Welcome to Colab!

Explore the Gemini API

The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.

How to get started?

- Go to Google AI Studio and log in with your Google account.
- Create an API key.
- Use a quickstart for Python, or call the REST API using curl.

Discover Gemini's advanced capabilities

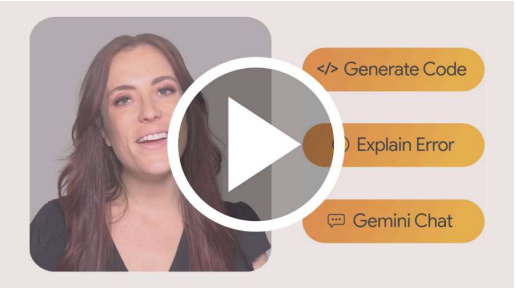
- Play with Gemini multimodal outputs, mixing text and images in an iterative way.
- Discover the multimodal Live API (demo here).
- Learn how to analyze images and detect items in your pictures using Gemini (bonus, there's a 3D version as well!).
- Unlock the power of Gemini thinking model, capable of solving complex task with its inner thoughts.

Explore complex use cases

- Use Gemini grounding capabilities to create a report on a company based on what the model can find on internet.
- Extract invoices and form data from PDF in a structured way.
- Create illustrations based on a whole book using Gemini large context window and Imagen.

To learn more, check out the Gemini cookbook or visit the Gemini API documentation.

Colab now has AI features powered by Gemini. The video below provides information on how to use these features, whether you're new to Python, or a seasoned veteran.



What is Colab?

Colab, or "Colaboratory", allows you to write and execute Python in your browser, with

- Zero configuration required
- Access to GPUs free of charge
- Easy sharing

Whether you're a **student**, a **data scientist** or an **AI researcher**, Colab can make your work easier. Watch Introduction to Colab or Colab Features You May Have Missed to learn more, or just get started below!

Getting started

The document you are reading is not a static web page, but an interactive environment called a **Colab notebook** that lets you write and execute code.

For example, here is a **code cell** with a short Python script that computes a value, stores it in a variable, and prints the result:

```
seconds_in_a_day = 24 * 60 * 60
seconds_in_a_day
```

```
86400
```

To execute the code in the above cell, select it with a click and then either press the play button to the left of the code, or use the keyboard shortcut "Command/Ctrl+Enter". To edit the code, just click the cell and start editing.

Variables that you define in one cell can later be used in other cells:

```
seconds_in_a_week = 7 * seconds_in_a_day
seconds_in_a_week
```

```
604800
```

Colab notebooks allow you to combine **executable code** and **rich text** in a single document, along with **images**, **HTML**, **LaTeX** and more. When you create your own Colab notebooks, they are stored in your Google Drive account. You can easily share your Colab notebooks with co-workers or friends, allowing them to comment on your notebooks or even edit them. To learn more, see Overview of Colab. To create a new Colab notebook you can use the File menu above, or use the following link: create a new Colab notebook.

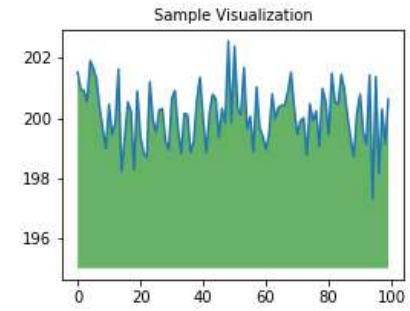
Colab notebooks are Jupyter notebooks that are hosted by Colab. To learn more about the Jupyter project, see jupyter.org.

Data science

With Colab you can harness the full power of popular Python libraries to analyze and visualize data. The code cell below uses **numpy** to generate some random data, and uses **matplotlib** to visualize it. To edit the code, just click the cell and start editing.

You can import your own data into Colab notebooks from your Google Drive account, including from spreadsheets, as well as from Github and many other sources. To learn more about importing data, and how Colab can be used for data science, see the links below under Working with Data.

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64

ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]

fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)

data = io.BytesIO()
plt.savefig(data)
image = F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
```



Colab notebooks execute code on Google's cloud servers, meaning you can leverage the power of Google hardware, including GPUs and TPUs, regardless of the power of your machine. All you need is a browser.

For example, if you find yourself waiting for **pandas** code to finish running and want to go faster, you can switch to a GPU Runtime and use libraries like RAPIDS cuDF that provide zero-code-change acceleration.

To learn more about accelerating pandas on Colab, see the 10 minute guide or US stock market data analysis demo.

Machine learning

With Colab you can import an image dataset, train an image classifier on it, and evaluate the model, all in just a few lines of code.

Double-click (or enter) to edit

Colab is used extensively in the machine learning community with applications including:

- Getting started with TensorFlow
- Developing and training neural networks
- Experimenting with TPUs
- Disseminating AI research
- Creating tutorials

To see sample Colab notebooks that demonstrate machine learning applications, see the machine learning examples below.

More Resources

Working with Notebooks in Colab

- Overview of Colab
- Guide to Markdown
- Importing libraries and installing dependencies
- Saving and loading notebooks in GitHub
- Interactive forms
- Interactive widgets

Working with Data

- Loading data: Drive, Sheets, and Google Cloud Storage
- Charts: visualizing data
- Getting started with BigQuery

Machine Learning

These are a few of the notebooks related to Machine Learning, including Google's online Machine Learning course. See the full course website for more.

- Intro to Pandas DataFrame
- Intro to RAPIDS cuDF to accelerate pandas
- Getting Started with cuML's accelerator mode
- Linear regression with tf.keras using synthetic data

Using Accelerated Hardware

- TensorFlow with GPUs
- TPUs in Colab

Featured examples

- Retraining an Image Classifier: Build a Keras model on top of a pre-trained image classifier to distinguish flowers.
- Text Classification: Classify IMDB movie reviews as either *positive* or *negative*.
- Style Transfer: Use deep learning to transfer style between images.
- Multilingual Universal Sentence Encoder Q&A: Use a machine learning model to answer questions from the SQuAD dataset.
- Video Interpolation: Predict what happened in a video between the first and the last frame.

```
#image captioning and segmentation

import numpy as np
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import LSTM, Dense, Embedding, Input
import matplotlib.pyplot as plt # Import matplotlib.pyplot as plt here

# Load VGG16 model for feature extraction
base_model = VGG16(weights='imagenet', include_top=False)
model = Model(inputs=base_model.input, outputs=base_model.output)

# Function to extract image features
def extract_features(image):
    image = tf.image.resize(image, (224, 224)) / 255.0  # Normalize
    image = np.expand_dims(image, axis=0)
    features = model.predict(image)
    return features

# Define LSTM-based captioning model
image_input = Input(shape=(None, None, 512))
caption_input = Input(shape=(None,))
embedding_layer = Embedding(input_dim=5000, output_dim=256)(caption_input)
lstm_layer = LSTM(512)(embedding_layer)
output_layer = Dense(5000, activation='softmax')(lstm_layer)

captioning_model = Model(inputs=[image_input, caption_input], outputs=output_layer)
captioning_model.compile(optimizer='adam', loss='categorical_crossentropy')

print("Image Captioning Model created successfully!")
```

```
import tensorflow as tf
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Conv2D, UpSampling2D, Input

# Load VGG16 model (pre-trained on ImageNet)
vgg16 = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Extract features from VGG16
inputs = Input(shape=(224, 224, 3))
features = vgg16(inputs)
```

```
# Decoder (segmentation layers)
x = Conv2D(512, (3,3), activation='relu', padding='same')(features)
x = UpSampling2D((2,2))(x)
x = Conv2D(256, (3,3), activation='relu', padding='same')(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(128, (3,3), activation='relu', padding='same')(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(64, (3,3), activation='relu', padding='same')(x)
x = UpSampling2D((2,2))(x)
x = Conv2D(1, (1,1), activation='sigmoid', padding='same')(x)  # Output segmentation mask

# Create model
segmentation_model = Model(inputs, x)

# Compile model
segmentation_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

print("Image Segmentation Model using VGG16 created successfully!")



