In [1]:

```
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense
import seaborn as sns
```

In [2]:

```
# Load stock price data (Microsoft Corporation - MSFT)
url = "https://query1.finance.yahoo.com/v7/finance/download/MSFT?period1=0&period2=9999999999&interval=1d&events=h
df = pd.read_csv(url)
df.head()
```

Out[2]:

	Date	Open	High	Low	Close	Adj Close	Volume
0	1986-03-13	0.088542	0.101563	0.088542	0.097222	0.060274	1031788800
1	1986-03-14	0.097222	0.102431	0.097222	0.100694	0.062427	308160000
2	1986-03-17	0.100694	0.103299	0.100694	0.102431	0.063504	133171200
3	1986-03-18	0.102431	0.103299	0.098958	0.099826	0.061889	67766400
4	1986-03-19	0.099826	0.100694	0.097222	0.098090	0.060812	47894400

In [3]:

```
# Check for null values in the DataFrame
null_values = df.isnull().sum()

# Display the columns with null values (if any)
columns_with_null = null_values[null_values > 0]
print("Columns with null values:")
print(columns_with_null)
```

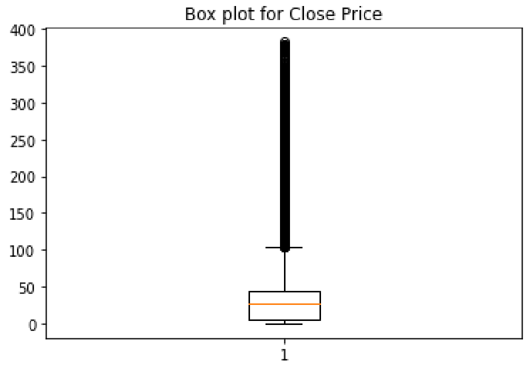
```
Columns with null values:
Series([], dtype: int64)
```

In [4]:

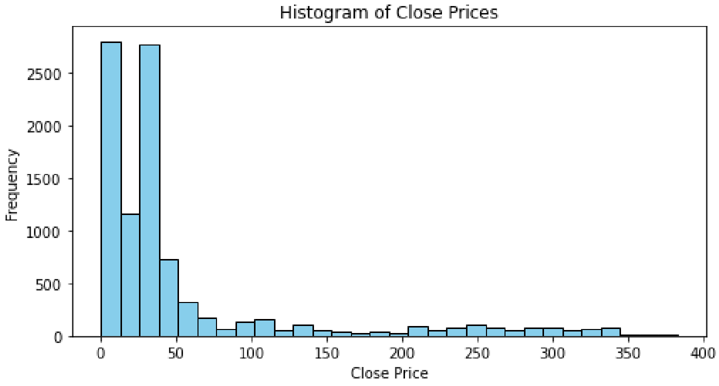
```
# Calculate the IQR for each column
Q1 = df.quantile(0.25)
Q3 = df.quantile(0.75)
IQR = Q3 - Q1

# Define a threshold for identifying outliers
outlier_threshold = 1.5

# Identify rows with potential outliers
outliers_mask = ((df < (Q1 - outlier_threshold * IQR)) | (df > (Q3 + outlier_threshold * IQR))).any(axis=1)

# Remove rows with outliers
df_no_outliers = df[~outliers_mask]

# Display the shape of the original and modified DataFrame
print("Original DataFrame shape:", df.shape)
print("DataFrame shape after removing outliers:", df_no_outliers.shape)
```

```
Original DataFrame shape: (9516, 7)
DataFrame shape after removing outliers: (7675, 7)

<ipython-input-4-584031639dbb>:10: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is dep
recated and will raise ValueError in a future version.  Do `left, right = left.align(right, axis=1, copy=False)`
before e.g. `left == right`
  outliers_mask = ((df < (Q1 - outlier_threshold * IQR)) | (df > (Q3 + outlier_threshold * IQR))).any(axis=1)
<ipython-input-4-584031639dbb>:10: FutureWarning: Automatic reindexing on DataFrame vs Series comparisons is dep
recated and will raise ValueError in a future version.  Do `left, right = left.align(right, axis=1, copy=False)`
before e.g. `left == right`
  outliers_mask = ((df < (Q1 - outlier_threshold * IQR)) | (df > (Q3 + outlier_threshold * IQR))).any(axis=1)
```

