Rameesa Nadeem

Fa21-Bds-024

BIG DATA ANALYSIS PROJECT

On Textual data scrapped and ingested from website hubspot

Installing Pyspark to run it on colab because gradio doesnot work on databricks

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!pip install pyspark
!pip install findspark
```

```
Requirement already satisfied: pyspark in /usr/local/lib/python3.11/dist-packages (3.5.4)
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.11/dist-packages (from pyspark) (0.10.9.7)
Requirement already satisfied: findspark in /usr/local/lib/python3.11/dist-packages (2.0.1)
```

Enusuring latest version is installed

```
!pip show pyspark
```

```
Name: pyspark
Version: 3.5.4
Summary: Apache Spark Python API
Home-page: https://github.com/apache/spark/tree/master/python
Author: Spark Developers
Author-email: dev@spark.apache.org
License: http://www.apache.org/licenses/LICENSE-2.0
Location: /usr/local/lib/python3.11/dist-packages
Requires: py4j
```

```
Required-by:
```

Setting Java Path

```
import os
import findspark

os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/usr/local/lib/python3.11/dist-packages/pyspark"  # Update this path

findspark.init()
```

Creating Spark Session

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("ColabPySpark").getOrCreate()

spark
```

SparkSession - in-memory

SparkContext

Spark UI

```
Version
    v3.5.4
Master
    local[*]
AppName
    ColabPySpark
```

Scrapping and Ingesting the data from the hubspot website

```
import requests
from bs4 import BeautifulSoup
import pandas as pd

url = "https://blog.hubspot.com/marketing"
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")

# Extract relevant content (e.g., headlines)
data = {"content": [item.get_text(strip=True) for item in soup.find_all("h2")]}
df = pd.DataFrame(data)

# Save the data as a CSV locally
df.to_csv("marketing_data.csv", index=False)
print("Scraped data saved as 'marketing_data.csv'")
```

```
Scraped data saved as 'marketing_data.csv'
```

Few Columns of csv looks like this

```
data_path = "marketing_data.csv"
data = spark.read.csv(data_path, header=True, inferSchema=True)

data.show(truncate=False)
```

```
+---------------------------------------------------------------+
|content                                                        |
+---------------------------------------------------------------+
|6 Steps to Create an Outstanding Marketing Plan [Free Templa...|
|Featured Articles                                              |
|Latest articles                                                |
|Artificial Intelligence                                        |
|Instagram Marketing                                            |
|From HubSpot's video library                                   |
|Marketing Strategy                                             |
|From the HubSpot Podcast Network                               |
|More content                                                   |
|Visit the HubSpot blogs                                        |
|Subscribe to HubSpot's Newsletters                             |
|Join 600,000+ Fellow MarketersThanks for Subscribing!          |
|Popular Features                                               |
|Free Tools                                                     |
|Company                                                        |
|Customers                                                      |
|Partners                                                       |
+---------------------------------------------------------------+
```

Data Preprocessing

Removing Punctuation converting to lower case and tokenizing the text

```
from pyspark.sql.functions import col, lower, regexp_replace, split
cleaned_data = (
    data.withColumn("content", lower(col("content")))
         .withColumn("content", regexp_replace(col("content"), "[^a-zA-Z\s]", ""))
         .withColumn("tokens", split(col("content"), r"\s+"))
)

cleaned_data.show(truncate=False)
```

```
+----------------------------------------------------------+---------------------------------------------------------------------+
|content                                                   |tokens                                                               |
+----------------------------------------------------------+---------------------------------------------------------------------+
| steps to create an outstanding marketing plan free templa|[, steps, to, create, an, outstanding, marketing, plan, free, templa]|
|featured articles                                         |[featured, articles]                                                 |
|latest articles                                           |[latest, articles]                                                   |
|artificial intelligence                                   |[artificial, intelligence]                                           |
|instagram marketing                                       |[instagram, marketing]                                               |
|from hubspots video library                               |[from, hubspots, video, library]                                     |
|marketing strategy                                        |[marketing, strategy]                                                |
|from the hubspot podcast network                          |[from, the, hubspot, podcast, network]                               |
|more content                                              |[more, content]                                                      |
|visit the hubspot blogs                                   |[visit, the, hubspot, blogs]                                         |
|subscribe to hubspots newsletters                         |[subscribe, to, hubspots, newsletters]                               |
|join  fellow marketersthanks for subscribing              |[join, fellow, marketersthanks, for, subscribing]                    |
|popular features                                          |[popular, features]                                                  |
|free tools                                                |[free, tools]                                                        |
|company                                                   |[company]                                                            |
|customers                                                  |[customers]                                                           |
|partners                                                   |[partners]                                                            |
+-----------------------------------------------------------+----------------------------------------------------------------------+
```

