Importing necessary libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScale
from ipywidgets import interact
from sklearn.model_selection import train_test_split

import plotly.express as px
import plotly.graph_objects as go
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, GRU, Bidirectional
from tensorflow.keras import layers, callbacks
```

```
WARNING:tensorflow:From C:\Users\Asus\anaconda3\lib\site-packages\keras\sr
c\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is depre
cated. Please use tf.compat.v1.losses.sparse_softmax_cross_entropy instea
d.
```

In [2]:
```
df=pd.read_csv(r"C:\Users\Asus\Downloads\Mastercard_stock_history.csv")
df
```

Out[2]:

	Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
0	2006-05-25	3.748967	4.283869	3.739664	4.279217	395343000	0.00	0.0
1	2006-05-26	4.307126	4.348058	4.103398	4.179680	103044000	0.00	0.0
2	2006-05-30	4.183400	4.184330	3.986184	4.093164	49898000	0.00	0.0
3	2006-05-31	4.125723	4.219679	4.125723	4.180608	30002000	0.00	0.0
4	2006-06-01	4.179678	4.474572	4.176887	4.419686	62344000	0.00	0.0
...	...	...	...	...	...	...	...	...
3867	2021-10-05	347.121403	348.130138	342.497241	342.776886	4724100	0.00	0.0
3868	2021-10-06	339.580960	348.439763	338.682072	348.250000	3712000	0.00	0.0
3869	2021-10-07	349.000000	357.899994	349.000000	353.910004	3209200	0.44	0.0
3870	2021-10-08	356.000000	360.369995	354.209991	354.959991	2336700	0.00	0.0
3871	2021-10-11	353.950012	354.880005	346.899994	347.149994	2766800	0.00	0.0

3872 rows × 8 columns

Data Exploration

-To check (rows, columns)

In [3]:
```
df.shape
```

Out[3]:
```
(3872, 8)
```

-To get hold of the basic nature of each columns

In [4]:
```
df.describe()
```

Out[4]:

	Open	High	Low	Close	Volume	Dividends	Stoc
count	3872.000000	3872.000000	3872.000000	3872.000000	3.872000e+03	3872.000000	3872.
mean	104.896814	105.956054	103.769349	104.882714	1.232250e+07	0.002329	0.
std	106.245511	107.303589	105.050064	106.168693	1.759665e+07	0.025851	0.
min	3.748967	4.102467	3.739664	4.083861	6.411000e+05	0.000000	0.
25%	22.347203	22.637997	22.034458	22.300391	3.529475e+06	0.000000	0.
50%	70.810079	71.375896	70.224002	70.856083	5.891750e+06	0.000000	0.
75%	147.688448	148.645373	146.822013	147.688438	1.319775e+07	0.000000	0.
max	392.653890	400.521479	389.747812	394.685730	3.953430e+08	0.440000	10.

-To check the datatypes, the object ones need to be changed (it enables mathematical operations, facilitates statistical analysis)

