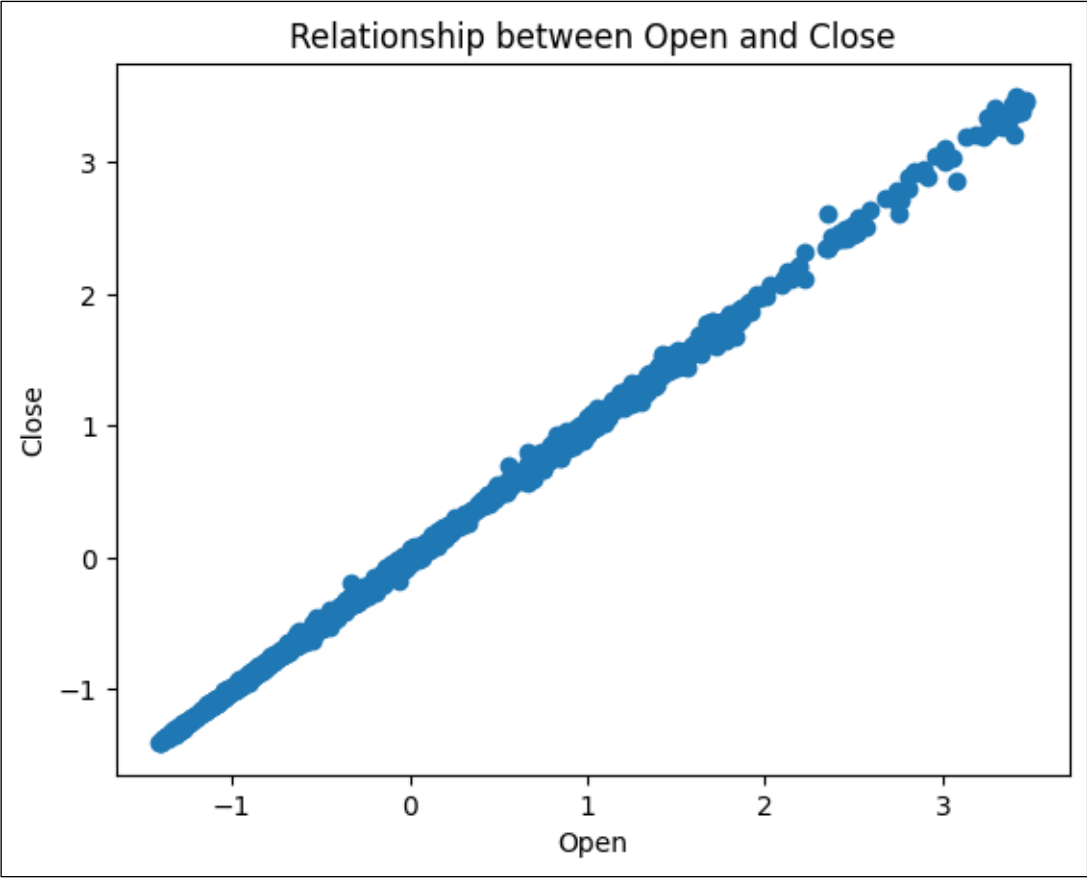