Counting Words frequency

```
from pyspark.sql.functions import explode
word_freq = (
    cleaned_data.withColumn("word", explode(col("tokens")))
                .groupBy("word")
                .count()
                .orderBy("count", ascending=False)
)

word_freq.show()
```

```
+---------------+-----+
|           word|count|
+---------------+-----+
|      marketing|    3|
|       articles|    2|
|           free|    2|
|       hubspots|    2|
|            the|    2|
|           from|    2|
|        hubspot|    2|
|             to|    2|
|      instagram|    1|
|        popular|    1|
|           more|    1|
|    subscribing|    1|
|            for|    1|
|    outstanding|    1|
|        library|    1|
|        content|    1|
|       featured|    1|
|marketersthanks|    1|
|         create|    1|
|       partners|    1|
+---------------+-----+
only showing top 20 rows
```

› Statistical Analysis

[ ] ↳ **2 cells hidden**

› Machine Learning Algorithms

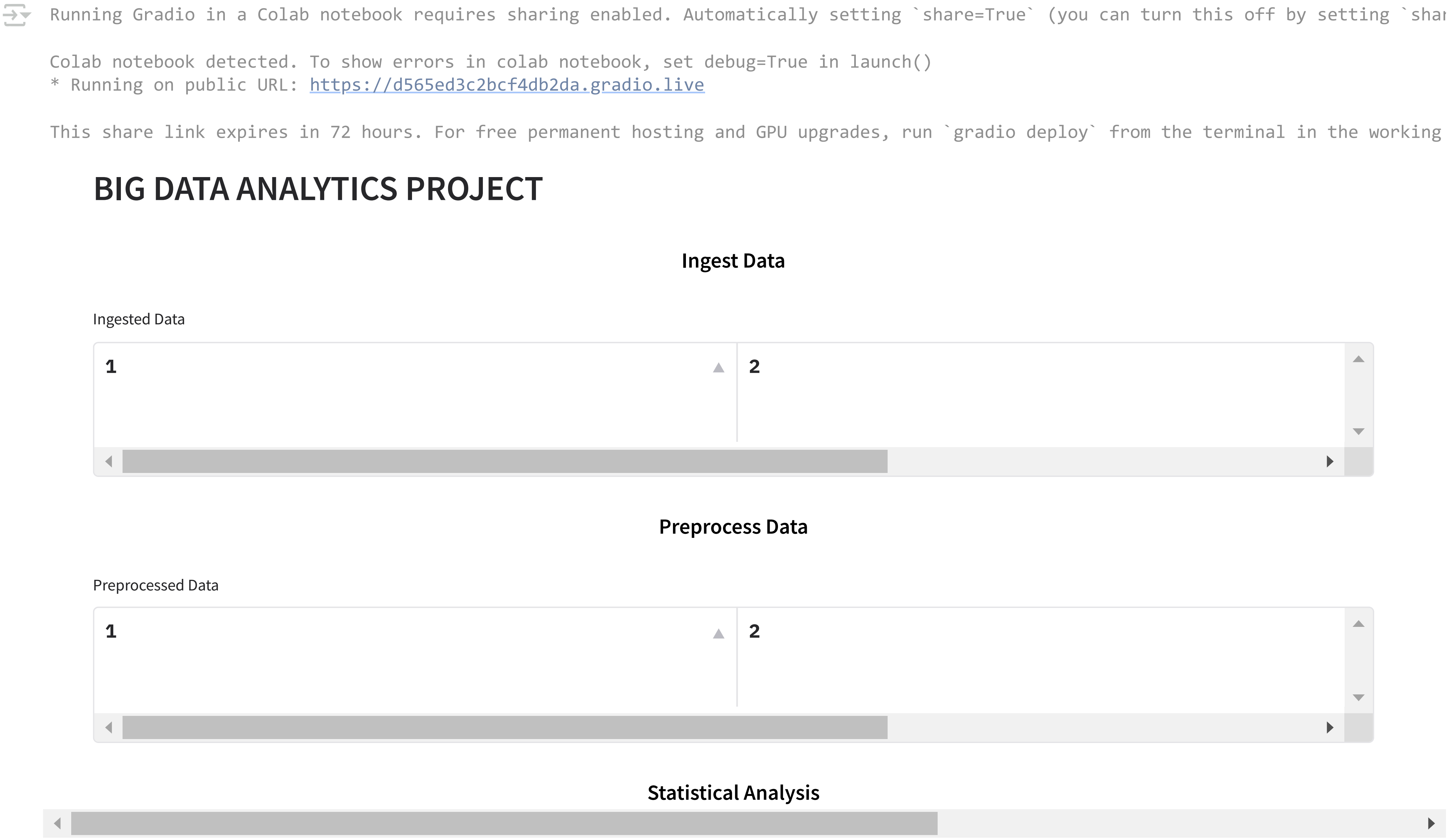
[ ] ↳ **17 cells hidden**

⌄ Integrating the Code with Frontend

Frontend is made using Gradio

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lower, regexp_replace, split, explode, length
from pyspark.ml.feature import HashingTF, IDF, VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.regression import LinearRegression
from pyspark.ml.classification import LinearSVC
from pyspark.ml import Pipeline
from sklearn.cluster import DBSCAN
from sklearn.feature_extraction.text import TfidfVectorizer
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Embedding, GlobalAveragePooling1D
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import gradio as gr
import pandas as pd
import requests
from bs4 import BeautifulSoup
from bs4 import BeautifulSoup
import numpy as np
import gym

# Initialize Spark
spark = SparkSession.builder.appName("ColabPySpark").getOrCreate()

# Data ingestion function
def ingest_data():
    url = "https://blog.coupler.io/marketing-data-analytics/"
    response = requests.get(url)
    soup = BeautifulSoup(response.content, "html.parser")
    data = {"content": [item.get_text(strip=True) for item in soup.find_all("h2")]}
    return pd.DataFrame(data)

# Data preprocessing function
def preprocess_data():
    df = ingest_data()
    spark_df = spark.createDataFrame(df)
    cleaned_data = (
        spark_df.withColumn("content", lower(col("content")))
