```
# Country Population from 1960 to 2022
```

```
# import libraries
import pandas as pd
```

```
from google.colab import drive
drive.mount("/content/grive")
```

```
Mounted at /content/grive
```

```
data=pd.read_csv("/content/grive/MyDrive/Tensorflow-demo/population_data.csv")
```

```
data.describe()
```

Show hidden output

```
data.head()
```

Show hidden output

```
data.describe()
```

Show hidden output

```
# Check for missing values
data.isnull().sum()
```

Show hidden output

```
# Calculate the world population by years
total_population = data.drop(columns=['Country Name']).sum()

# Plot the population over time
import matplotlib.pyplot as plt

years = total_population.index.astype(int)
population_values = total_population.values

plt.figure(figsize=(12, 6))
plt.plot(years, population_values, marker='o')
plt.title('Total World Population Over Time')
plt.xlabel('Year')
plt.ylabel('Total Population')
plt.grid(True)
plt.show()
```

Show hidden output

```
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
# Extract data for a specific country
country = 'United States'
country_data = data[data['Country Name'] == country].drop(columns=['Country Name']).values.flatten()

# Prepare the dataset for LSTM
def create_sequences(data, seq_length):
    sequences = []
    labels = []
    for i in range(len(data) - seq_length):
        sequences.append(data[i:i+seq_length])
        labels.append(data[i+seq_length])
    return np.array(sequences), np.array(labels)

# Parameters
seq_length = 10
X, y = create_sequences(country_data, seq_length)

# Reshape data to fit LSTM model
X = X.reshape((X.shape[0], X.shape[1], 1))

# Split into train and test sets
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 the LSTM model
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(seq_length, 1)))
model.add(Dropout(0.2))
model.add(LSTM(50, return_sequences=False))
model.add(Dropout(0.2))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=16, validation_data=(X_test, y_test))

# Predict future population
predictions = model.predict(X_test)

# Plot predictions vs actual values
plt.figure(figsize=(12, 6))
plt.plot(range(len(y_test)), y_test, color='blue', label='Actual Population')
plt.plot(range(len(predictions)), predictions, color='red', linestyle='dashed', label='Predicted Population')
plt.title(f'Population Prediction for {country}')
plt.xlabel('Time Step')
plt.ylabel('Population')
plt.legend()
plt.show()
```

```
Epoch 1/100
3/3 [==============================] - 8s 837ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 2/100
3/3 [==============================] - 0s 27ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 3/100
3/3 [==============================] - 0s 30ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 4/100
3/3 [==============================] - 0s 29ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 5/100
3/3 [==============================] - 0s 28ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 6/100
3/3 [==============================] - 0s 32ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 7/100
3/3 [==============================] - 0s 41ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 8/100
3/3 [==============================] - 0s 37ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 9/100
3/3 [==============================] - 0s 27ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 10/100
3/3 [==============================] - 0s 30ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 11/100
3/3 [==============================] - 0s 31ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 12/100
3/3 [==============================] - 0s 29ms/step - loss: 66314973655072768.0000 - val_loss: 105325938266341376.0000
Epoch 13/100
```