# Generate dummy segmentation mask (binary example)
segmentation_mask = np.random.randint(0, 2, (224, 224))

# Count pixel distribution
segmented_pixels = np.sum(segmentation_mask)
non_segmented_pixels = segmentation_mask.size - segmented_pixels

# Plot Pie Chart
labels = ['Segmented Area', 'Non-Segmented Area']
sizes = [segmented_pixels, non_segmented_pixels]
colors = ['lightblue', 'gray']

plt.figure(figsize=(6,6))
plt.pie(sizes, labels=labels, autopct='%1.1f%%', colors=colors)
plt.title("Segmentation Area Distribution")
plt.show()

# bar chart
plt.figure(figsize=(8,6))
plt.hist(segmentation_mask.flatten(), bins=2, color='blue', edgecolor='black')
plt.xticks([0,1], ['Non-Segmented', 'Segmented'])
plt.ylabel("Pixel Count")
plt.title("Segmentation Pixel Frequency")
plt.show()

#linechart
plt.figure(figsize=(10,4))
plt.plot(np.sum(segmentation_mask, axis=1), color='green', marker='o')
plt.xlabel("Row Index")
plt.ylabel("Number of Segmented Pixels")
plt.title("Segmentation Pattern Across Image Rows")
plt.grid()
plt.show()

# Bubble chart
x_coords = np.random.randint(0, 224, 50)
y_coords = np.random.randint(0, 224, 50)
sizes = np.random.randint(10, 200, 50)

plt.figure(figsize=(8,6))
plt.scatter(x_coords, y_coords, s=sizes, alpha=0.5, color='purple')
plt.xlabel("X-coordinate")
plt.ylabel("Y-coordinate")
plt.title("Bubble Chart - Segmented Pixel Density")
plt.show()

#Histogram Chart
import numpy as np
import matplotlib.pyplot as plt

# Generate dummy segmentation mask (binary example)
segmentation_mask = np.random.randint(0, 2, (224, 224))  # 0 = background, 1 = segmented object
```

```
segmentation_mask = np.random.randint(0, 2, (224, 224))  # 0 - background, 1 - segmented object

# Flatten the mask for histogram representation
pixel_values = segmentation_mask.flatten()

# Create Histogram
plt.figure(figsize=(8,6))
plt.hist(pixel_values, bins=[-0.5, 0.5, 1.5], color='blue', edgecolor='black', rwidth=0.8)

# Labels and Formatting
plt.xticks([0,1], ['Non-Segmented', 'Segmented'])
plt.xlabel("Pixel Value")
plt.ylabel("Frequency")
plt.title("Histogram of Segmented Pixels")
plt.grid(axis='y')

# Show the plot
plt.show()

# Scatter Plot
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 20, 25, 30, 50]

# Create scatter plot
plt.scatter(x, y, color='blue', label='Data Points')

# Add labels and title
plt.xlabel('X-axis')
plt.ylabel('Y-axis')
plt.title('Simple Scatter Plot')

# Show legend
plt.legend()

# Display the plot
plt.show()

#Area chart
import matplotlib.pyplot as plt

# Sample data
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 30, 25]

# Create an area chart
plt.fill_between(x, y, color="skyblue", alpha=0.5)

# Add labels and title
plt.xlabel("X-axis")
plt.ylabel("Y-axis")
plt.title("Simple Area Chart")

# Display the plot
plt.show()

#box plot

import matplotlib.pyplot as plt
import numpy as np

# Sample data
data = [7, 8, 9, 10, 15, 20, 22, 23, 24, 25, 30, 32, 35, 40, 45]

# Create a box plot
plt.boxplot(data, vert=True, patch_artist=True, boxprops=dict(facecolor="lightblue"))

# Add labels and title
plt.xlabel("Data")
plt.ylabel("Values")
plt.title("Simple Box Plot")
```

```
# Display the plot
plt.show()

#Heatmap

import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

# Sample data: 5x5 matrix
data = np.random.rand(5, 5)

# Create a heatmap
sns.heatmap(data, annot=True, cmap="coolwarm")

# Add title
plt.title("Simple Heatmap")

# Display the plot
plt.show()

#Radar Chart
import numpy as np
import matplotlib.pyplot as plt

# Sample data
categories = ["Speed", "Agility", "Strength", "Endurance", "Flexibility"]
values = [80, 60, 75, 90, 85]

# Convert categorical data to radians
angles = np.linspace(0, 2 * np.pi, len(categories), endpoint=False)

# Complete the loop
values += values[:1]
angles = np.append(angles, angles[0])

# Create plot
fig, ax = plt.subplots(figsize=(6, 6), subplot_kw=dict(polar=True))

ax.fill(angles, values, color="skyblue", alpha=0.4)
ax.plot(angles, values, color="blue", linewidth=2)

# Add category labels
ax.set_xticks(angles[:-1])
ax.set_xticklabels(categories)

# Display the radar chart
plt.title("Simple Radar Chart")
plt.show()

# Time series
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
from prophet import Prophet
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

class StockTimeSeriesAnalysis:
    def __init__(self, ticker='AAPL', start_date='2022-01-01', end_date='2023-01-01'):
        self.ticker = ticker
        self.start_date = start_date
        self.end_date = end_date
        self.data = self.fetch_stock_data()

    def fetch_stock_data(self):
```

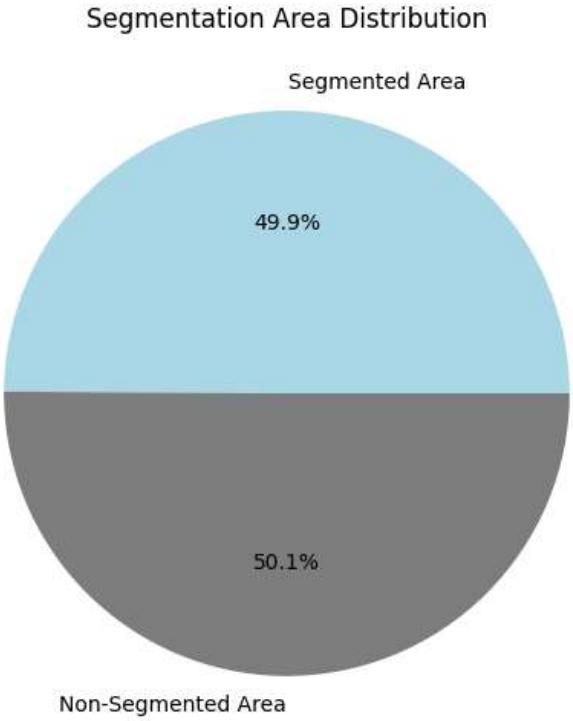
```
    def fetch_stock_data(self):
        """Fetch stock data using yfinance"""
        stock_data = yf.download(self.ticker, start=self.start_date, end=self.end_date)
        return stock_data['Close']

    def prophet_forecast(self):
        """Prophet Forecasting"""
        df = pd.DataFrame({
            'ds': self.data.index,
            # Explicitly flatten the values to ensure it's a 1-dimensional array
            'y': self.data.values.flatten()
        })

        model = Prophet()
        model.fit(df)

        future = model.make_future_dataframe(periods=30)
        forecast = model.predict(future)

        model.plot(forecast)
        plt.title(f'{self.ticker} Prophet Forecast')
        plt.show()
        return forecast

    def arima_forecast(self, order=(1,1,1)):
        """ARIMA Forecasting"""
        model = ARIMA(self.data, order=order)
        results = model.fit()
        forecast = results.forecast(steps=30)

        plt.figure(figsize=(10,6))
        # Ensure the index matches the forecast length for plotting
        plt.plot(self.data.index[-len(forecast):], forecast, label='ARIMA Forecast')
        plt.title(f'{self.ticker} ARIMA Forecast')
        plt.show()
        return forecast

    def lstm_forecast(self, look_back=60):
        """LSTM Neural Network Forecasting"""
        # Prepare data
        scaler = MinMaxScaler(feature_range=(0, 1))
        scaled_data = scaler.fit_transform(self.data.values.reshape(-1, 1))