In [5]:
```
df.dtypes
```

Out[5]:
```
Date             object
Open            float64
High            float64
Low             float64
Close           float64
Volume            int64
Dividends       float64
Stock Splits    float64
dtype: object
```

-To check for null values,(none-found..)

In [6]:
```
df.isnull().sum()
```

Out[6]:
```
Date            0
Open            0
High            0
Low             0
Close           0
Volume          0
Dividends       0
Stock Splits    0
dtype: int64
```

-Date column has been made as the index (easy time-based indexing, aids in time series analysis and forecasting)

In [7]:
```
df.index=pd.to_datetime(df['Date'])
df.drop(columns=['Date'], inplace=True)
df
```

Out[7]:

Date	Open	High	Low	Close	Volume	Dividends	Stock Splits
2006-05-25	3.748967	4.283869	3.739664	4.279217	395343000	0.00	0.0
2006-05-26	4.307126	4.348058	4.103398	4.179680	103044000	0.00	0.0
2006-05-30	4.183400	4.184330	3.986184	4.093164	49898000	0.00	0.0
2006-05-31	4.125723	4.219679	4.125723	4.180608	30002000	0.00	0.0
2006-06-01	4.179678	4.474572	4.176887	4.419686	62344000	0.00	0.0
...	...	...	...	...	...	...	...
2021-10-05	347.121403	348.130138	342.497241	342.776886	4724100	0.00	0.0
2021-10-06	339.580960	348.439763	338.682072	348.250000	3712000	0.00	0.0
2021-10-07	349.000000	357.899994	349.000000	353.910004	3209200	0.44	0.0
2021-10-08	356.000000	360.369995	354.209991	354.959991	2336700	0.00	0.0
2021-10-11	353.950012	354.880005	346.899994	347.149994	2766800	0.00	0.0

3872 rows × 7 columns

-The correlation matrix tells us how strongly/weakly one variable is related to the other

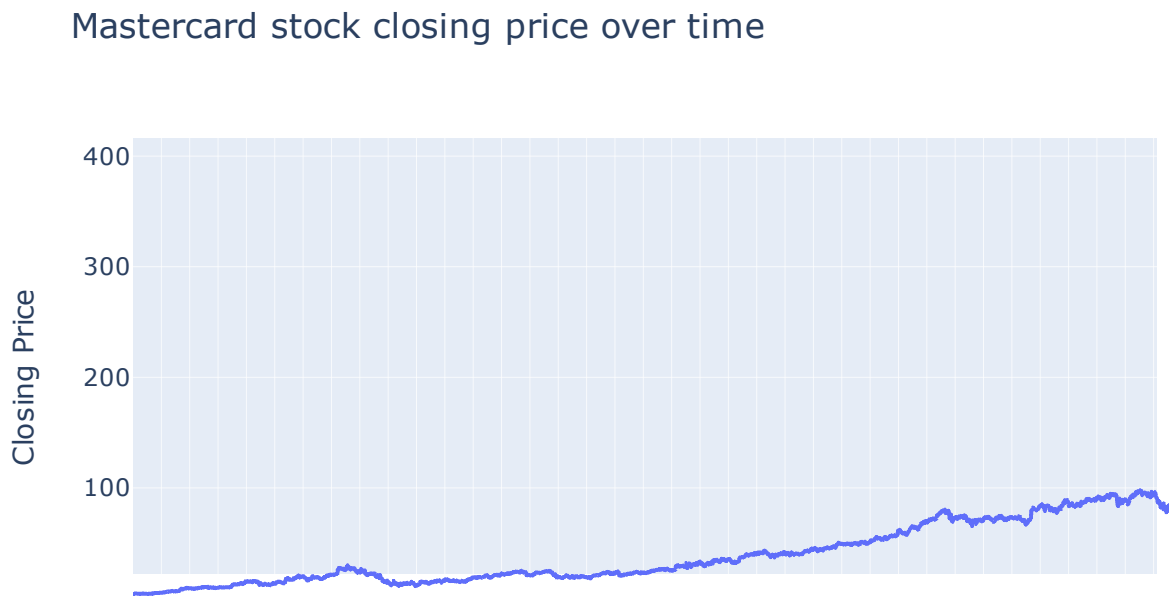
In [8]:
```
correlation_matrix= df.corr()
correlation_matrix
```

Out[8]:

	Open	High	Low	Close	Volume	Dividends	Stock Splits
Open	1.000000	0.999905	0.999868	0.999785	-0.386222	0.089059	-0.004019
High	0.999905	1.000000	0.999832	0.999878	-0.384819	0.089429	-0.003897
Low	0.999868	0.999832	1.000000	0.999904	-0.387965	0.089989	-0.003923
Close	0.999785	0.999878	0.999904	1.000000	-0.386453	0.090125	-0.003862
Volume	-0.386222	-0.384819	-0.387965	-0.386453	1.000000	-0.039454	-0.004661
Dividends	0.089059	0.089429	0.089989	0.090125	-0.039454	1.000000	-0.001448
Stock Splits	-0.004019	-0.003897	-0.003923	-0.003862	-0.004661	-0.001448	1.000000

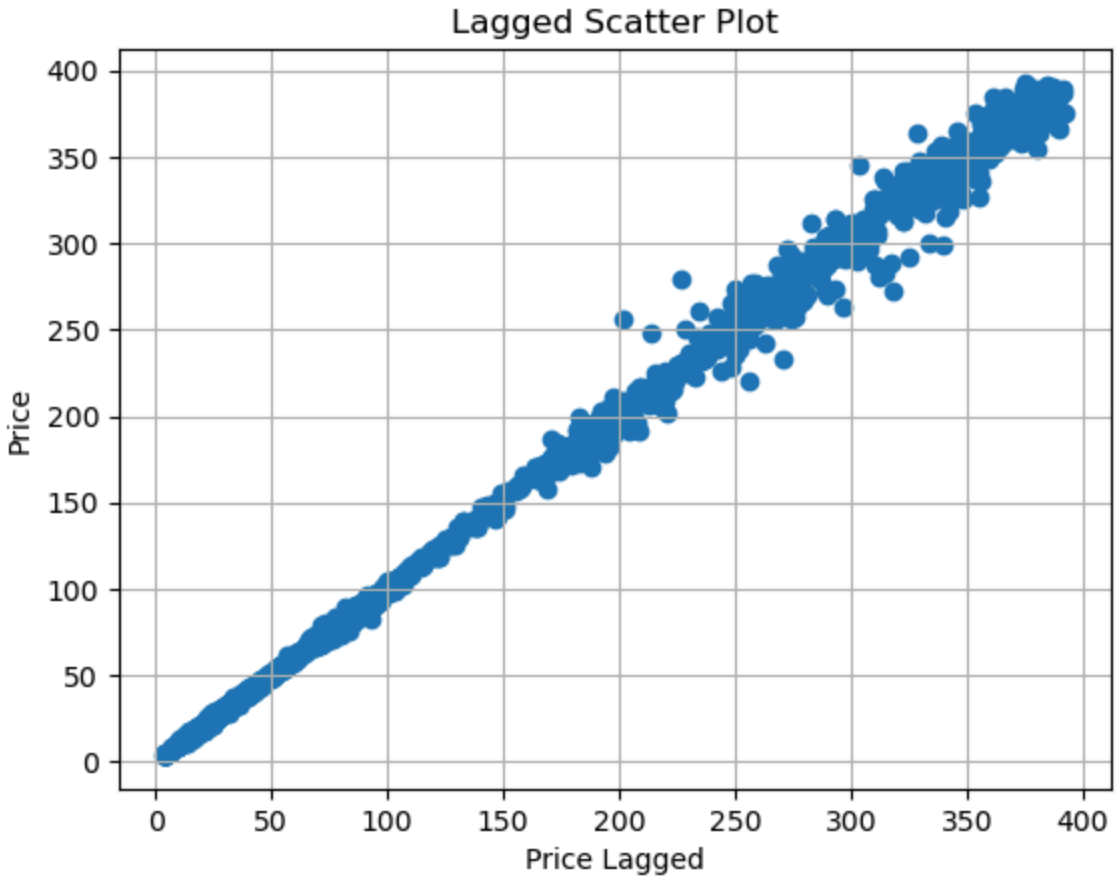
Data Visualization

In [9]:
```
df_reset_index=df.reset_index()