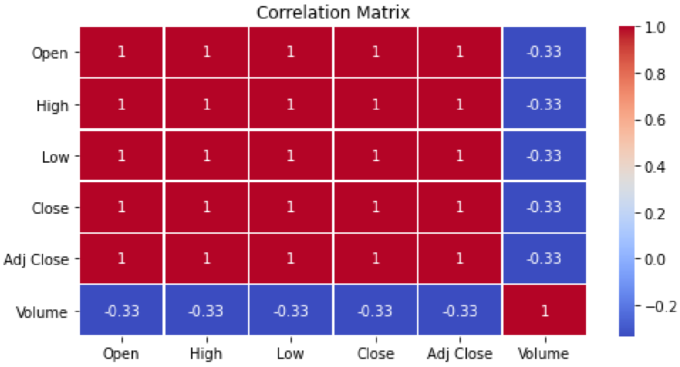
In [5]:
```
# Box plot for a specific column (replace 'Close' with the column of interest)
plt.boxplot(df['Close'])
plt.title('Box plot for Close Price')
plt.show()
```



In [6]:
```
# Plot histogram for 'Close' column
plt.figure(figsize=(8, 4))
plt.hist(df['Close'], bins=30, color='skyblue', edgecolor='black')
plt.title('Histogram of Close Prices')
plt.xlabel('Close Price')
plt.ylabel('Frequency')
plt.show()
```



In [7]:
```
# Create a correlation matrix
correlation_matrix = df.corr()

# Plot the correlation matrix using a heatmap
plt.figure(figsize=(8, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix')
plt.show()
```



In [8]:
```
# Use the 'Close' prices for prediction
data = df['Close'].values.reshape(-1, 1)
```

In [9]:
```
data
```

Out[9]:
```
array([[9.72220000e-02],
       [1.00694000e-01],
       [1.02431000e-01],
       ...,
       [3.71299988e+02],
       [3.74380005e+02],
       [3.74369995e+02]])
```

In [10]:

```
# Normalize the data
scaler = MinMaxScaler(feature_range=(0, 1))
data_normalized = scaler.fit_transform(data)
```

In [11]:

```
# Split the data into training and testing sets
train_size = int(len(data_normalized) * 0.80)
test_size = len(data_normalized) - train_size
train_data, test_data = data_normalized[0:train_size, :], data_normalized[train_size:len(data_normalized), :]
train_data
```

Out[11]:

```
array([[1.81490417e-05],
       [2.72235625e-05],
       [3.17634365e-05],
       ...,
       [1.32562553e-01],
       [1.31281872e-01],
       [1.32065958e-01]])
```

In [12]:

```
test_data
```

Out[12]:

```
array([[0.13052392],
       [0.13460118],
       [0.1359864 ],
       ...,
       [0.97020456],
       [0.97825459],
       [0.97822842]])
```

In [13]:

```
# Create sequences for LSTM
def create_sequences(data, seq_length):
    x, y = [], []
    for i in range(len(data)-seq_length):
        x.append(data[i:(i+seq_length), 0])
        y.append(data[i+seq_length, 0])
    return np.array(x), np.array(y)
```

In [14]:

```
# Create sequences for LSTM
seq_length = 10  # You can adjust this parameter based on your preference
x_train, y_train = create_sequences(train_data, seq_length)
x_test, y_test = create_sequences(test_data, seq_length)
```

In [15]:

```
# Ensure the length of x_train, y_train, x_test, and y_test match
x_train = x_train[:min(len(x_train), len(y_train))]
y_train = y_train[:min(len(x_train), len(y_train))]
x_test = x_test[:min(len(x_test), len(y_test))]
y_test = y_test[:min(len(x_test), len(y_test))]
```

In [16]:

```
# Build the LSTM model
model = Sequential()
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train.shape[1], 1)))
model.add(LSTM(units=50))
model.add(Dense(units=1))
model.compile(optimizer='adam', loss='mean_squared_error')
```

