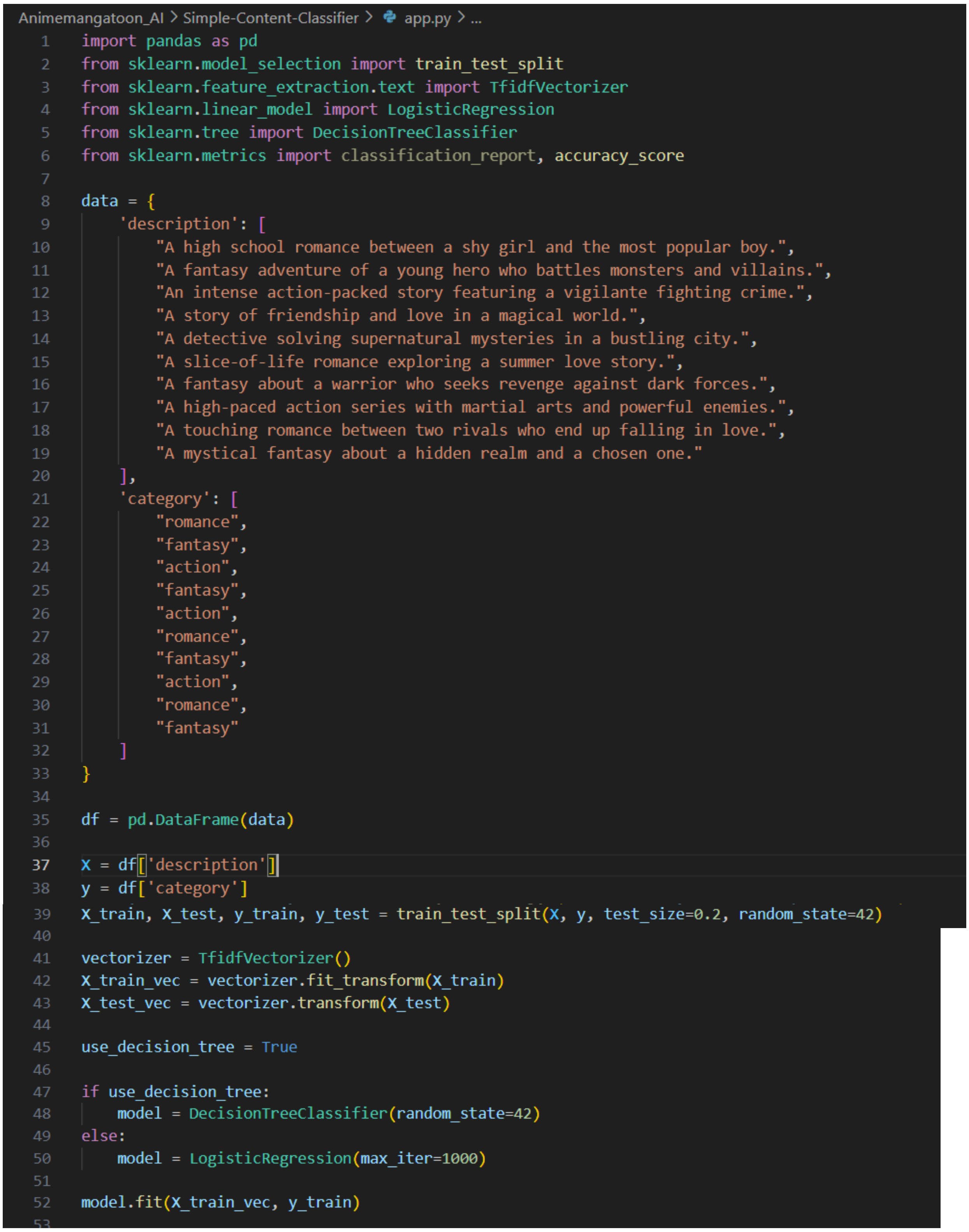
Code Documentation

Task 1: Simple Content Classifier

Data Preparation and Model Training:



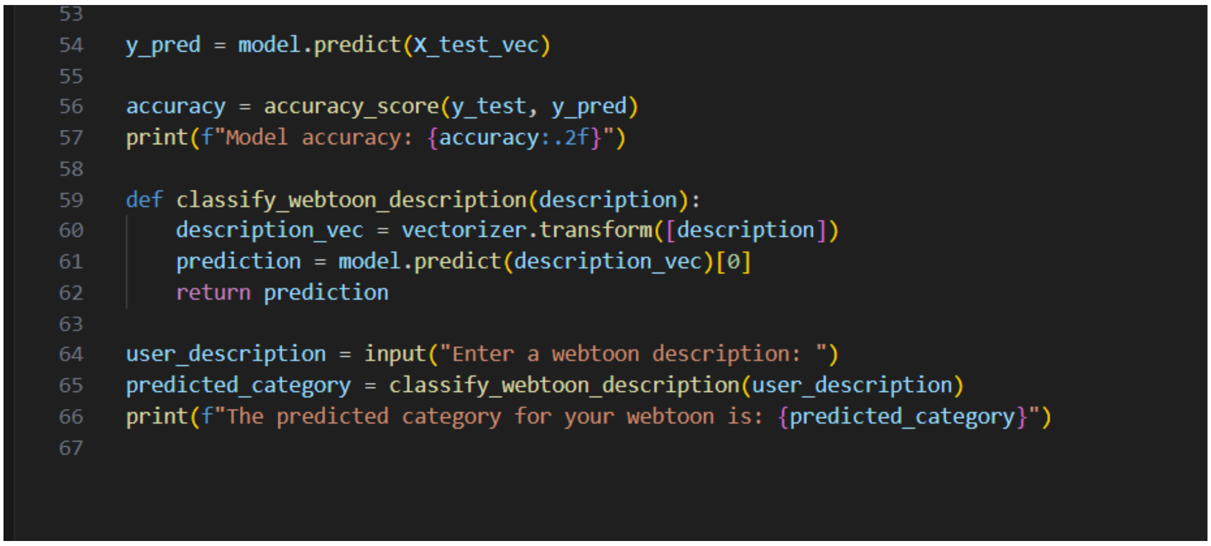
Animemangatoon_AI > Simple-Content-Classifier > app.py > ...

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report, accuracy_score

data = {
    'description': [
        "A high school romance between a shy girl and the most popular boy.",
        "A fantasy adventure of a young hero who battles monsters and villains.",
        "An intense action-packed story featuring a vigilante fighting crime.",
        "A story of friendship and love in a magical world.",
        "A detective solving supernatural mysteries in a bustling city.",
        "A slice-of-life romance exploring a summer love story.",
        "A fantasy about a warrior who seeks revenge against dark forces.",
        "A high-paced action series with martial arts and powerful enemies.",
        "A touching romance between two rivals who end up falling in love.",
        "A mystical fantasy about a hidden realm and a chosen one."
    ],
    'category': [
        "romance",
        "fantasy",
        "action",
        "fantasy",
        "action",
        "romance",
        "fantasy",
        "action",
        "romance",
        "fantasy"
    ]
}

df = pd.DataFrame(data)

X = df['description']
y = df['category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

vectorizer = TfidfVectorizer()
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

use_decision_tree = True

if use_decision_tree:
    model = DecisionTreeClassifier(random_state=42)
else:
    model = LogisticRegression(max_iter=1000)