fig=px.line(df_reset_index, x='Date', y='Close', title='Mastercard stock cl
            labels={'Close':'Closing Price', 'Date':'Date'})
fig.update_xaxes(type='category')
fig.show()
```



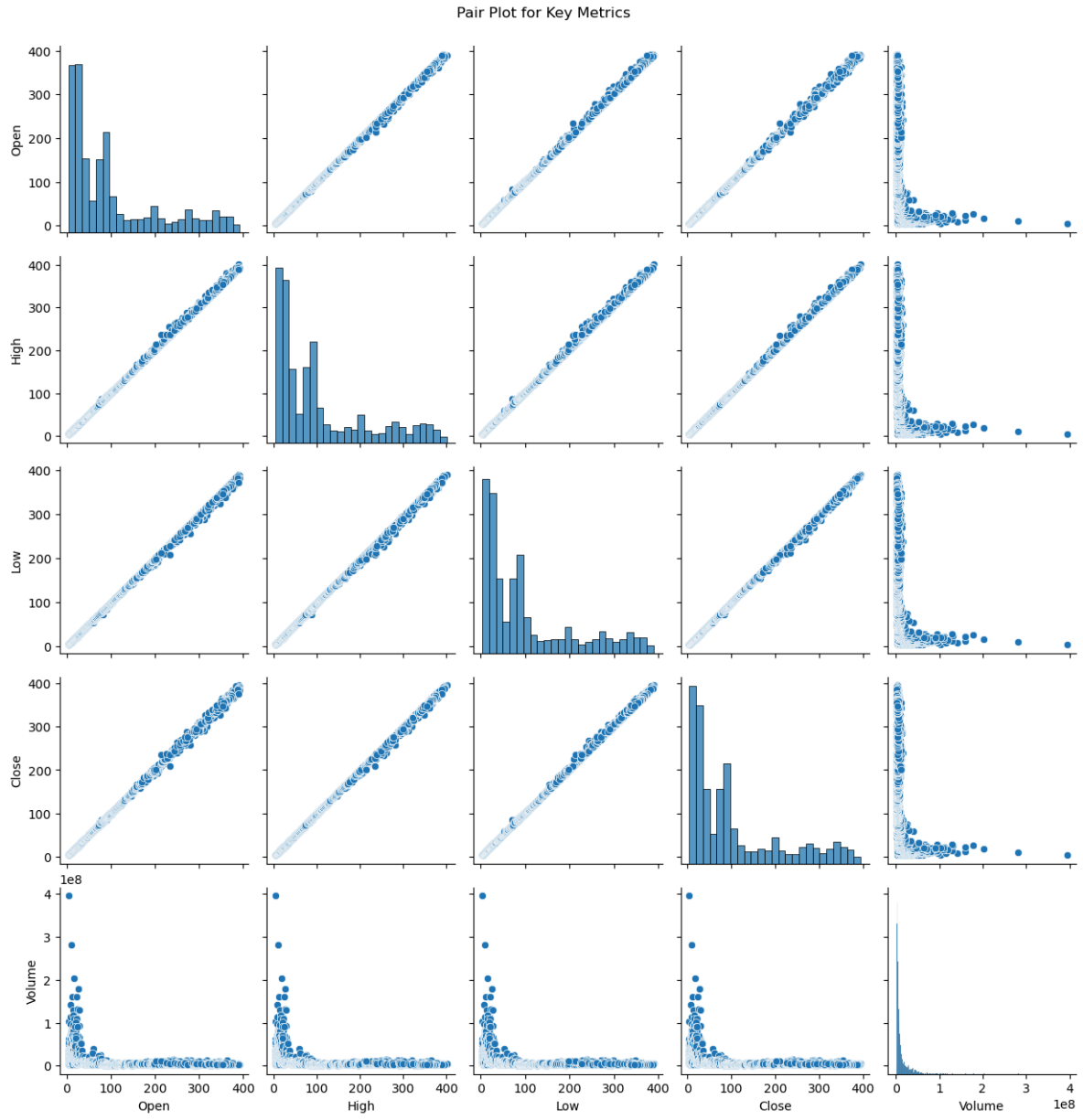
-Scatter plot between prices and lagged prices, The graph below shows that there is a positive correlation between the original prices and the lagged prices which means higher the lagged price, higher the current price. Hence this piece of info can be used to predict future prices more accurately, trading strategies, etc

In [10]:
```
lag = 3
df['Price Lagged'] = df['Open'].shift(lag)