        X, y = [], []
        for i in range(look_back, len(scaled_data)):
            X.append(scaled_data[i-look_back:i, 0])
            y.append(scaled_data[i, 0])

        X, y = np.array(X), np.array(y)
        X = X.reshape((X.shape[0], X.shape[1], 1))

        # Split train/test
        train_size = int(len(X) * 0.8)
        X_train, X_test = X[:train_size], X[train_size:]
        y_train, y_test = y[:train_size], y[train_size:]

        # Build LSTM model
        model = Sequential([
            LSTM(50, activation='relu', input_shape=(X_train.shape[1], 1)),
            Dense(1)
        ])
        model.compile(optimizer='adam', loss='mse')

        # Train model
        model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

        # Predict
        predictions = model.predict(X_test)
        predictions = scaler.inverse_transform(predictions)

        plt.figure(figsize=(10,6))
        # Correcting the x-axis for plotting actual values to match the length of y_test
        plt.plot(self.data.index[train_size + look_back:train_size + look_back + len(y_test)],
```

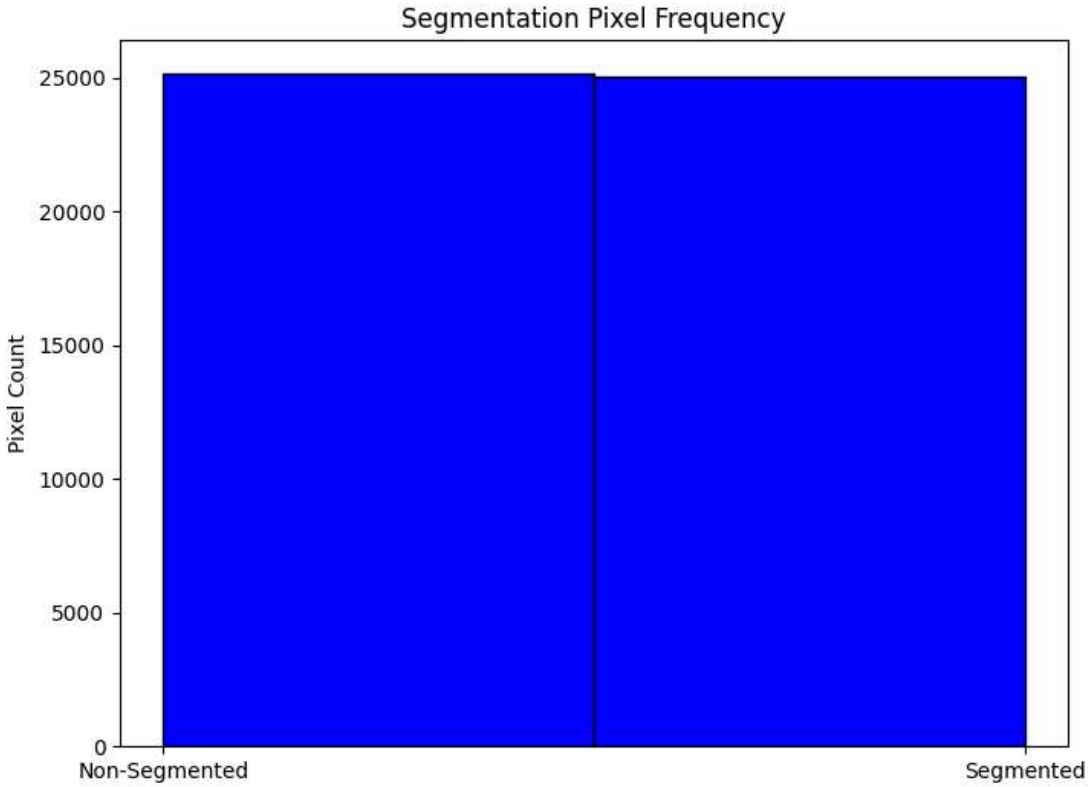
```
                scaler.inverse_transform(y_test.reshape(-1, 1)),
                label='Actual')
        # Correct the x-axis for plotting predicted values to match the length of predictions
        plt.plot(self.data.index[train_size + look_back:],
                predictions,
                label='Predicted')
        plt.title(f'{self.ticker} LSTM Forecast')
        plt.legend()
        plt.show()

        return predictions

# Main execution
def main():
    # Create analysis instance
    analysis = StockTimeSeriesAnalysis(
        ticker='AAPL',
        start_date='2022-01-01',
        end_date='2023-01-01'
    )

    # Run forecasting models
    print("Prophet Forecast:")
    prophet_forecast = analysis.prophet_forecast()

    print("\nARIMA Forecast:")
    arima_forecast = analysis.arima_forecast()

    print("\nLSTM Forecast:")
    lstm_forecast = analysis.lstm_forecast()

if __name__ == "__main__":
    main()


class SARIMAForecasting:
    def __init__(self, data, seasonal_period=12):
        """
        Initialize SARIMA Forecasting Class

        Parameters:
        - data: Time series data
        - seasonal_period: Seasonal periodicity (default 12 for monthly data)
        """
        self.data = data
        self.seasonal_period = seasonal_period

    def check_stationarity(self):
        """
        Check time series stationarity using Augmented Dickey-Fuller test
        """
        from statsmodels.tsa.stattools import adfuller

        result = adfuller(self.data)
        print('ADF Statistic:', result[0])
        print('p-value:', result[1])

        if result[1] <= 0.05:
            print("Data is stationary")
        else:
            print("Data is non-stationary, consider differencing")

    def difference_series(self, difference_order=1):
        """
        Apply differencing to make series stationary