model.fit(X_train_vec, y_train)
```

Explanation:

1. Data Loading: The script reads a CSV file containing webtoon descriptions and their categories.
2. Feature Extraction: Uses CountVectorizer to transform text descriptions into numerical vectors suitable for model training.
3. Data Splitting: Splits the dataset into training (80%) and testing (20%) sets to evaluate the model's performance.
4. Model Training: Trains a Decision Tree model using the training data.

Model Evaluation and Prediction:



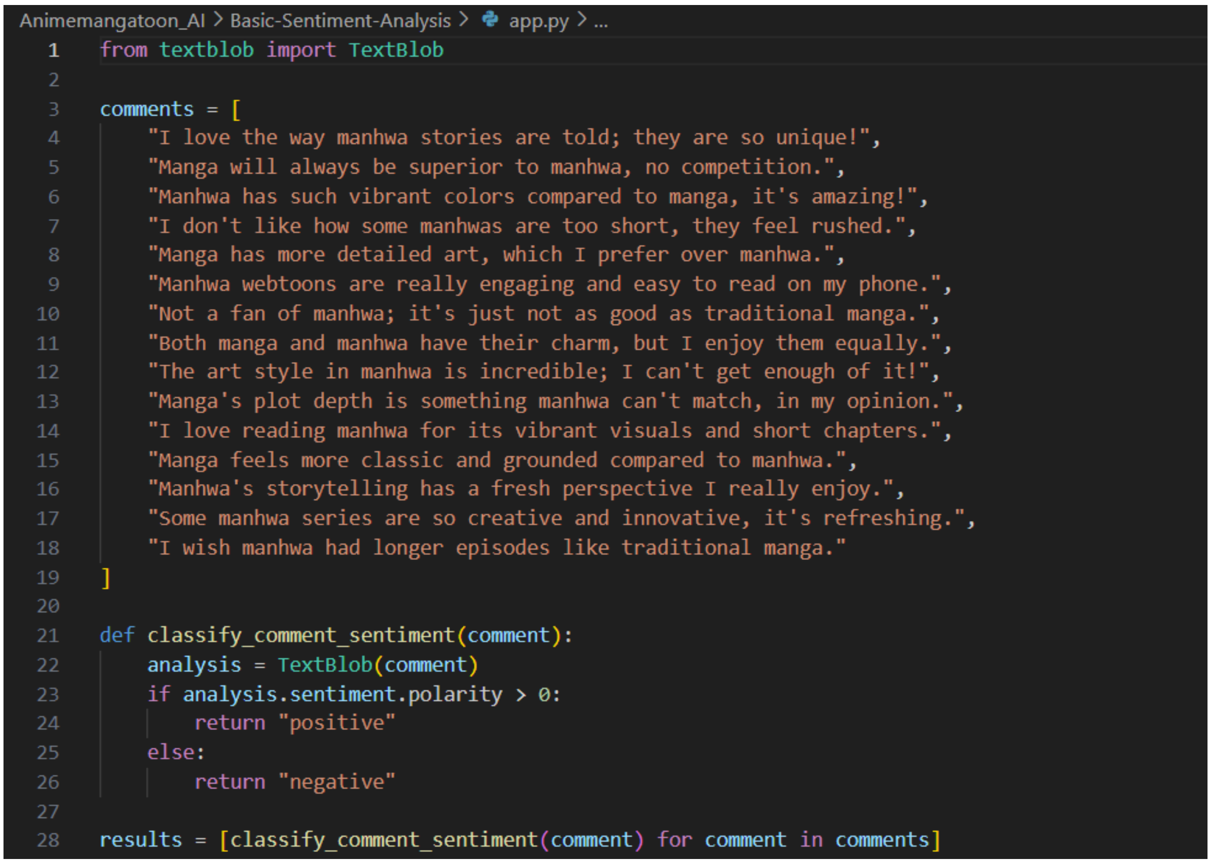
```python

y_pred = model.predict(X_test_vec)

accuracy = accuracy_score(y_test, y_pred)
print(f"Model accuracy: {accuracy:.2f}")

def classify_webtoon_description(description):
    description_vec = vectorizer.transform([description])
    prediction = model.predict(description_vec)[0]
    return prediction

user_description = input("Enter a webtoon description: ")
predicted_category = classify_webtoon_description(user_description)
print(f"The predicted category for your webtoon is: {predicted_category}")

```

Explanation:

1. Model Evaluation: The script evaluates the model using the testing set and calculates the accuracy.
2. Classification Function: Defines a function to classify new user-provided descriptions.
3. Example Usage: The script tests the function with an example description, predicting and displaying its category.

Task 2: Basic Sentiment Analysis on User Comments

Data Loading and Sentiment Analysis:



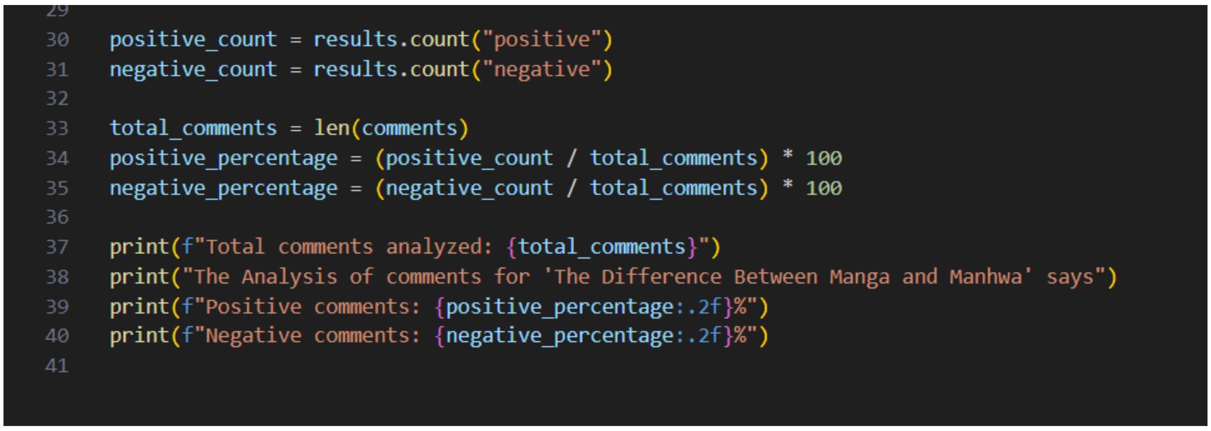
Animemangatoon_AI > Basic-Sentiment-Analysis > app.py > ...