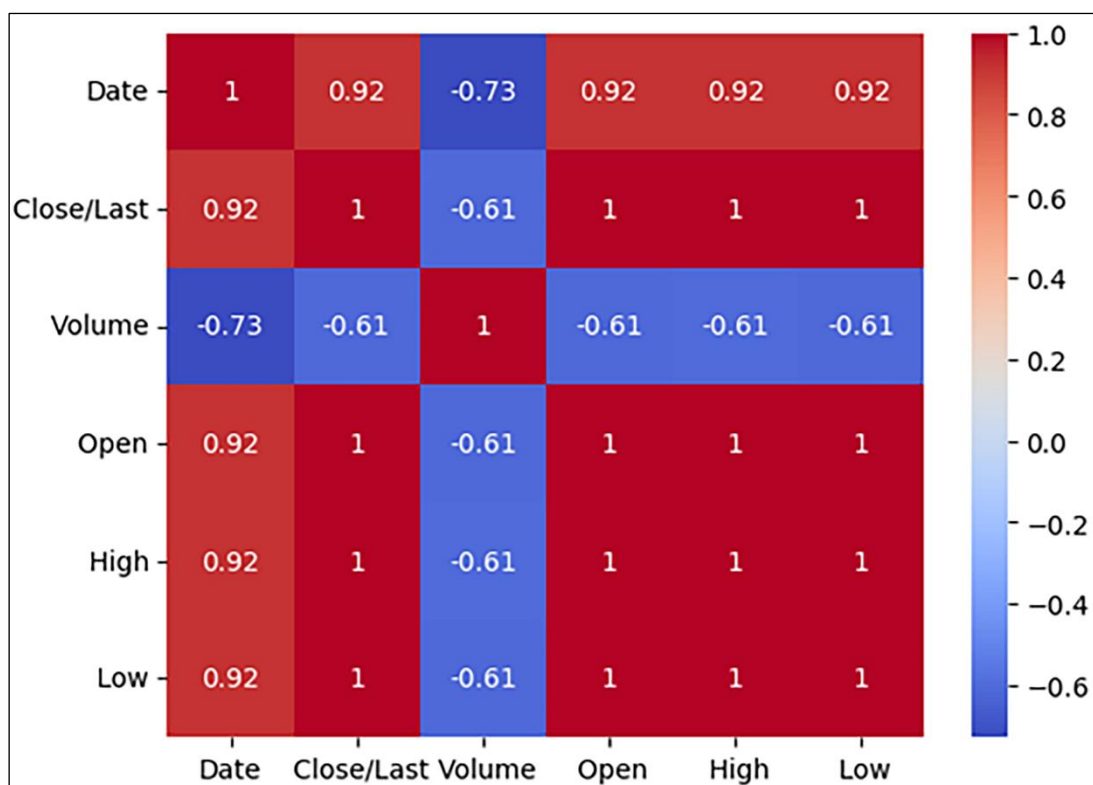
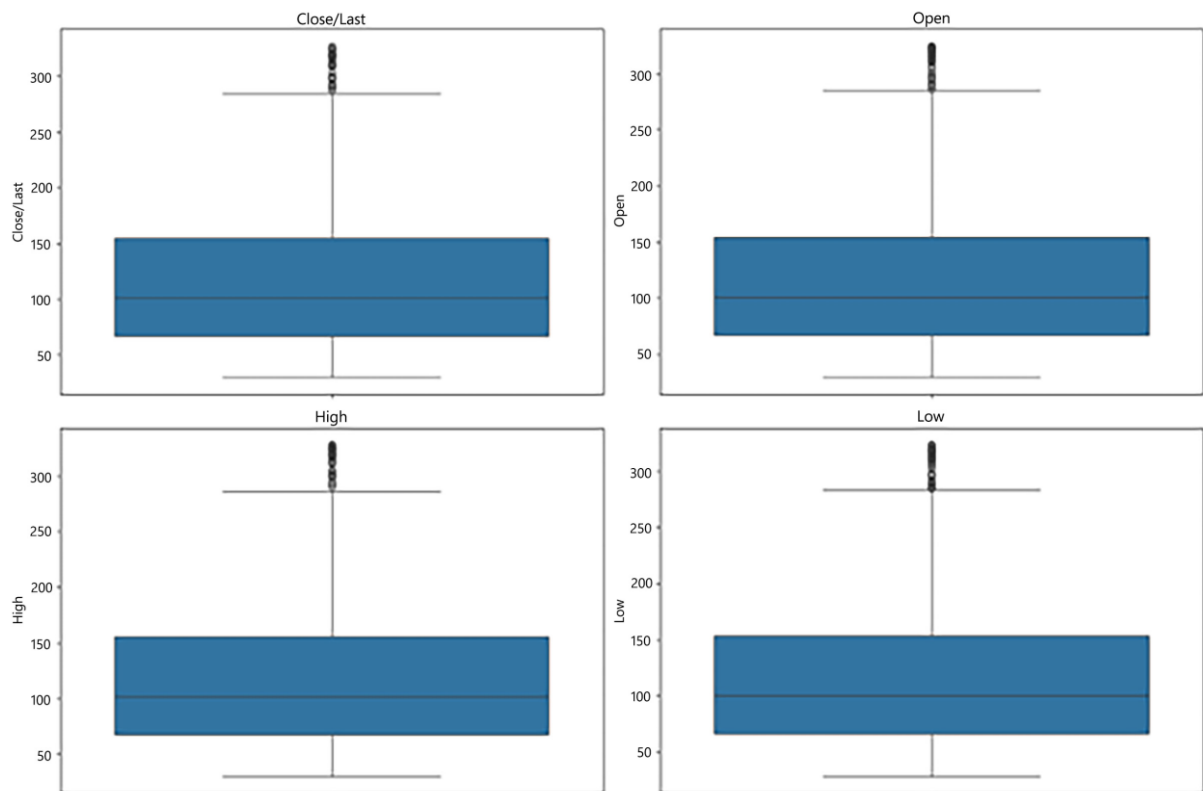


