[1]:
```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential
from keras.layers import Dense, SimpleRNN, Dropout
```

[2]:
```python
# Load data
data = pd.read_csv("TSLA.csv")
print(data.head())
```

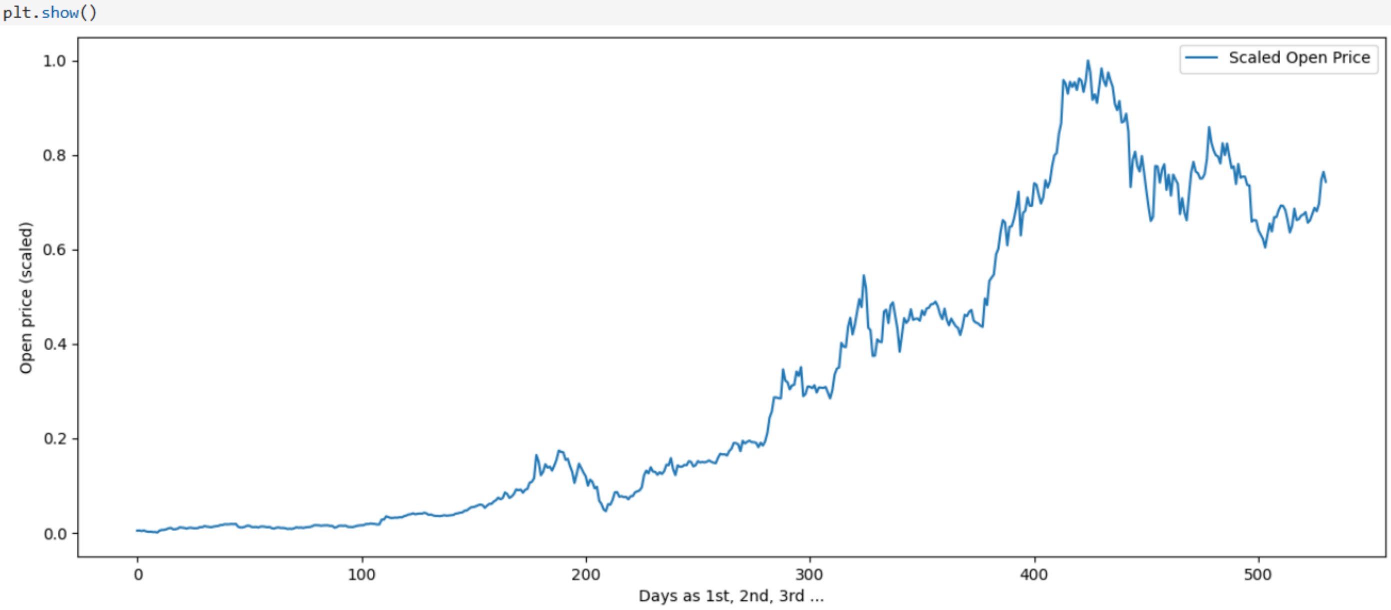
```
         Date       Open       High        Low      Close     Volume  \
0  2019-05-21  39.551998  41.480000  39.208000  41.015999   90019500
1  2019-05-22  39.820000  40.787998  38.355999  38.546001   93426000
2  2019-05-23  38.868000  39.894001  37.243999  39.098000  132735500
3  2019-05-24  39.966000  39.995998  37.750000  38.125999   70683000
4  2019-05-28  38.240002  39.000000  37.570000  37.740002   51564500

   Dividends  Stock Splits
0          0           0.0
1          0           0.0
2          0           0.0
3          0           0.0
4          0           0.0
```

[3]:
```python
# Prepare data
length_data = len(data)
split_ratio = 0.7
length_train = round(length_data * split_ratio)
length_validation = length_data - length_train
print("Data length:", length_data)
print("Train data length:", length_train)
print("Validation data length:", length_validation)
```

```
Data length: 758
Train data length: 531
Validation data length: 227
```

[4]:
```python
# Split data into training and validation sets
train_data = data[:length_train].iloc[:, :2]
validation_data = data[length_train:].iloc[:, :2]
train_data_numeric = train_data.select_dtypes(include=['float64', 'int64'])
```