# Create a lag plot
plt.scatter(df['Price Lagged'], df['Open'])
plt.xlabel('Price Lagged')
plt.ylabel('Price')
plt.title('Lagged Scatter Plot')
plt.grid(True)
plt.show()
```



-Pair plot for key metrics

In [11]:
```
sns.pairplot(df[['Open', 'High', 'Low', 'Close', 'Volume']], height=2.5)
plt.suptitle('Pair Plot for Key Metrics', y=1.02)
plt.grid()
plt.show()
```



Time-Series Analysis

-Finding the moving average

In [13]:
```
rolling_mean = df['Close'].rolling(window=30).mean()
rolling_std = df['Close'].rolling(window=30).std()
```

-It helps to smooth out short-term fluctuations and highlight long-term trends or patterns in the data.

In [14]:
```
rolling_mean
```

Out[14]:
```
Date
2006-05-25           NaN
2006-05-26           NaN
2006-05-30           NaN
2006-05-31           NaN
2006-06-01           NaN
                    ...
2021-10-05    348.600217
2021-10-06    348.164434
2021-10-07    348.006539
2021-10-08    348.091382
2021-10-11    347.820344
Name: Close, Length: 3872, dtype: float64
```

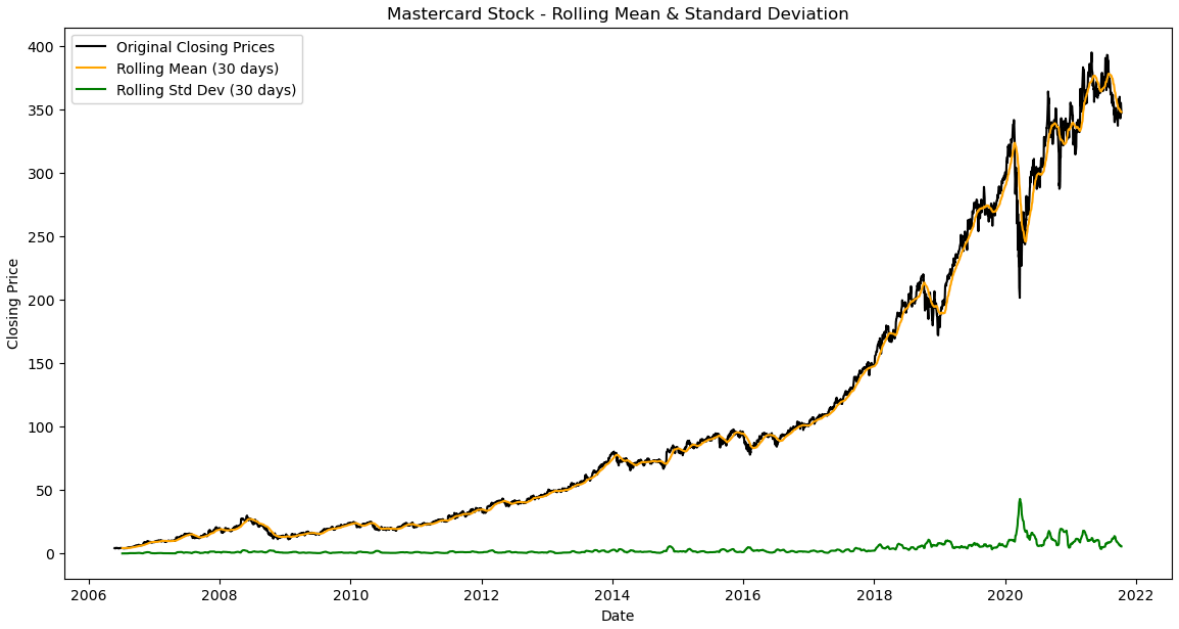
-It measures the dispersion or variability of data points around the rolling mean. A higher rolling standard deviation indicates greater variability or volatility in the data, while a lower standard deviation suggests more stability.

In [15]:
```
rolling_std
```

Out[15]:
```
Date
2006-05-25         NaN
2006-05-26         NaN
2006-05-30         NaN
2006-05-31         NaN
2006-06-01         NaN
                   ...
2021-10-05    6.473496
2021-10-06    6.010974
2021-10-07    5.784059
2021-10-08    5.868994
2021-10-11    5.711143
Name: Close, Length: 3872, dtype: float64
```

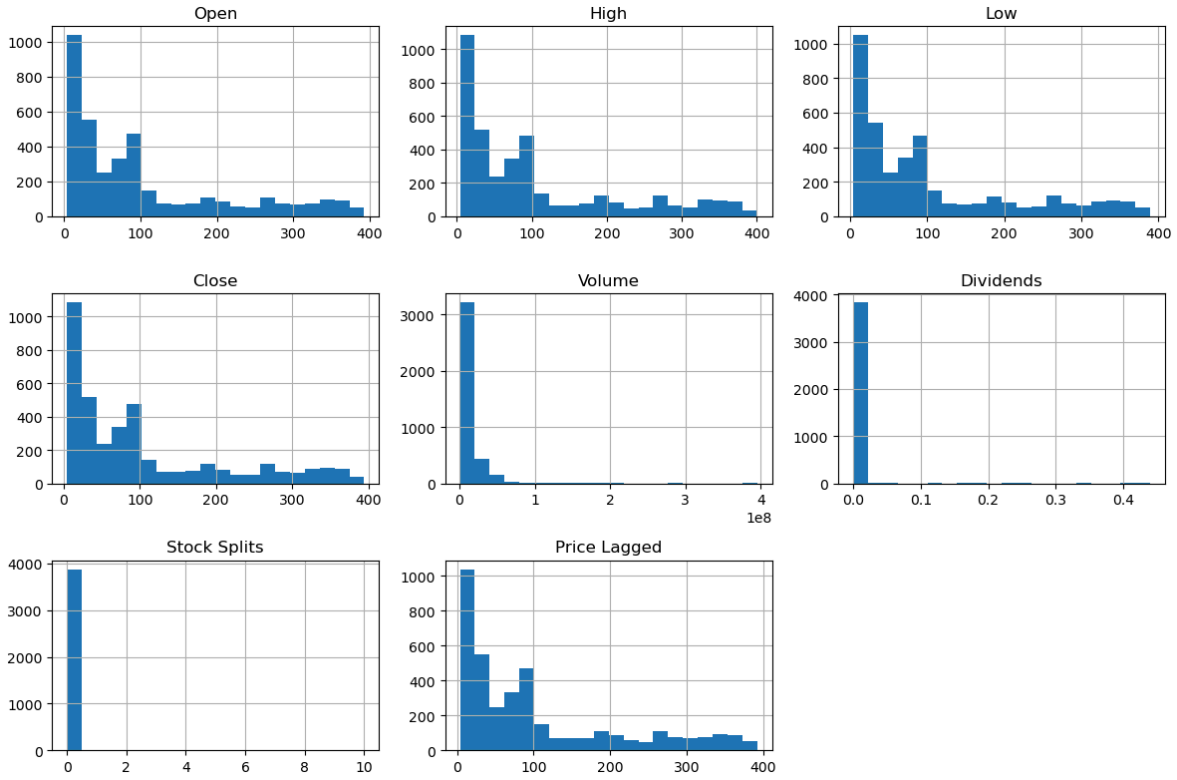
-The rolling STD basically shows how bumpy the ride for Mastercard stock has been over the past years

In [16]:
```
plt.figure(figsize=(14, 7))