                .withColumn("content", regexp_replace(col("content"), "[^a-zA-Z\\s]", ""))
                .withColumn("tokens", split(col("content"), "\\s+"))
    )
    return cleaned_data
from pyspark.sql.functions import size

def compute_statistics(cleaned_data):
    word_freq = (
        cleaned_data.withColumn("word", explode(col("tokens")))
                    .groupBy("word")
                    .count()
    )
    variance = word_freq.selectExpr("VAR_SAMP(count) as variance").first()["variance"]

    # Add content_length and word_count columns
    cleaned_data = cleaned_data.withColumn("content_length", length(col("content")))
    cleaned_data = cleaned_data.withColumn("word_count", size(col("tokens")))  # Use size() for array length

    # Compute correlation
    correlation = cleaned_data.stat.corr("content_length", "word_count")
    return f"Variance: {variance}", f"Correlation: {correlation}"
    return f"Variance: {variance}", f"Correlation: {correlation}"

# K-Means Clustering
def run_kmeans(cleaned_data):
    hashing_tf = HashingTF(inputCol="tokens", outputCol="raw_features", numFeatures=100)
    idf = IDF(inputCol="raw_features", outputCol="features")
    kmeans = KMeans(featuresCol="features", k=5)
    pipeline = Pipeline(stages=[hashing_tf, idf, kmeans])
    model = pipeline.fit(cleaned_data)
    clusters = model.transform(cleaned_data).select("content", "prediction")
    return clusters.toPandas()

# Linear Regression
def run_linear_regression(cleaned_data):
    hashing_tf = HashingTF(inputCol="tokens", outputCol="raw_features", numFeatures=100)
    idf = IDF(inputCol="raw_features", outputCol="features")
    cleaned_data = cleaned_data.withColumn("target", (length(col("content")) % 100))
    assembler = VectorAssembler(inputCols=["features"], outputCol="final_features")
    lr = LinearRegression(featuresCol="final_features", labelCol="target")
    pipeline = Pipeline(stages=[hashing_tf, idf, assembler, lr])
    model = pipeline.fit(cleaned_data)
    predictions = model.transform(cleaned_data).select("content", "target", "prediction")
    return predictions.toPandas()

from pyspark.ml.classification import LogisticRegression
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

def run_classification(cleaned_data):
    # Feature extraction (TF-IDF)
    hashing_tf = HashingTF(inputCol="tokens", outputCol="raw_features", numFeatures=100)
    idf = IDF(inputCol="raw_features", outputCol="features")