        Parameters:
        - difference_order: Number of differences to apply
        """
        differenced_data = self.data

        for _ in range(difference_order):
```

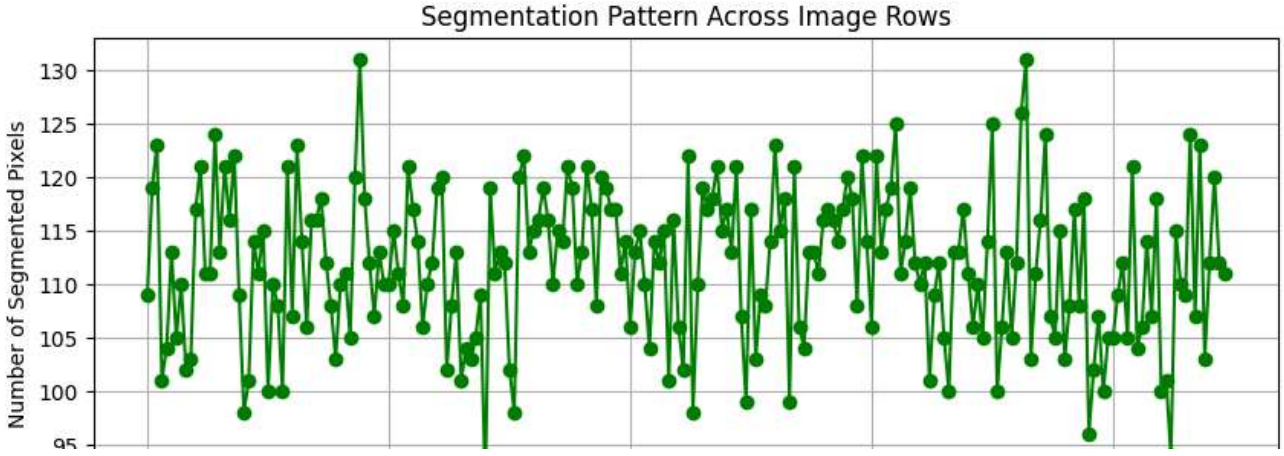
```
            differenced_data = differenced_data.diff().dropna()

        return differenced_data

    def plot_acf_pacf(self):
        """
        Plot Autocorrelation and Partial Autocorrelation Functions
        """
        # Import necessary plotting functions
        from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))

        plot_acf(self.data, ax=ax1)
        plot_pacf(self.data, ax=ax2)

        plt.tight_layout()
        plt.show()

    def fit_sarima_model(self, order=(1,1,1), seasonal_order=(1,1,1,12)):
        """
        Fit SARIMA Model

        Parameters:
        - order: (p,d,q) - Non-seasonal parameters
        - seasonal_order: (P,D,Q,m) - Seasonal parameters
        """
        try:
            model = SARIMAX(
                self.data,
                order=order,
                seasonal_order=seasonal_order
            )

            results = model.fit()
            print(results.summary())

            return results

        except Exception as e:
            print(f"Error fitting SARIMA model: {e}")
            return None

    def forecast(self, model, steps=12):
        """
        Generate SARIMA Forecast

        Parameters:
        - model: Fitted SARIMA model
        - steps: Number of forecast periods
        """
        try:
            forecast = model.get_forecast(steps=steps)
            forecast_mean = forecast.predicted_mean
            forecast_conf_int = forecast.conf_int()

            # Plotting
            plt.figure(figsize=(12, 6))
            plt.plot(self.data.index, self.data, label='Original Data')
            plt.plot(forecast_mean.index, forecast_mean, color='red', label='Forecast')
            plt.fill_between(
                forecast_conf_int.index,
                forecast_conf_int.iloc[:, 0],
                forecast_conf_int.iloc[:, 1],
                color='pink',
                alpha=0.3
            )
            plt.title('SARIMA Forecast')
            plt.legend()
            plt.show()

            return forecast_mean, forecast_conf_int
```

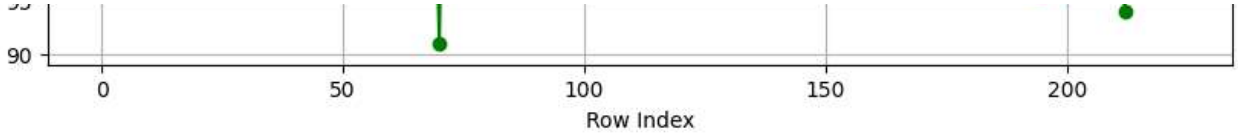
```
            except Exception as e:
                print(f"Forecast error: {e}")
                return None

        def model_evaluation(self, actual, forecast):
            """
            Evaluate model performance

            Parameters:
            - actual: Actual time series values
            - forecast: Predicted values
            """
            # Import mean_absolute_error
            from sklearn.metrics import mean_absolute_error

            mse = mean_squared_error(actual, forecast)
            mae = mean_absolute_error(actual, forecast)
            rmse = np.sqrt(mse)

            print("Model Performance Metrics:")
            print(f"Mean Squared Error (MSE): {mse}")
            print(f"Mean Absolute Error (MAE): {mae}")
            f"Root Mean Squared Error (RMSE): {rmse}" # Removed the print statement here

    def main():
        # Example usage with sample data
        # Replace with your actual time series data
        np.random.seed(42)
        dates = pd.date_range(start='2020-01-01', end='2022-12-31', freq='M')
        data = pd.Series(
            np.cumsum(np.random.randn(len(dates))) + 10,
            index=dates
        )

        # Initialize SARIMA Forecasting
        sarima_forecast = SARIMAForecasting(data)

        # Check stationarity
        sarima_forecast.check_stationarity()

        # Plot ACF and PACF
        sarima_forecast.plot_acf_pacf()

        # Fit SARIMA Model
        model = sarima_forecast.fit_sarima_model(
            order=(1,1,1),           # Non-seasonal parameters
            seasonal_order=(1,1,1,12) # Seasonal parameters
        )

        # Generate Forecast
        if model:
            forecast_mean, forecast_conf_int = sarima_forecast.forecast(model, steps=12)

            # Model Evaluation
            sarima_forecast.model_evaluation(
                actual=data[-12:],    # Last 12 actual values
                forecast=forecast_mean
            )