plt.plot(df.index, df['Close'], label='Original Closing Prices', color='bla
plt.plot(df.index, rolling_mean, label='Rolling Mean (30 days)', color='ora
plt.plot(df.index, rolling_std, label='Rolling Std Dev (30 days)', color='g

plt.title('Mastercard Stock - Rolling Mean & Standard Deviation')
plt.xlabel('Date')
plt.ylabel('Closing Price')
plt.legend()
plt.show()
```

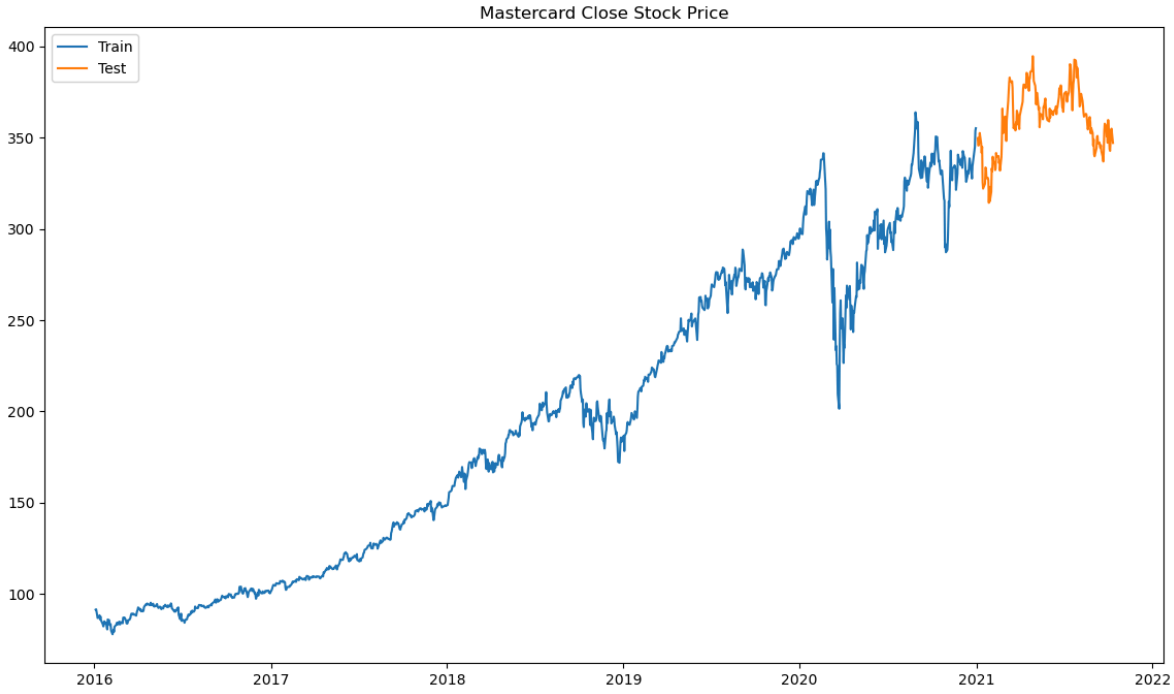


-The code snippet provided generates histograms for each numerical column in the DataFrame df

In [17]:
```
df.hist(figsize=(12, 8), bins=20)
plt.tight_layout()
plt.show()
```



In [18]:
```
train = df[(df.index.year<2021) & (df.index.year>=2016)]
test = df[df.index.year>=2021]

sc = MinMaxScaler(feature_range=(0,1))
training_set_scaled = sc.fit_transform(train['Close'].values.reshape(-1, 1)

plt.figure(figsize=(14,8))
plt.plot(train['Close'], label='Train')
plt.plot(test['Close'], label='Test')
plt.legend()
plt.title('Mastercard Close Stock Price')
plt.show()
```



In [19]:
```
X_train = []
y_train = []
for i in range(80,len(training_set_scaled)):
    X_train.append(training_set_scaled[i-80:i,0])
    y_train.append(training_set_scaled[i,0])
X_train, y_train = np.array(X_train), np.array(y_train)
```

In [20]:
```
X_train = np.reshape(X_train, (X_train.shape[0],X_train.shape[1],1))
```

In [21]:
```
model = Sequential([
    LSTM(units=50, return_sequences=True, input_shape=(X_train.shape[1],1)),
    layers.Dropout(0.2),

    layers.LSTM(units=50, return_sequences=True),
    layers.Dropout(0.2),

    layers.LSTM(units=50, return_sequences=True),
    layers.Dropout(0.2),

    layers.LSTM(units=50),
    layers.Dropout(0.2),

    layers.Dense(units=1),
])