[5]:
```python
# Scale the data
scaler = MinMaxScaler(feature_range=(0, 1))
train_data_numeric_scaled = scaler.fit_transform(train_data_numeric)
train_data_scaled = pd.DataFrame(train_data_numeric_scaled, columns=train_data_numeric.columns)
train_data_scaled = pd.concat([train_data['Date'].reset_index(drop=True), train_data_scaled], axis=1)

# Plot scaled data
plt.figure(figsize=(15, 6))
plt.plot(train_data_scaled['Open'], label="Scaled Open Price")  # Only plot the 'Open' column
plt.xlabel("Days as 1st, 2nd, 3rd ...")
plt.ylabel("Open price (scaled)")
plt.legend()
plt.show()
```



[6]:
```python
# Prepare data for training
train_data_array = train_data_scaled['Open'].values  # Assuming 'Open' is the column to use
X_train = []
y_train = []
time_step = 50

# Prepare X_train and y_train
for i in range(time_step, len(train_data_array)):
    X_train.append(train_data_array[i-time_step:i])
    y_train.append(train_data_array[i])

# Convert lists to numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)

# Reshape the data
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
y_train = np.reshape(y_train, (y_train.shape[0], 1))
print("Shape of X_train after reshape:", X_train.shape)
print("Shape of y_train after reshape:", y_train.shape)
```

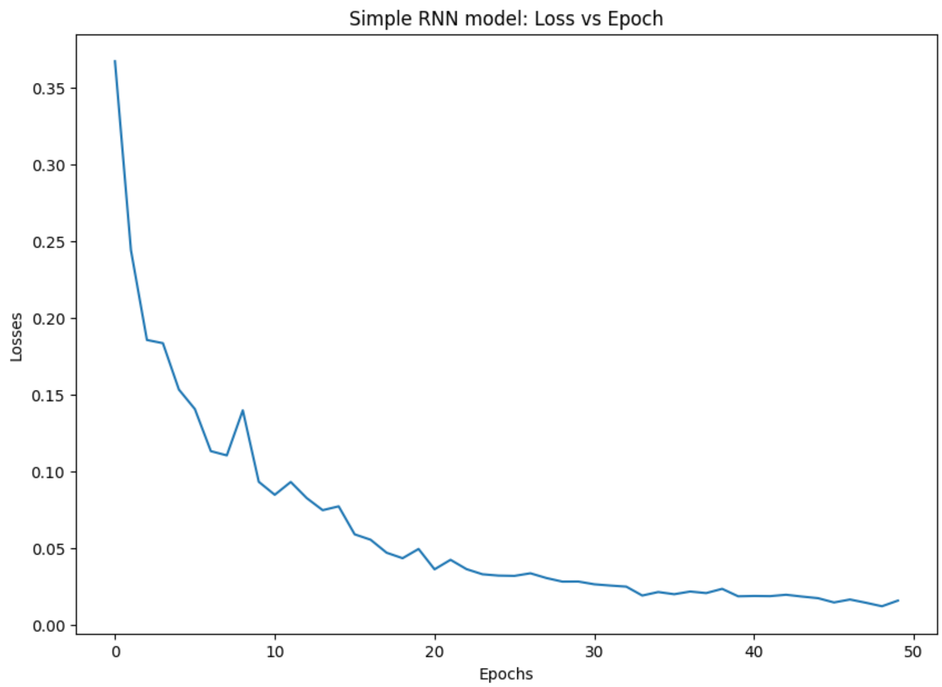
```
Shape of X_train after reshape: (481, 50, 1)
Shape of y_train after reshape: (481, 1)
```

[7]:
```python
# Initialize the RNN
regressor = Sequential()

# Adding RNN layers
regressor.add(SimpleRNN(units=50, activation="tanh", return_sequences=True, input_shape=(X_train.shape[1], X_train.shape[2])))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units=50, activation="tanh", return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units=50, activation="tanh", return_sequences=True))
regressor.add(Dropout(0.2))
regressor.add(SimpleRNN(units=50))
regressor.add(Dropout(0.2))

# Adding the output layer
regressor.add(Dense(units=1))

# Compiling the RNN
regressor.compile(optimizer="adam", loss="mean_squared_error", metrics=["accuracy"])

# Fitting the RNN
history = regressor.fit(X_train, y_train, epochs=50, batch_size=32)
```