    if __name__ == "__main__":
        main()
```

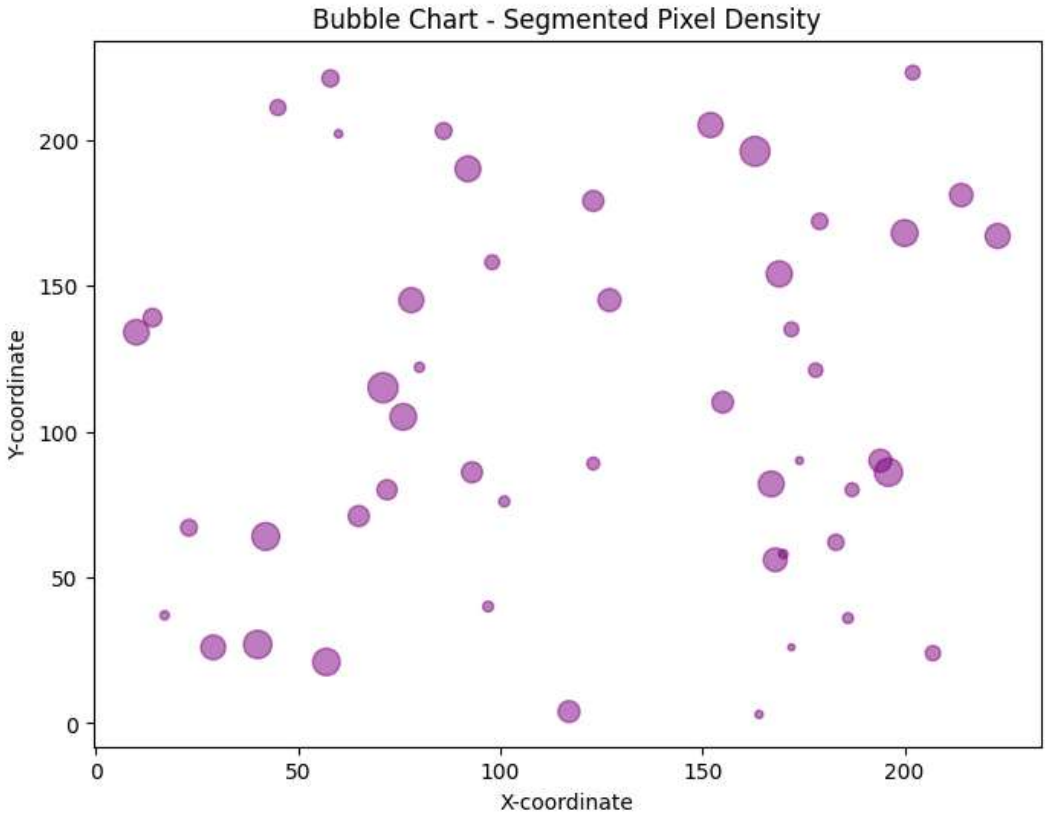
```
Image Captioning Model created successfully!
Image Segmentation Model using VGG16 created successfully!
```

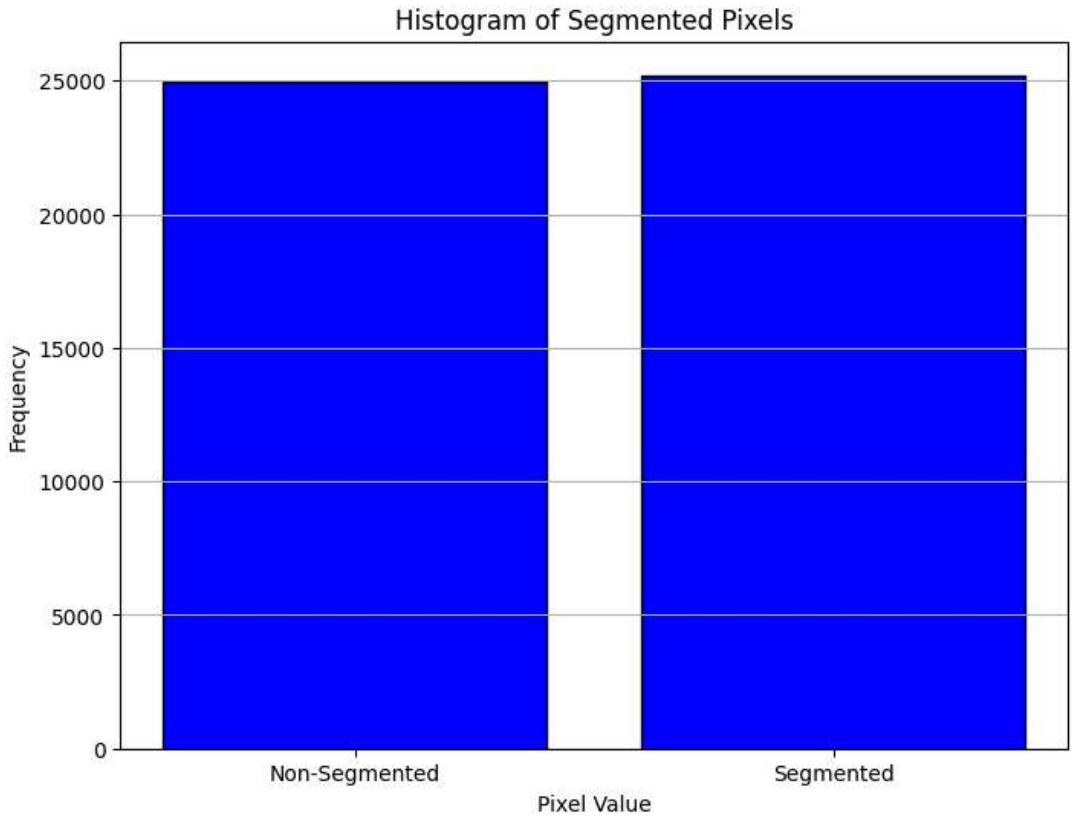


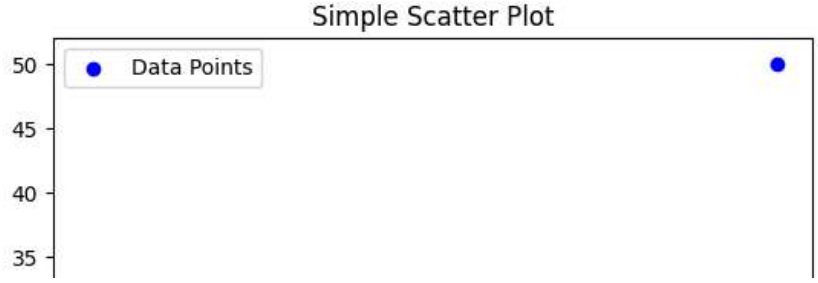


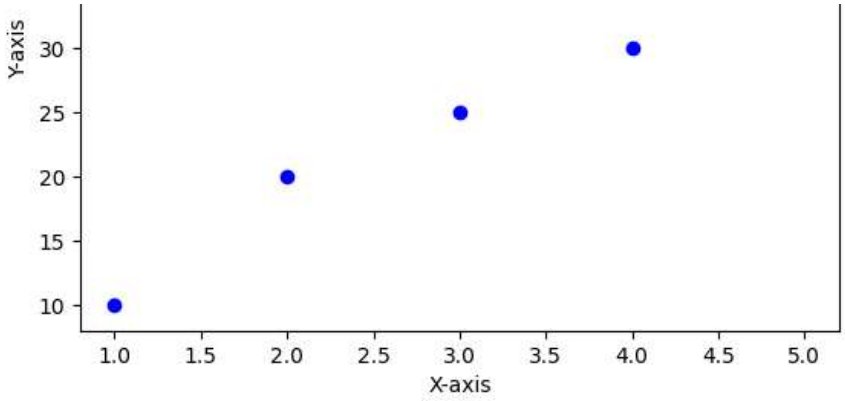


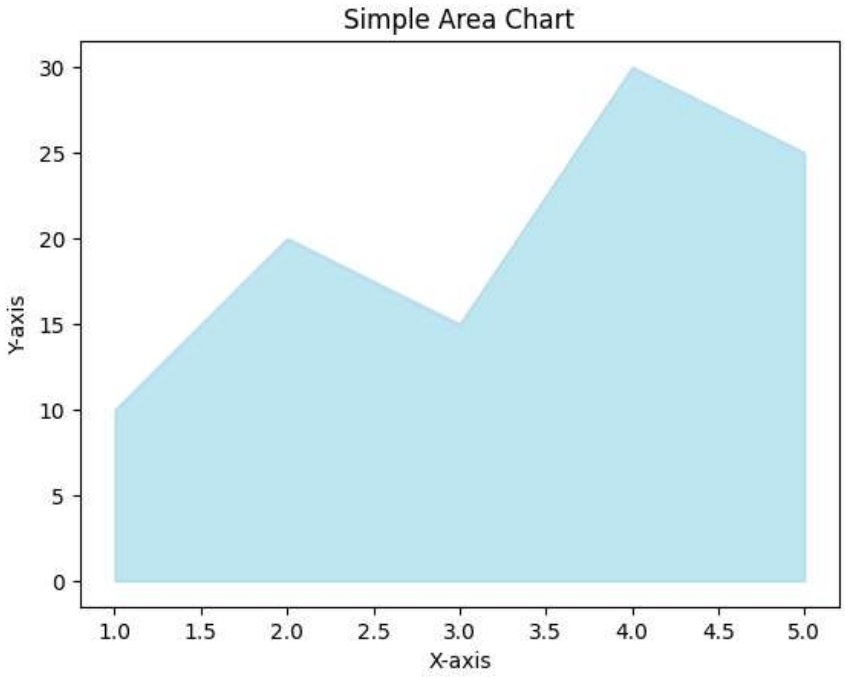

90
0
50
100
150
200
Row Index

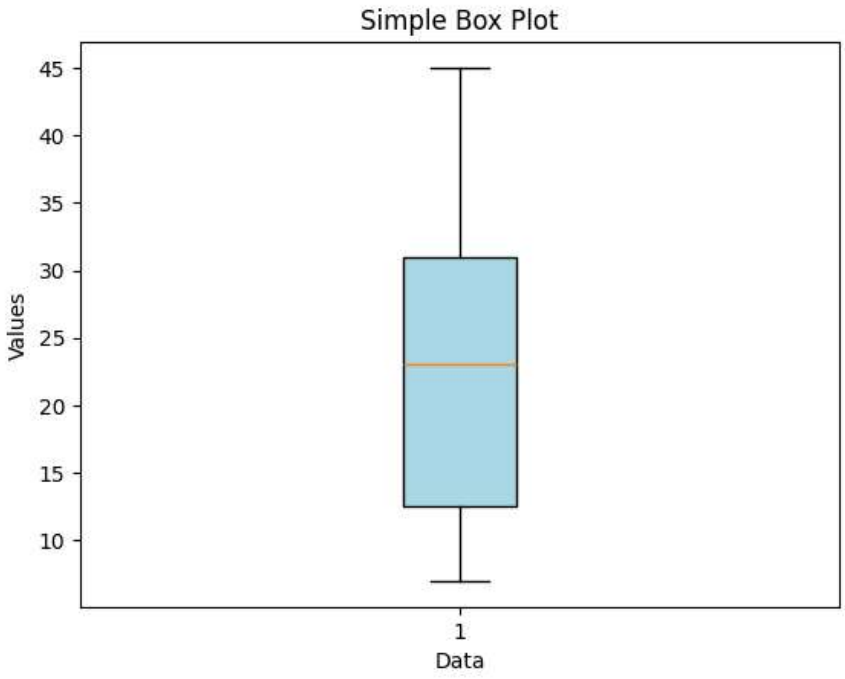

Bubble Chart - Segmented Pixel Density
Y-coordinate
X-coordinate
0
50
100
150
200


Histogram of Segmented Pixels
Frequency
Pixel Value
Non-Segmented
Segmented
0
5000
10000
15000
20000
25000

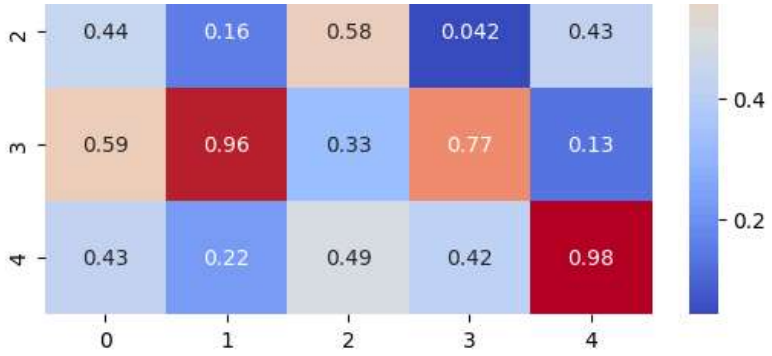

Simple Scatter Plot
Data Points
50
45
40
35

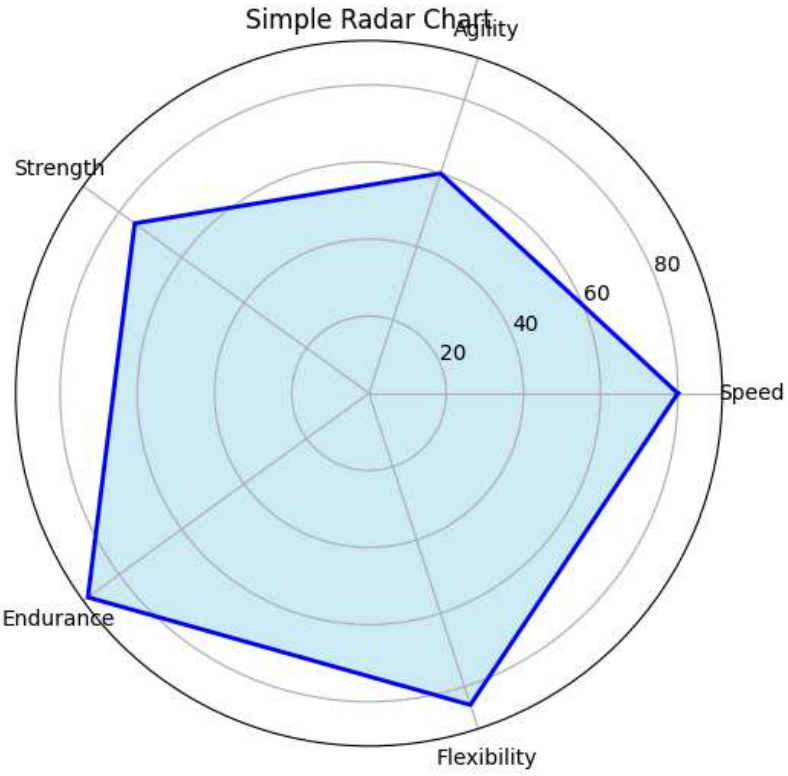

Y-axis
30
25
20
15
10
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
X-axis


Simple Area Chart
Y-axis
30
25
20
15
10
5
0
1.0
1.5
2.0
2.5
3.0
3.5
4.0
4.5
5.0
X-axis

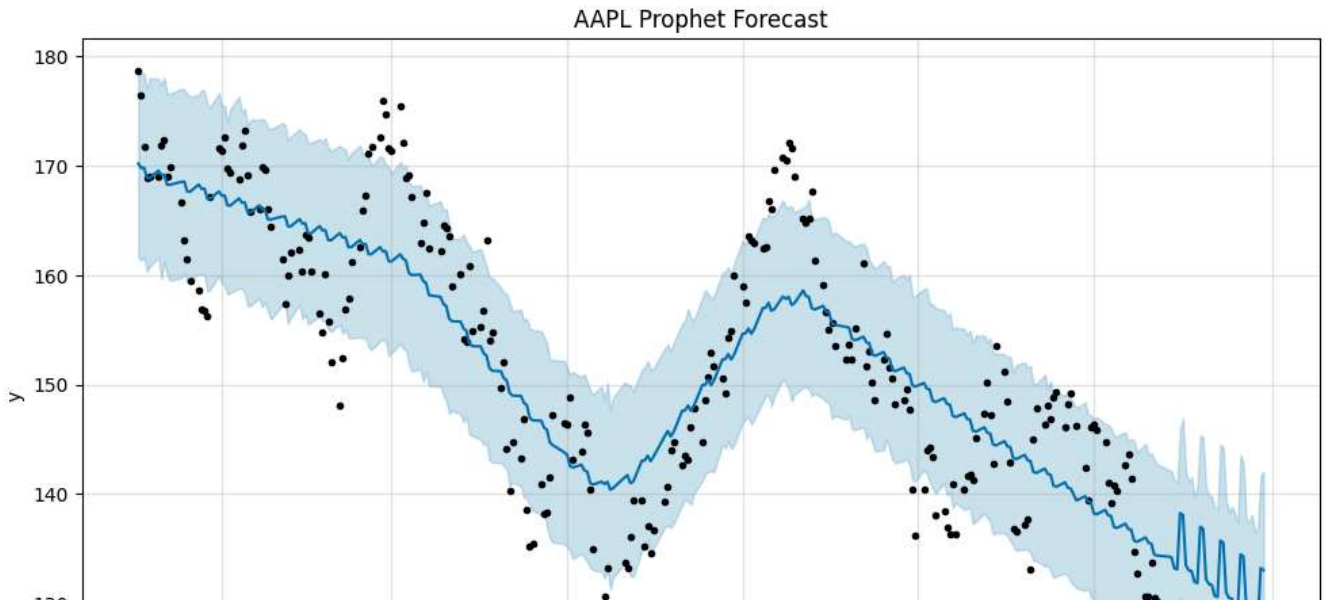

Simple Box Plot
Values
45
40
35
30
25
20
15
10
1
Data

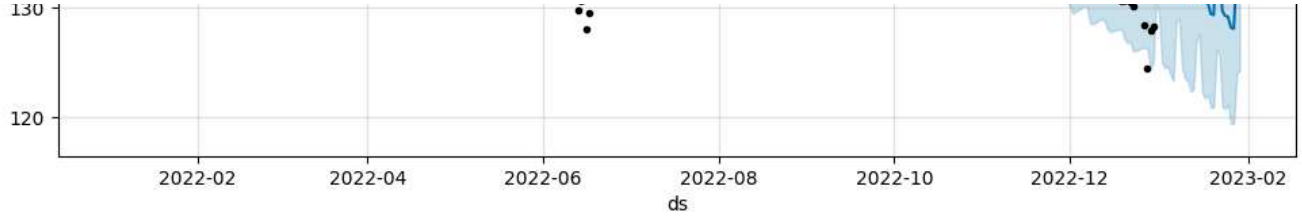

Simple Heatmap
0
1
0.96
0.18
0.15
0.36
0.096
0.16
0.85
0.17
0.11
0.88
0.8
0.6



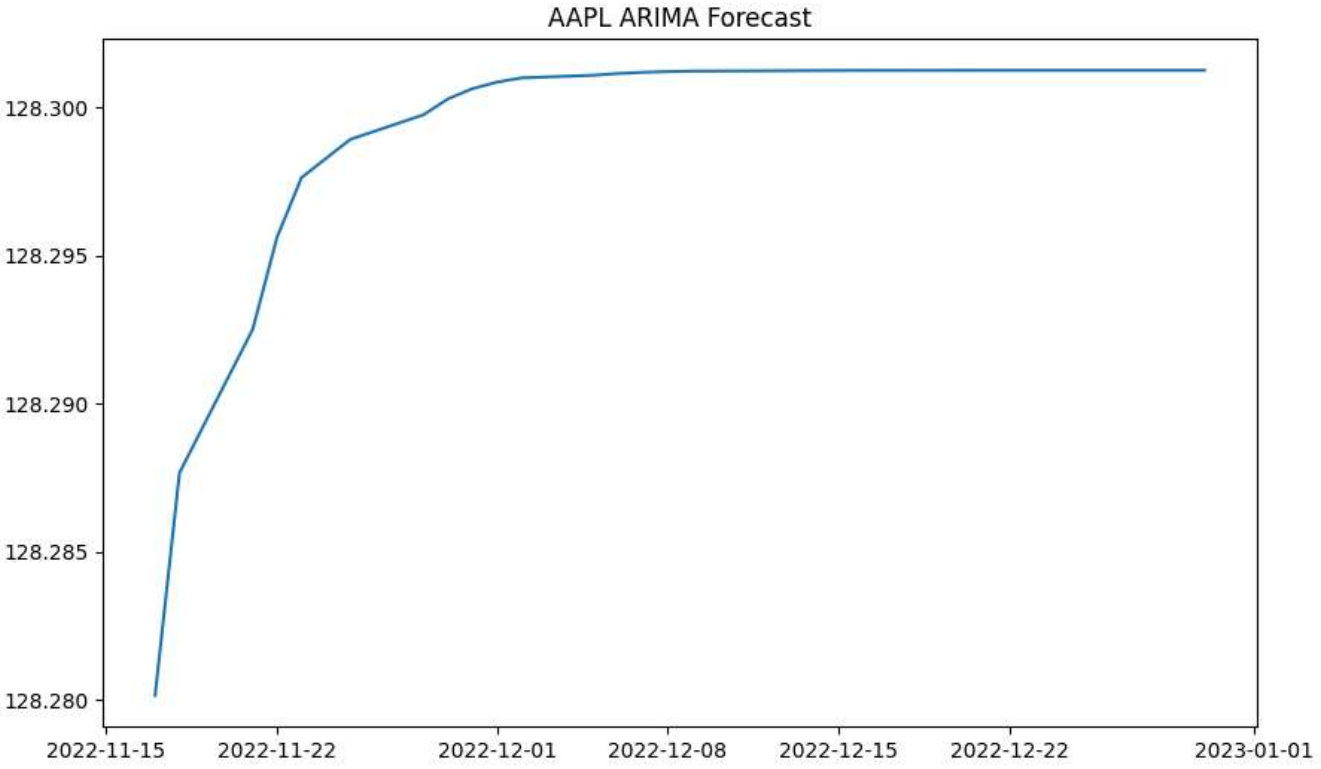