model.compile(optimizer='adam',loss='mse', metrics=['mse','mae', 'mape'])
```

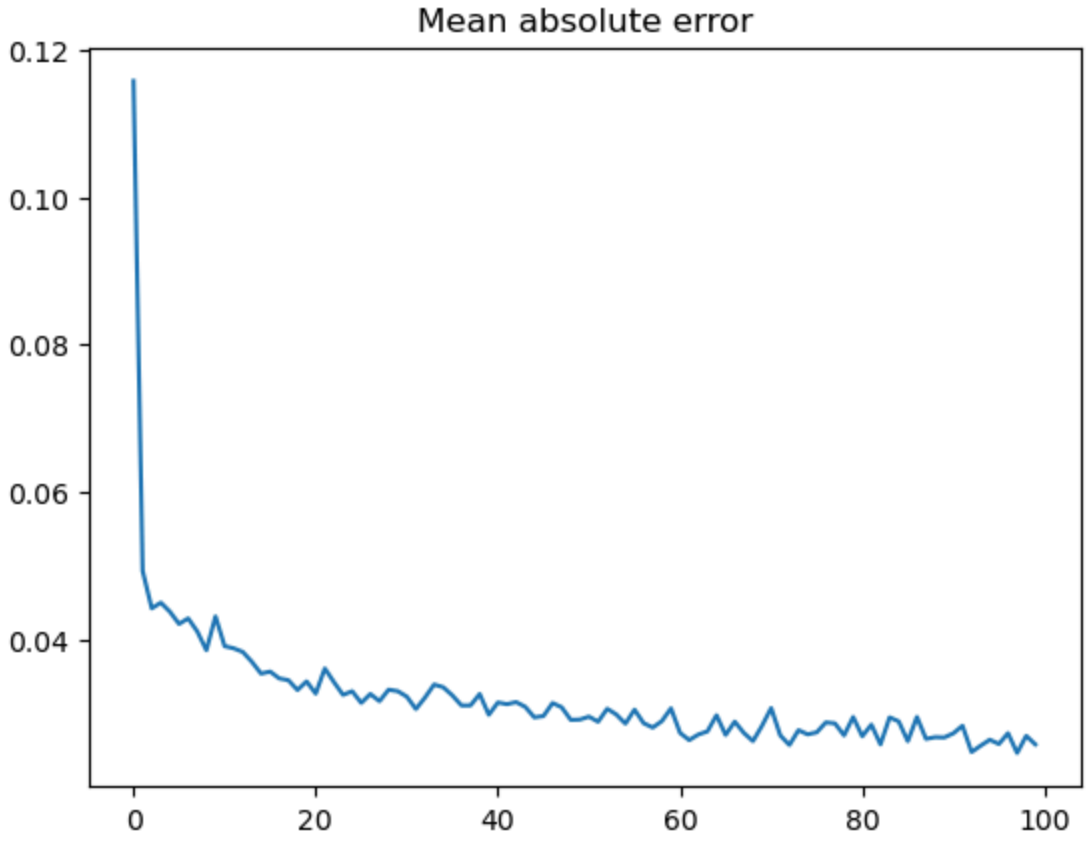
```
WARNING:tensorflow:From C:\Users\Asus\anaconda3\lib\site-packages\keras\sr
c\layers\rnn\lstm.py:148: The name tf.executing_eagerly_outside_functions
is deprecated. Please use tf.compat.v1.executing_eagerly_outside_functions
instead.

WARNING:tensorflow:From C:\Users\Asus\anaconda3\lib\site-packages\keras\sr
c\optimizers\__init__.py:309: The name tf.train.Optimizer is deprecated. P
lease use tf.compat.v1.train.Optimizer instead.
```

In [22]:
```
early_stopping = callbacks.EarlyStopping(
    patience=6,
    min_delta=0.001,
    restore_best_weights=True,
)