In [17]:

```
# Train the model
model.fit(x_train, y_train, epochs=10, batch_size=32)
```

```
Epoch 1/10
238/238 [==============================] - 14s 23ms/step - loss: 1.1570e-04
Epoch 2/10
238/238 [==============================] - 5s 23ms/step - loss: 8.1858e-06
Epoch 3/10
238/238 [==============================] - 5s 22ms/step - loss: 8.4024e-06
Epoch 4/10
238/238 [==============================] - 5s 23ms/step - loss: 8.9314e-06
Epoch 5/10
238/238 [==============================] - 5s 23ms/step - loss: 7.9108e-06
Epoch 6/10
238/238 [==============================] - 5s 23ms/step - loss: 7.9149e-06
Epoch 7/10
238/238 [==============================] - 5s 23ms/step - loss: 7.0859e-06
Epoch 8/10
238/238 [==============================] - 5s 23ms/step - loss: 6.5196e-06
Epoch 9/10
238/238 [==============================] - 5s 23ms/step - loss: 6.3166e-06
Epoch 10/10
238/238 [==============================] - 5s 23ms/step - loss: 6.3118e-06
```

Out[17]:

```
<keras.src.callbacks.History at 0x1e323b77a00>
```

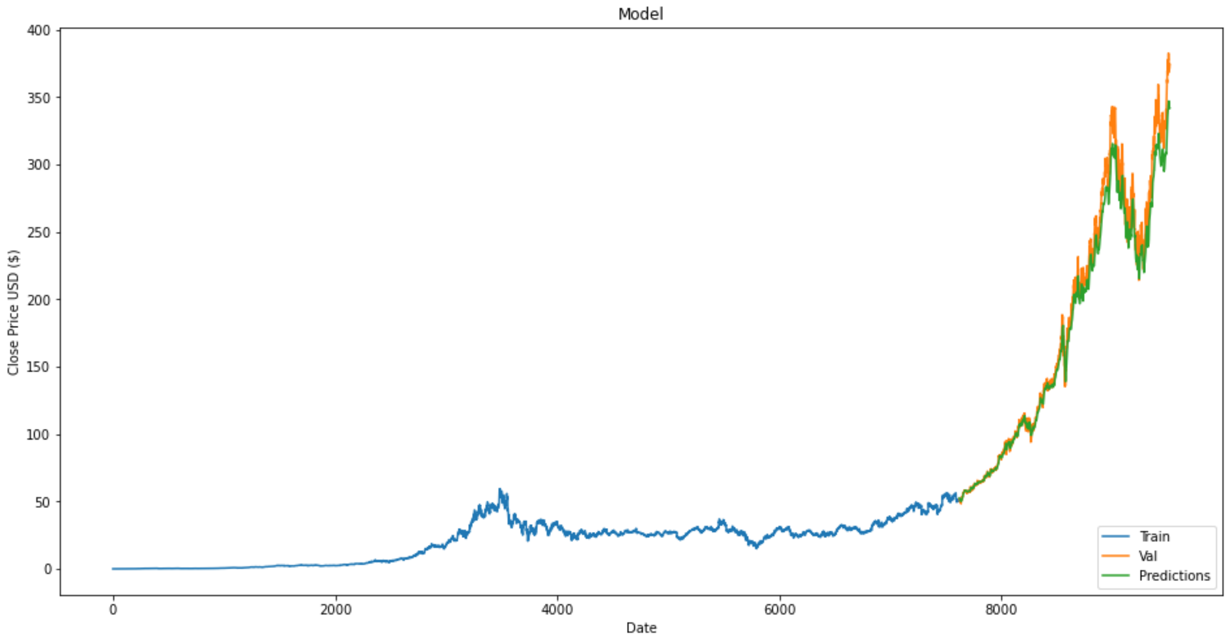
In [18]:

```
# Make predictions on the test data
predictions = model.predict(x_test)
predictions = scaler.inverse_transform(predictions)
```

```
60/60 [==============================] - 2s 9ms/step
```

In [27]:

```
# Ensure lengths match
valid = df.iloc[train_size + seq_length:].copy()
valid['Predictions'] = predictions[:len(valid)]

# Plot the results
plt.figure(figsize=(16, 8))
plt.title('Model')
plt.xlabel('Date')
plt.ylabel('Close Price USD ($)')
plt.plot(train['Close'])
plt.plot(valid[['Close', 'Predictions']])
plt.legend(['Train', 'Val', 'Predictions'], loc='lower right')
plt.show()
```


Model
Close Price USD ($)
Date
Train
Val
Predictions