```
[**********100%**********]  1 of 1 completed
INFO:prophet:Disabling yearly seasonality. Run prophet with yearly_seasonality=True to override this.
INFO:prophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
DEBUG:cmdstanpy:input tempfile: /tmp/tmppjkcpfo2/8jy9kwud.json
DEBUG:cmdstanpy:input tempfile: /tmp/tmppjkcpfo2/4cov2lc1.json
DEBUG:cmdstanpy:idx 0
DEBUG:cmdstanpy:running CmdStan, num_threads: None
DEBUG:cmdstanpy:CmdStan args: ['/usr/local/lib/python3.11/dist-packages/prophet/stan_model/prophet_model.bin', 'ra
03:25:49 - cmdstanpy - INFO - Chain [1] start processing
INFO:cmdstanpy:Chain [1] start processing
Prophet Forecast:
03:25:49 - cmdstanpy - INFO - Chain [1] done processing
INFO:cmdstanpy:Chain [1] done processing
```

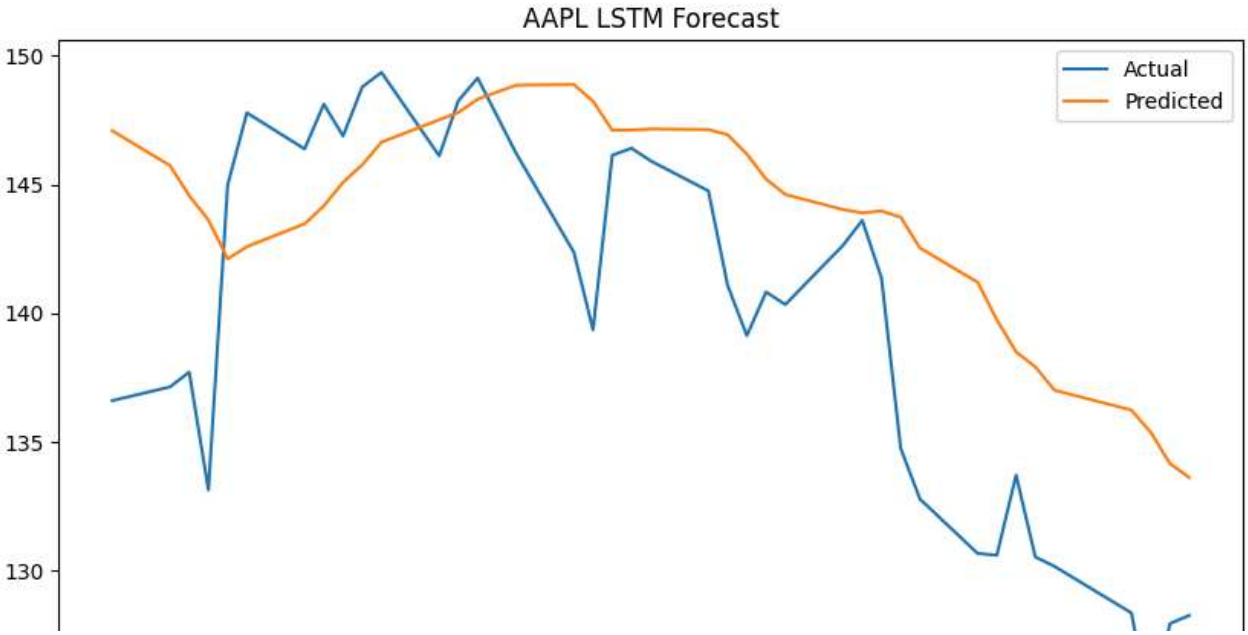


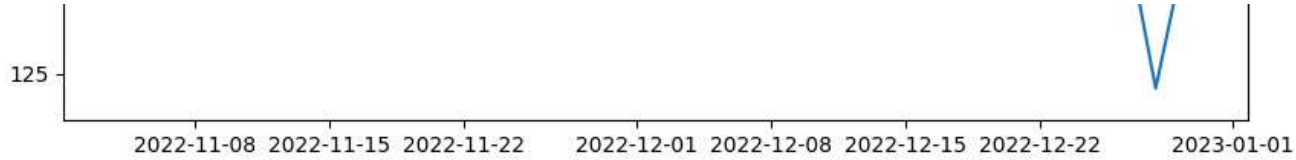