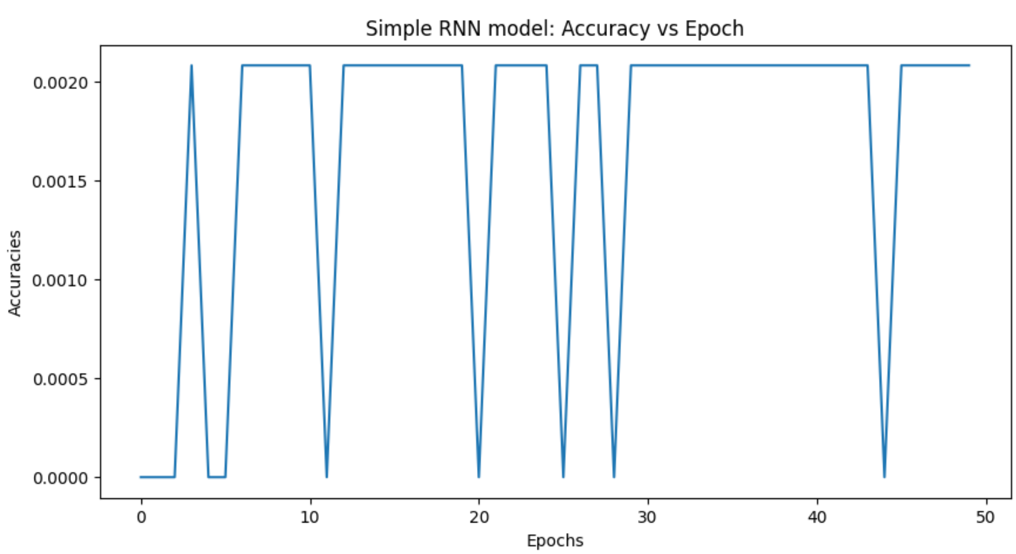
```
Epoch 1/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 5s 19ms/step - accuracy: 0.0000e+00 - loss: 0.4228
Epoch 2/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss: 0.2665
Epoch 3/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss: 0.1890
Epoch 4/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0018 - loss: 0.1702
Epoch 5/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss: 0.1555
Epoch 6/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss: 0.1379
Epoch 7/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 7.9303e-04 - loss: 0.1123
Epoch 8/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0011 - loss: 0.1181
Epoch 9/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0025 - loss: 0.1380
Epoch 10/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 7.9303e-04 - loss: 0.0942
Epoch 11/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0036 - loss: 0.0889
Epoch 12/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0000e+00 - loss: 0.0804
Epoch 13/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0016 - loss: 0.0749
Epoch 14/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.0036 - loss: 0.0725
Epoch 15/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 0.0063 - loss: 0.0778
Epoch 16/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 29ms/step - accuracy: 0.0036 - loss: 0.0592
```

```
Epoch 46/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 6.3984e-04 - loss: 0.0154
Epoch 47/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 23ms/step - accuracy: 3.6714e-04 - loss: 0.0179
Epoch 48/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 19ms/step - accuracy: 0.0016 - loss: 0.0140
Epoch 49/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 20ms/step - accuracy: 0.0018 - loss: 0.0136
Epoch 50/50
16/16 ━━━━━━━━━━━━━━━━━━━━ 0s 22ms/step - accuracy: 0.0018 - loss: 0.0186
```

[8]:
```
# Plotting loss
plt.figure(figsize=(10, 7))
plt.plot(history.history["loss"])
plt.xlabel("Epochs")
plt.ylabel("Losses")
plt.title("Simple RNN model: Loss vs Epoch")
plt.show()
```