```python
from textblob import TextBlob

comments = [
    "I love the way manhwa stories are told; they are so unique!",
    "Manga will always be superior to manhwa, no competition.",
    "Manhwa has such vibrant colors compared to manga, it's amazing!",
    "I don't like how some manhwas are too short, they feel rushed.",
    "Manga has more detailed art, which I prefer over manhwa.",
    "Manhwa webtoons are really engaging and easy to read on my phone.",
    "Not a fan of manhwa; it's just not as good as traditional manga.",
    "Both manga and manhwa have their charm, but I enjoy them equally.",
    "The art style in manhwa is incredible; I can't get enough of it!",
    "Manga's plot depth is something manhwa can't match, in my opinion.",
    "I love reading manhwa for its vibrant visuals and short chapters.",
    "Manga feels more classic and grounded compared to manhwa.",
    "Manhwa's storytelling has a fresh perspective I really enjoy.",
    "Some manhwa series are so creative and innovative, it's refreshing.",
    "I wish manhwa had longer episodes like traditional manga."
]

def classify_comment_sentiment(comment):
    analysis = TextBlob(comment)
    if analysis.sentiment.polarity > 0:
        return "positive"
    else:
        return "negative"

results = [classify_comment_sentiment(comment) for comment in comments]
```

Explanation:

1. TextBlob Usage: Uses the TextBlob library to perform sentiment analysis on each comment.
2. Function Definition: Defines a function that categorizes comments as "positive" or "negative" based on polarity scores.
3. Results Computation: Applies the function to each comment, storing results for later evaluation.

Summary of Sentiment Results:



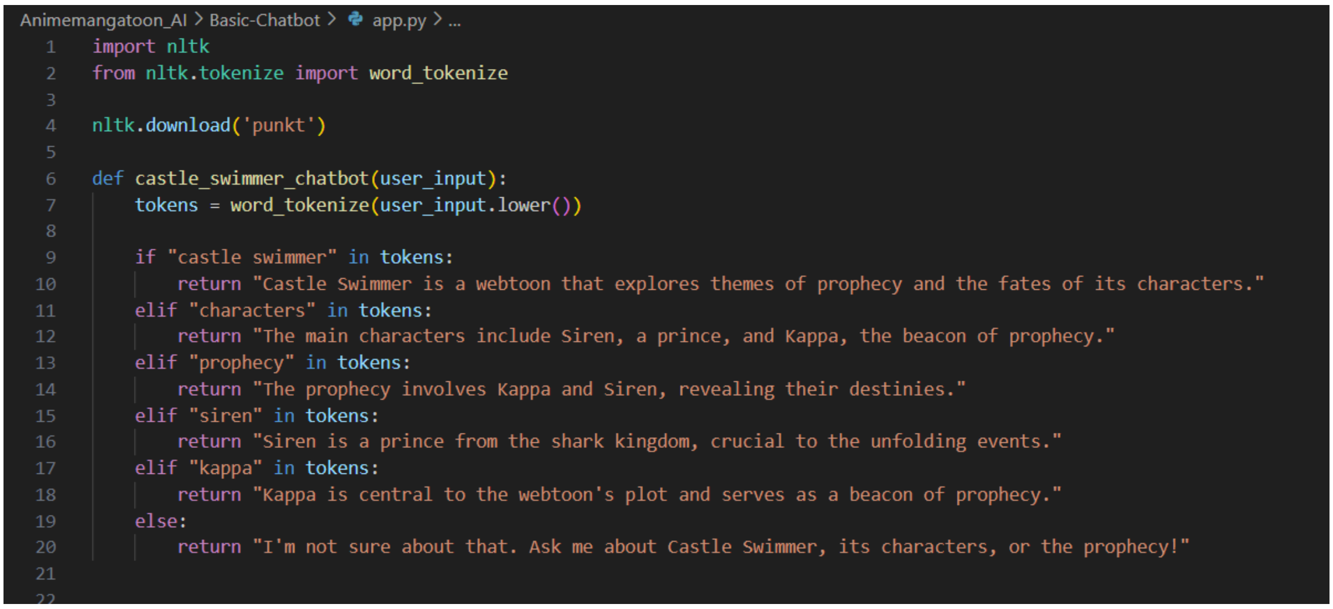
```python
positive_count = results.count("positive")
negative_count = results.count("negative")