```
ARIMA Forecast:
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:473: ValueWarning: A date index has been
  self._init_dates(dates, freq)
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:837: ValueWarning: No supported index is
  return get_prediction_index(
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/base/tsa_model.py:837: FutureWarning: No supported index i
  return get_prediction_index(
```

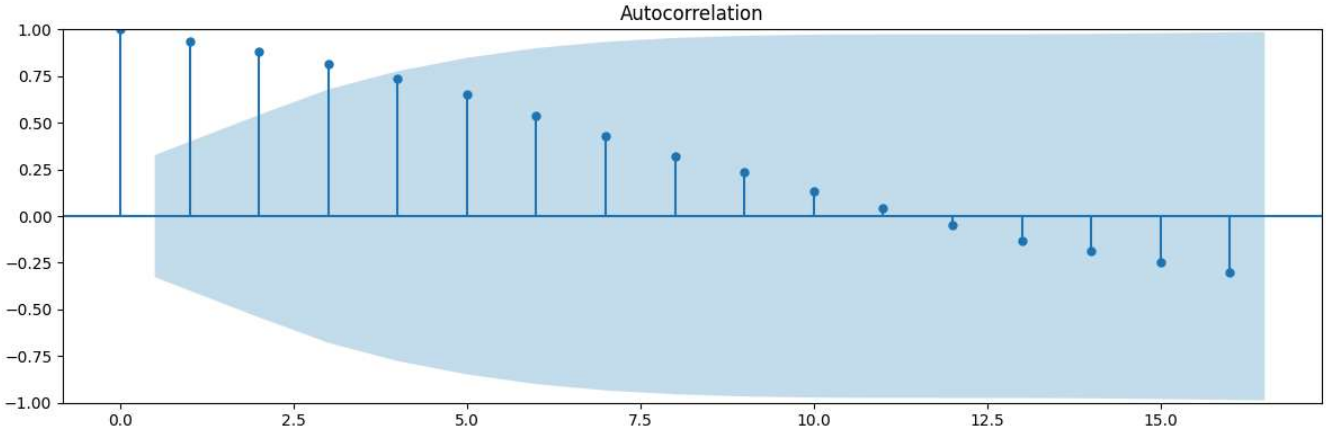


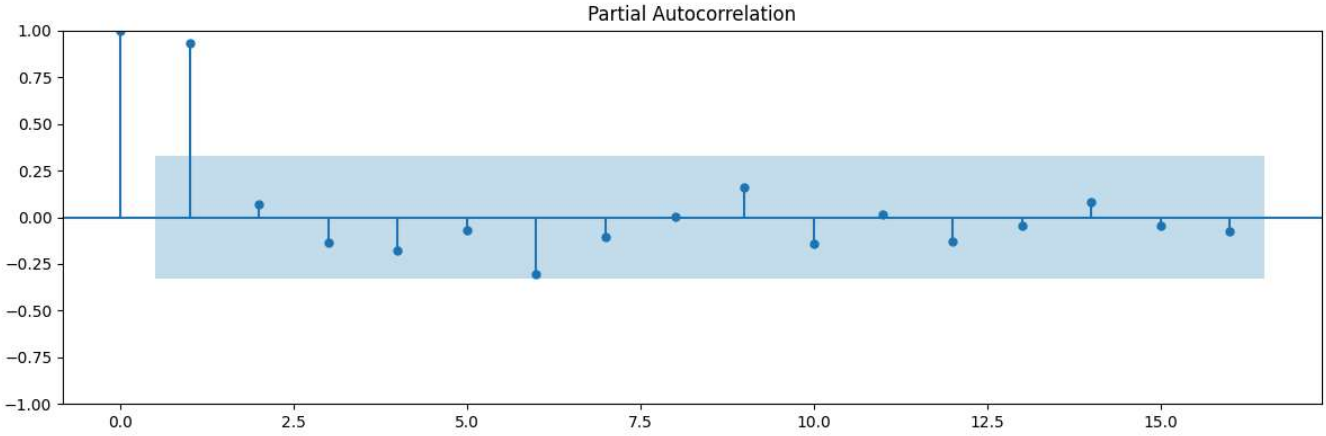