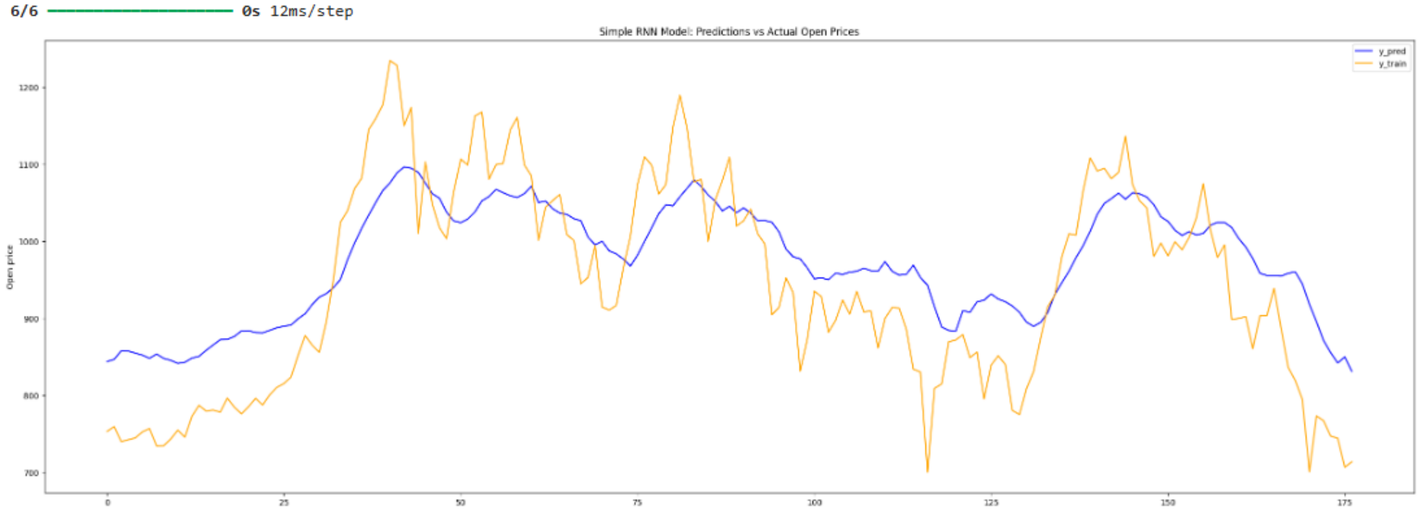
[9]:
```
# Plotting accuracy
plt.figure(figsize=(10, 5))
plt.plot(history.history["accuracy"])
plt.xlabel("Epochs")
plt.ylabel("Accuracies")
plt.title("Simple RNN model: Accuracy vs Epoch")
plt.show()
```



[10]:
```python
# Make predictions
# Create a test set from the validation data
validation_data_numeric_scaled = scaler.transform(validation_data.select_dtypes(include=['float64', 'int64']))
validation_data_scaled = pd.DataFrame(validation_data_numeric_scaled, columns=validation_data.select_dtypes(include=['float64', 'int64']).columns)

# Convert the 'Open' column to a NumPy array for processing
validation_data_array = validation_data_scaled['Open'].values

# Prepare the validation input
X_val = []
for i in range(time_step, len(validation_data_array)):
    X_val.append(validation_data_array[i-time_step:i])

# Convert to numpy array and reshape
X_val = np.array(X_val)
X_val = np.reshape(X_val, (X_val.shape[0], X_val.shape[1], 1))
```

[12]:
```python
# Make predictions
y_pred = regressor.predict(X_val)

# Rescale predictions to original scale
# Since y_pred is a 2D array, we can reshape it directly.
y_pred_rescaled = scaler.inverse_transform(np.concatenate((np.zeros((y_pred.shape[0], 1)), y_pred), axis=1))[:, 1]

# Plot predictions vs actual data
plt.figure(figsize=(30, 10))
plt.plot(y_pred_rescaled, color="b", label="y_pred")  # Predicted values
plt.plot(validation_data.iloc[time_step:, 1].values, color="orange", label="y_train")  # Actual values
plt.xlabel("Days")
plt.ylabel("Open price")
plt.title("Simple RNN Model: Predictions vs Actual Open Prices")
plt.legend()
plt.show()
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
```



[13]:
```python
dataset_validation = validation_data.Open.values #getting "open" column and converting to array
dataset_validation = np. reshape(dataset_validation, (-1,1)) # converting 1D to 2D array
scaled_dataset_validation = scaler.fit_transform(dataset_validation) # scaling open values to between 0 and 1
print("Shape of scaled validation dataset :", scaled_dataset_validation.shape)
```

```
Shape of scaled validation dataset : (227, 1)
```

[14]:
```python
X_test = []
y_test = []

for i in range(time_step, length_validation):
    X_test.append (scaled_dataset_validation[i-time_step: i,0])
    y_test.append (scaled_dataset_validation[i,0])
```