    # Add mock labels for classification (e.g., 1 for even-length content, 0 for odd-length content)
    cleaned_data = cleaned_data.withColumn("label", (length(col("content")) % 2).cast("double"))

    # Train-test split
    train_data, test_data = cleaned_data.randomSplit([0.8, 0.2], seed=42)

    # Logistic Regression model
    lr = LogisticRegression(featuresCol="features", labelCol="label", maxIter=10)

    # Build pipeline
    # Build pipeline
    pipeline = Pipeline(stages=[hashing_tf, idf, lr])

    # Train model
    model = pipeline.fit(train_data)

    # Make predictions
    predictions = model.transform(test_data).select("content", "label", "prediction")

    # Evaluate the model
    evaluator = MulticlassClassificationEvaluator(
        labelCol="label", predictionCol="prediction", metricName="accuracy"
    )
    accuracy = evaluator.evaluate(predictions)

    return predictions.toPandas(), f"Accuracy: {accuracy}"

# SVM Classification
def run_svm(cleaned_data):
    hashing_tf = HashingTF(inputCol="tokens", outputCol="raw_features", numFeatures=100)
    idf = IDF(inputCol="raw_features", outputCol="features")
    cleaned_data = cleaned_data.withColumn("label", (length(col("content")) % 2).cast("double"))
    svm = LinearSVC(featuresCol="features", labelCol="label", maxIter=10)
    pipeline = Pipeline(stages=[hashing_tf, idf, svm])
    model = pipeline.fit(cleaned_data)
    predictions = model.transform(cleaned_data).select("content", "label", "prediction")
    return predictions.toPandas()

# DBSCAN Clustering
def run_dbscan(cleaned_data):
    texts = cleaned_data.select("content").rdd.flatMap(lambda x: x).collect()
    vectorizer = TfidfVectorizer(max_features=100)
    X = vectorizer.fit_transform(texts).toarray()
    dbscan = DBSCAN(eps=0.5, min_samples=5)
    labels = dbscan.fit_predict(X)
    return pd.DataFrame({"Text": texts, "Cluster": labels})

def run_deep_learning(cleaned_data):
    texts = cleaned_data.select("content").rdd.flatMap(lambda x: x).collect()
    if not texts:  # Check if the dataset is empty
        return pd.DataFrame({"Error": ["No data available for deep learning."]})
    labels = [1 if len(text) % 2 == 0 else 0 for text in texts]
    tokenizer = Tokenizer(num_words=5000)
    tokenizer.fit_on_texts(texts)
    X = pad_sequences(tokenizer.texts_to_sequences(texts), maxlen=100)

    # Ensure the input data is float32
    X = np.array(X, dtype="float32")
    labels = np.array(labels, dtype="float32")

    model = Sequential([
        Embedding(input_dim=5000, output_dim=32, input_length=100),
        GlobalAveragePooling1D(),
        Dense(16, activation='relu'),
        Dense(1, activation='sigmoid')
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

    try:
        model.fit(X, labels, epochs=5, batch_size=32, verbose=0)
        predictions = model.predict(X)
        return pd.DataFrame({"Text": texts, "Prediction": predictions.flatten()})
    except Exception as e:
        return pd.DataFrame({"Error": [str(e)]})


def run_reinforcement_learning():
    env = gym.make("CartPole-v1", new_step_api=True)
    action_space = env.action_space.n
    state_space = [20] * env.observation_space.shape[0]
    q_table = np.zeros(state_space + [action_space])
    alpha, gamma, epsilon = 0.1, 0.9, 0.1  # Learning rate, discount factor, exploration rate

    def discretize_state(state, bins):
        state_bounds = list(zip(env.observation_space.low, env.observation_space.high))
        state_bounds[1] = [-3.5, 3.5]  # Clip velocity
        state_bounds[3] = [-3.5, 3.5]  # Clip angular velocity
        ratios = [(state[i] - state_bounds[i][0]) / (state_bounds[i][1] - state_bounds[i][0]) for i in range(len(state))]
        discrete_state = [int(r * (b - 1)) for r, b in zip(ratios, bins)]
        return tuple(np.clip(discrete_state, 0, np.array(bins) - 1))

    for episode in range(500):  # Increased episodes for better training
    for episode in range(500):  # Increased episodes for better training
        state = discretize_state(env.reset(), state_space)
        done = False
        while not done:
            # Epsilon-greedy action selection
            if np.random.rand() < epsilon:
                action = np.random.choice(action_space)  # Explore
            else:
                action = np.argmax(q_table[state])  # Exploit