```
LSTM Forecast:
/usr/local/lib/python3.11/dist-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`
  super().__init__(**kwargs)
2/2 ━━━━━━━━━━━━━━━━━━━━ 0s 183ms/step
```






```
<ipython-input-3-4e6fd116ec52>:511: FutureWarning: 'M' is deprecated and will be removed in a future version, plea
  dates = pd.date_range(start='2020-01-01', end='2022-12-31', freq='M')
ADF Statistic: -0.2432954471407034
p-value: 0.9331602815879096
Data is non-stationary, consider differencing
```






```
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:966: UserWarning: Non-stationary sta
  warn('Non-stationary starting autoregressive parameters'
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:978: UserWarning: Non-invertible sta
  warn('Non-invertible starting MA parameters found.'
/usr/local/lib/python3.11/dist-packages/statsmodels/tsa/statespace/sarimax.py:866: UserWarning: Too few observatio
  warn('Too few observations to estimate starting parameters%s.'
                                SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                   36
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood                -38.270
Date:                            Fri, 23 May 2025   AIC                             86.541
Time:                                    03:26:12   BIC                             92.218
Sample:                                01-31-2020   HQIC                            87.968
                                     - 12-31-2022
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0645      8.535      0.008      0.994     -16.665      16.794
ma.L1         -0.0961      8.459     -0.011      0.991     -16.676      16.483
ar.S.L12      -0.4927   1801.844     -0.000      1.000   -3532.042    3531.057
ma.S.L12      -0.3902   4322.809  -9.03e-05      1.000   -8472.940    8472.159
sigma2         1.0888   1859.012      0.001      1.000   -3642.507    3644.685
===================================================================================
Ljung-Box (L1) (Q):                   0.57   Jarque-Bera (JB):                 1.36
Prob(Q):                              0.45   Prob(JB):                         0.51
Heteroskedasticity (H):               0.53   Skew:                             0.51
Prob(H) (two-sided):                  0.38   Kurtosis:                         2.39
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```


```
Model Performance Metrics:
Mean Squared Error (MSE): 11.948513343612246
Mean Absolute Error (MAE): 3.368874997731735
```