[15]:
```python
X_test, y_test = np.array(X_test), np.array(y_test)
```

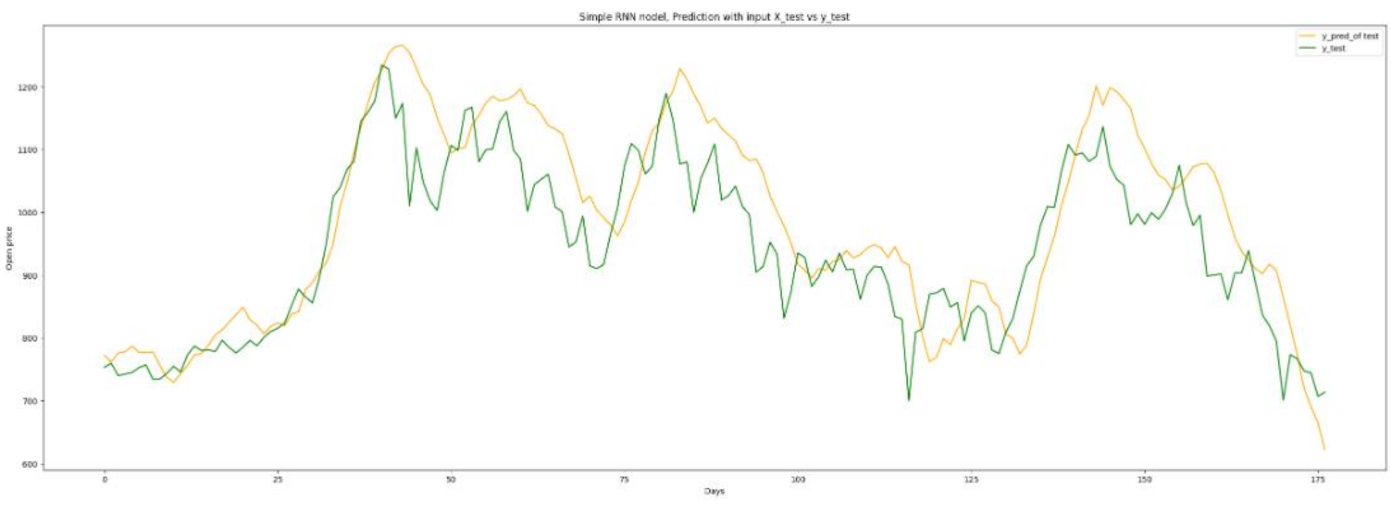
[16]:
```python
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
y_test = np.reshape(y_test, (-1,1)) # reshape to 2D array
print("Shape of X_test after reshape :",X_test.shape)
print("Shape of y_test after reshape :",y_test.shape)
```

```
Shape of X_test after reshape : (177, 50, 1)
Shape of y_test after reshape : (177, 1)
```

[17]:
```python
y_pred_of_test = regressor.predict(X_test)
# scaling back from 0-1 to original
y_predict = y_pred_of_test
y_pred_of_test = scaler.inverse_transform(y_pred_of_test )
print("Shape of y_pred_of_test :" ,y_pred_of_test.shape)
```

```
6/6 ━━━━━━━━━━━━━━━━━━━━ 0s 12ms/step
Shape of y_pred_of_test : (177, 1)
```

[19]:
```python
plt.figure(figsize = (30,10))
plt.plot(y_pred_of_test, label = "y_pred_of test", c = "orange" )
plt.plot(scaler.inverse_transform(y_test), label = "y_test", c = "g")
plt.xlabel (" Days " )
plt.ylabel("Open price")
plt.title("Simple RNN nodel, Prediction with input X_test vs y_test")
plt.legend()
plt.show()
```



[24]:
```python
# Make sure y_pred is a 1D array for plotting
y_pred_flattened = y_pred.flatten()  # Flatten y_pred to a 1D array

# We need to correctly match the lengths of the data being plotted
plt.subplots(figsize=(30, 12))

# Plotting the training and validation data
plt.plot(train_data.Date, train_data.Open, label="train_data", color="b")
plt.plot(validation_data.Date, validation_data.Open, label="validation_data", color="g")

# Since y_pred corresponds to the validation set predictions, we plot it against validation_data.Date
plt.plot(validation_data.Date.iloc[time_step:], y_pred_flattened, label="y_pred", color="r")

# If you have predictions for the test set, ensure it is defined and correctly shaped
# Ensure y_pred_of_test is properly computed before plotting
# plt.plot(validation_data.Date.iloc[time_step:], y_pred_of_test.flatten(), label="y_pred_of_test", color="orange")

plt.xlabel("Days")
plt.ylabel("Open price")
plt.title("Simple RNN Model: Train-Validation-Prediction")
plt.legend()
plt.show()
```

[25]:
```
from sklearn.metrics import r2_score
r2_score(y_test,y_predict)
```

[25]: 0.6146066816622466