total_comments = len(comments)
positive_percentage = (positive_count / total_comments) * 100
negative_percentage = (negative_count / total_comments) * 100

print(f"Total comments analyzed: {total_comments}")
print("The Analysis of comments for 'The Difference Between Manga and Manhwa' says")
print(f"Positive comments: {positive_percentage:.2f}%")
print(f"Negative comments: {negative_percentage:.2f}%")

```

Explanation:

1. Count Sentiment Results: Counts how many comments are classified as positive or negative.
2. Calculate Percentages: Computes the percentage of each sentiment type.
3. Summary Output: Displays the overall distribution of positive and negative comments.

Task 3: Basic Chatbot for Castle Swimmer

Importing Libraries and Setting Up the Chatbot

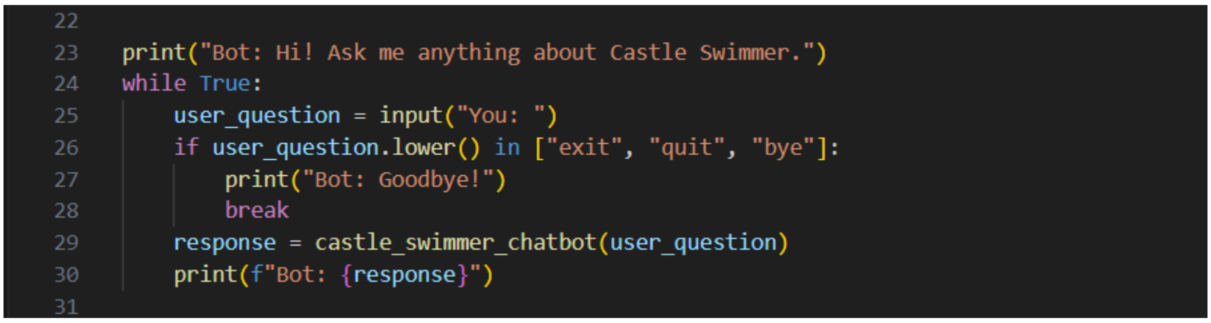

Animemangatoon_AI > Basic-Chatbot > app.py > ...

```python
import nltk
from nltk.tokenize import word_tokenize

nltk.download('punkt')

def castle_swimmer_chatbot(user_input):
    tokens = word_tokenize(user_input.lower())

    if "castle swimmer" in tokens:
        return "Castle Swimmer is a webtoon that explores themes of prophecy and the fates of its characters."
    elif "characters" in tokens:
        return "The main characters include Siren, a prince, and Kappa, the beacon of prophecy."
    elif "prophecy" in tokens:
        return "The prophecy involves Kappa and Siren, revealing their destinies."
    elif "siren" in tokens:
        return "Siren is a prince from the shark kingdom, crucial to the unfolding events."
    elif "kappa" in tokens:
        return "Kappa is central to the webtoon's plot and serves as a beacon of prophecy."
    else:
        return "I'm not sure about that. Ask me about Castle Swimmer, its characters, or the prophecy!"
```

Explanation:

1. Import Libraries: Imports NLTK and its word_tokenize function to tokenize the user's input.
2. Tokenization: Tokenizes the user input into individual words for efficient keyword matching.
3. Keyword-Based Responses: Uses tokenized input to detect keywords and provides predefined responses based on these keywords.
4. Default Response: Returns a default message if none of the keywords match.

Chat Loop and User Interaction:



```python

print("Bot: Hi! Ask me anything about Castle Swimmer.")
while True:
    user_question = input("You: ")
    if user_question.lower() in ["exit", "quit", "bye"]:
        print("Bot: Goodbye!")
        break
    response = castle_swimmer_chatbot(user_question)
    print(f"Bot: {response}")

```

Explanation:

1. Greeting Message: Displays a greeting message when the chatbot starts.
2. User Input Loop: The loop continues until the user types “exit,” “quit,” or “bye” to exit the conversation.
3. Calling the Chatbot Function: The chatbot function is called with the user’s input, and the response is printed out.
4. Response Output: Displays the chatbot's response, maintaining a conversation-like format.