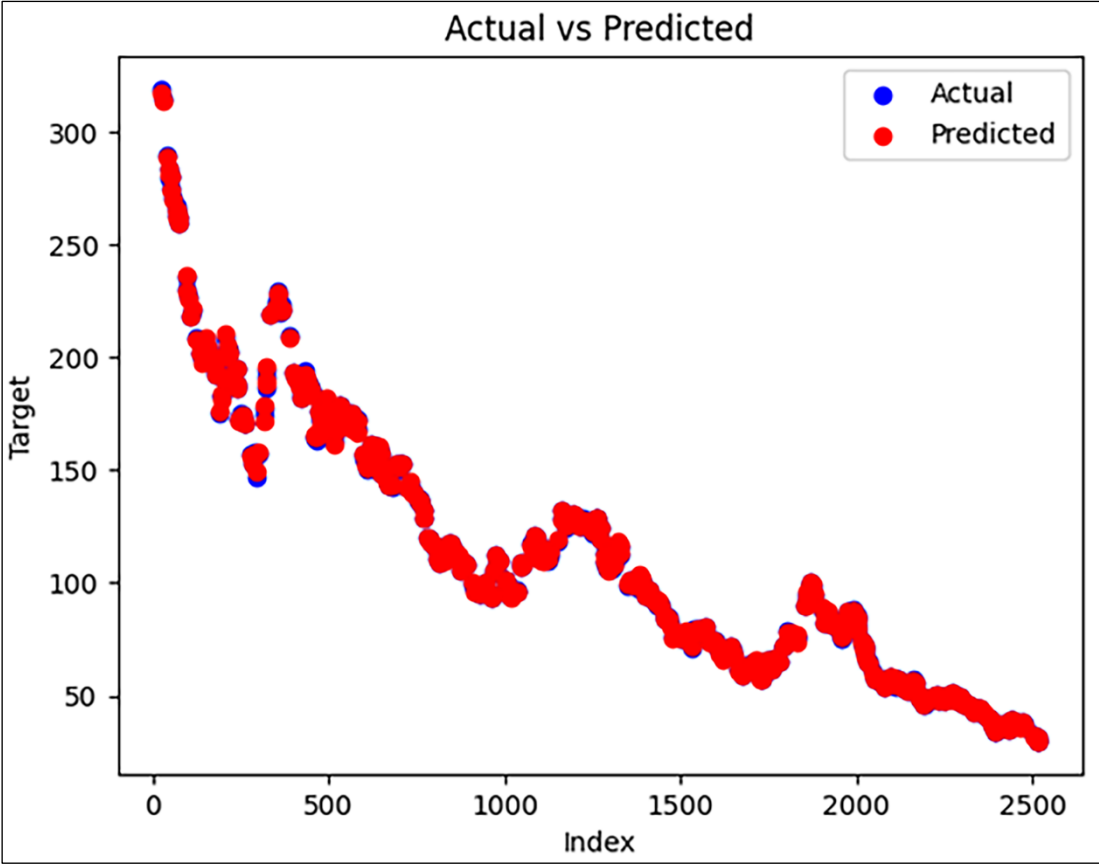




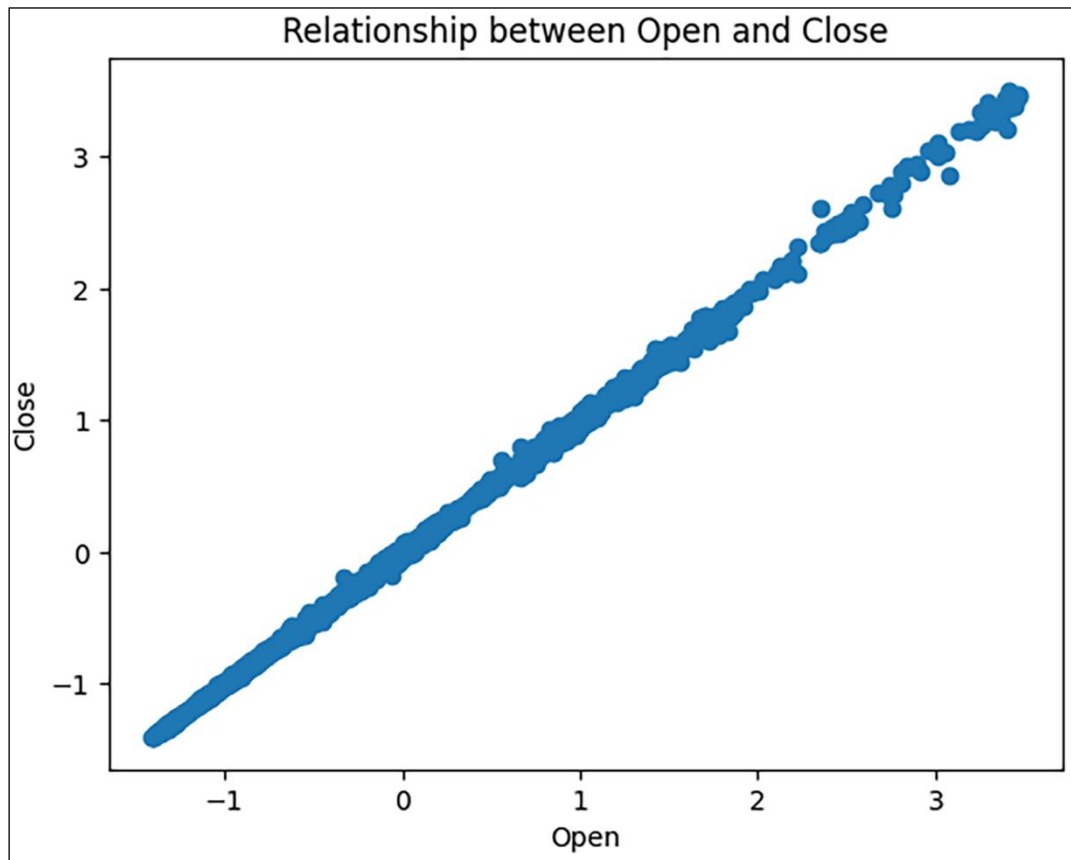


Chapter 18: Regression with Copilot Chat





Chapter 19: Regression with Copilot Suggestions



Chapter 20: Increasing Efficiency with GitHub Copilot



GitHub Copilot

Welcome **@softchris**, I'm your Copilot and I'm here to help you get things done faster. I can identify issues, explain and even improve code.

You can ask generic questions, but what I'm really good at is helping you with your code. For example:

- ✧ Generate unit tests for my code
- ✧ Explain the selected code
- ✧ Propose a fix for the bugs in my code

If you want to learn more about my capabilities and limitations, [check out the Copilot documentation](#).

I'm powered by AI, so surprises and mistakes are possible. Make sure to verify any generated code or suggestions, and [share feedback](#) so that we can learn and improve.

@terminal	Ask how to do something in the terminal
@vscode	Ask about VS Code
/api	@vscode
@workspace	Ask about your workspace
/explain	@workspace
/fix	@workspace
/new	@workspace
/newNotebook	@workspace
/tests	@workspace
/help	
/clear	

@workspace
/newNotebook
products


</> GitHub Copilot used @workspace /newNotebook
👍
💬

Here's an outline for a Jupyter notebook on handling product data:

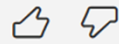
- **Import Required Libraries**
- **Load Product Data**
- **Data Exploration**
- **Data Cleaning**
- **Data Analysis**
- **Visualize Product Sales**

Create Notebook

@workspace improve selected code to validate request body to ensure it contains a valid email address, use primarily functionality from other files in the workspace to solve the problem



GitHub Copilot used @workspace



Used 3 references >

You can modify the `post_customers` function in your `app.py` to use the `validate_email` function from `utils.py`. Here's how you can do it:

```
from flask import request
from utils import validate_email

@app.route('/customers', methods=['POST'])
def post_customers():
    data = request.get_json()

    if 'email' not in data:
        return jsonify({'message': 'No email provided'})

    email = data['email']

    if not validate_email(email):
        return jsonify({'message': 'Invalid email'})

    return jsonify({'message': 'success'})
```



GitHub Copilot used @vscode



You can open a terminal window in VS Code by using the **Terminal: Create New Terminal** command.

Show in Command Palette

Chapter 21: Agents in Software Development