            # Perform action
            next_state_raw, reward, terminated, truncated, _ = env.step(action)
            next_state = discretize_state(next_state_raw, state_space)
            done = terminated or truncated

            # Reward scaling
            reward = reward if not terminated else -100  # Penalize for termination

            # Update Q-value
            best_next_action = np.max(q_table[next_state])
            q_table[state][action] += alpha * (reward + gamma * best_next_action - q_table[state][action])

            # Move to the next state
            state = next_state

    if np.any(q_table):
        return q_table.tolist()
    else:
        return "Q-Table generation failed. Check reward structure or state discretization."



import matplotlib.pyplot as plt
import seaborn as sns
from io import BytesIO
import base64

# Visualization: Word Frequency Distribution
def plot_word_frequencies(cleaned_data):
    word_freq = (
        cleaned_data.withColumn("word", explode(col("tokens")))
```

```
                    .groupBy("word")
                    .count()
                    .orderBy("count", ascending=False)
    ).toPandas()

    plt.figure(figsize=(10, 6))
    sns.barplot(x="count", y="word", data=word_freq.head(10), palette="viridis")
    plt.title("Top 10 Word Frequencies")
    plt.xlabel("Frequency")
    plt.ylabel("Words")
    return plt.gcf()

# Visualization: K-Means Clusters
def plot_kmeans_clusters(kmeans_result):
    plt.figure(figsize=(10, 6))
    sns.countplot(x="prediction", data=kmeans_result, palette="viridis")
    plt.title("K-Means Cluster Distribution")
    plt.xlabel("Cluster")
    plt.ylabel("Frequency")
    return plt.gcf()

# Visualization: DBSCAN Clusters
def plot_dbscan_clusters(dbscan_result):
    plt.figure(figsize=(10, 6))
    sns.countplot(x="Cluster", data=dbscan_result, palette="viridis")
    plt.title("DBSCAN Cluster Distribution")
    plt.xlabel("Cluster")
    plt.ylabel("Frequency")
    return plt.gcf()

from sklearn.decomposition import LatentDirichletAllocation
from sklearn.feature_extraction.text import CountVectorizer

def topic_modeling(cleaned_data):
    texts = cleaned_data.select("content").rdd.flatMap(lambda x: x).collect()
    vectorizer = CountVectorizer(max_features=1000, stop_words="english")
    X = vectorizer.fit_transform(texts)
    lda = LatentDirichletAllocation(n_components=5, random_state=42)
    lda.fit(X)
    topics = lda.components_
    words = vectorizer.get_feature_names_out()
    topic_words = [[words[i] for i in topic.argsort()[-10:]] for topic in topics]
    return pd.DataFrame({"Topic": range(1, 6), "Top Words": topic_words})


# Visualization: Reinforcement Learning Q-Table Heatmap
def plot_q_table(q_table):
    plt.figure(figsize=(12, 8))
    sns.heatmap(q_table, annot=False, cmap="viridis")
    plt.title("Reinforcement Learning Q-Table Heatmap")
    plt.xlabel("Actions")
    plt.ylabel("States")
    return plt.gcf()
# Gradio App
with gr.Blocks() as demo:
    gr.Markdown("# BIG DATA ANALYTICS PROJECT")

    # Section 1: Data Ingestion
    with gr.Row():
        with gr.Column():
            ingest_btn = gr.Button("Ingest Data")
            ingest_output = gr.Dataframe(label="Ingested Data")
            ingest_btn.click(lambda: ingest_data(), inputs=[], outputs=[ingest_output])

    # Section 2: Data Preprocessing
    with gr.Row():
        with gr.Column():
            preprocess_btn = gr.Button("Preprocess Data")
            preprocess_output = gr.Dataframe(label="Preprocessed Data")
            preprocess_btn.click(lambda: preprocess_data().toPandas(), inputs=[], outputs=[preprocess_output])

    # Section 3: Statistical Analysis
    with gr.Row():
        with gr.Column():
            stats_btn = gr.Button("Statistical Analysis")
            stats_output = gr.Textbox(label="Statistical Analysis")
            stats_btn.click(lambda: compute_statistics(preprocess_data()), inputs=[], outputs=[stats_output])

    # Section 4: Machine Learning Models
    with gr.Row():
        with gr.Column():
            kmeans_btn = gr.Button("Run K-Means Clustering")
            kmeans_output = gr.Dataframe(label="K-Means Clusters")
            kmeans_output = gr.Dataframe(label="K-Means Clusters")
            kmeans_btn.click(lambda: run_kmeans(preprocess_data()), inputs=[], outputs=[kmeans_output])

        with gr.Column():
            regression_btn = gr.Button("Run Linear Regression")
            regression_output = gr.Dataframe(label="Linear Regression Results")
            regression_btn.click(lambda: run_linear_regression(preprocess_data()), inputs=[], outputs=[regression_output])

    with gr.Row():
        with gr.Column():
            svm_btn = gr.Button("Run SVM Classification")
            svm_output = gr.Dataframe(label="SVM Predictions")
            svm_btn.click(lambda: run_svm(preprocess_data()), inputs=[], outputs=[svm_output])

        with gr.Column():
            dbscan_btn = gr.Button("Run DBSCAN Clustering")
            dbscan_output = gr.Dataframe(label="DBSCAN Clusters")
            dbscan_btn.click(lambda: run_dbscan(preprocess_data()), inputs=[], outputs=[dbscan_output])
    with gr.Row():
        with gr.Column():
            classification_btn = gr.Button("Run Classification Model")
            classification_output = gr.Dataframe(label="Classification Predictions")
            classification_accuracy = gr.Textbox(label="Classification Accuracy")
            classification_btn.click(
                lambda: run_classification(preprocess_data()),
                inputs=[],
                outputs=[classification_output, classification_accuracy]
            )

        with gr.Column():
            topic_btn = gr.Button("Run Topic Modeling")
            topic_output = gr.Dataframe(label="Topic Modeling")
            topic_btn.click(lambda: topic_modeling(preprocess_data()), inputs=[], outputs=[topic_output])



    # Section 5: Deep Learning and Reinforcement Learning
    with gr.Row():
        with gr.Column():
            deeplearning_btn = gr.Button("Run Deep Learning")
            deeplearning_output = gr.Dataframe(label="Deep Learning Predictions")
            deeplearning_btn.click(lambda: run_deep_learning(preprocess_data()), inputs=[], outputs=[deeplearning_output])
            deeplearning_btn.click(lambda: run_deep_learning(preprocess_data()), inputs=[], outputs=[deeplearning_output])

        with gr.Column():
            rl_btn = gr.Button("Run Reinforcement Learning")
            rl_output = gr.Textbox(label="Reinforcement Learning Q-Table")
            rl_btn.click(run_reinforcement_learning, inputs=[], outputs=[rl_output])

    # Section 6: Visualizations
    with gr.Row():
        with gr.Column():
            word_freq_btn = gr.Button("Visualize Word Frequencies")
            word_freq_plot = gr.Plot(label="Word Frequencies")
            word_freq_btn.click(lambda: plot_word_frequencies(preprocess_data()), inputs=[], outputs=[word_freq_plot])

        with gr.Column():
            kmeans_vis_btn = gr.Button("Visualize K-Means Clusters")
            kmeans_plot = gr.Plot(label="K-Means Cluster Distribution")
            kmeans_vis_btn.click(lambda: plot_kmeans_clusters(run_kmeans(preprocess_data())), inputs=[], outputs=[kmeans_plot])

        with gr.Column():
            dbscan_vis_btn = gr.Button("Visualize DBSCAN Clusters")
            dbscan_plot = gr.Plot(label="DBSCAN Cluster Distribution")
            dbscan_vis_btn.click(lambda: plot_dbscan_clusters(run_dbscan(preprocess_data())), inputs=[], outputs=[dbscan_plot])

        with gr.Column():
            q_table_vis_btn = gr.Button("Visualize Q-Table")
            q_table_plot = gr.Plot(label="Q-Table Heatmap")
            q_table_vis_btn.click(lambda: plot_q_table(run_reinforcement_learning()), inputs=[], outputs=[q_table_plot])

demo.launch()
```



```
Running Gradio in a Colab notebook requires sharing enabled. Automatically setting `share=True` (you can turn this off by setting `shar

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://d565ed3c2bcf4db2da.gradio.live

This share link expires in 72 hours. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```

BIG DATA ANALYTICS PROJECT

Ingest Data

Ingested Data

1	2

Preprocess Data

Preprocessed Data

1	2

Statistical Analysis