history = model.fit(X_train,y_train,epochs=100,batch_size=24, callbacks=[ea
```

```
50/50 [==============================] - ETA: 0s - loss: 0.0017 - mse:
0.0017 - mae: 0.0286 - mape: 11.0379WARNING:tensorflow:Early stopping c
onditioned on metric `val_loss` which is not available. Available metri
cs are: loss,mse,mae,mape
50/50 [==============================] - 5s 92ms/step - loss: 0.0017 -
mse: 0.0017 - mae: 0.0286 - mape: 11.0379
Epoch 56/100
50/50 [==============================] - ETA: 0s - loss: 0.0019 - mse:
0.0019 - mae: 0.0305 - mape: 11.9567WARNING:tensorflow:Early stopping c
onditioned on metric `val_loss` which is not available. Available metri
cs are: loss,mse,mae,mape
50/50 [==============================] - 4s 89ms/step - loss: 0.0019 -
mse: 0.0019 - mae: 0.0305 - mape: 11.9567
Epoch 57/100
50/50 [==============================] - ETA: 0s - loss: 0.0017 - mse:
0.0017 - mae: 0.0286 - mape: 10.9848WARNING:tensorflow:Early stopping c
onditioned on metric `val_loss` which is not available. Available metri
cs are: loss,mse,mae,mape
50/50 [==============================] - 5s 92ms/step - loss: 0.0017 -
mse: 0.0017 - mae: 0.0286 - mape: 10.9848
```

In [23]:
```
history_frame = pd.DataFrame(history.history)
```

In [24]:
```
plt.plot(history_frame.loc[:, ['mae']])
plt.title('Mean absolute error')
plt.show()
```



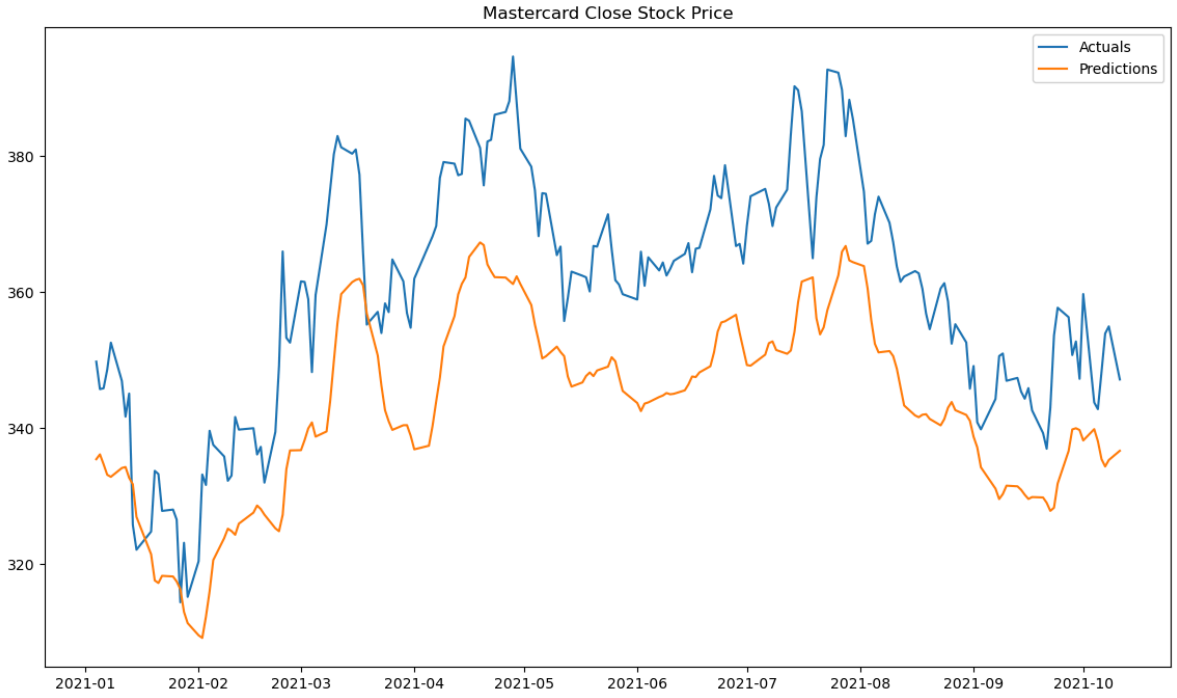
In [28]:
```
dataset_total = df['Close']
inputs = dataset_total[len(dataset_total)-len(test['Close'].values) - 80:].
inputs = inputs.reshape(-1,1)
inputs  = sc.transform(inputs)
```

In [29]:
```
X_test = []

for i in range(80,len(inputs)):
    X_test.append(inputs[i-80:i,0])
X_test = np.array(X_test)
X_test = np.reshape(X_test, (X_test.shape[0],X_test.shape[1],1))
predicted_stock_price = model.predict(X_test)
predicted_stock_price = sc.inverse_transform(predicted_stock_price)
```

```
7/7 [==============================] - 0s 23ms/step
```

In [30]:
```
predictions = pd.DataFrame()
predictions['Actuals'] = test['Close']
predictions['Predictions'] = predicted_stock_price
```

In [31]:
```
plt.figure(figsize=(14,8))
plt.title('Mastercard Close Stock Price')
plt.plot(predictions['Actuals'], label='Actuals')
plt.plot(predictions['Predictions'], label='Predictions')
plt.legend()
plt.show()
```



In [25]:
```
mae_value = history.history['mae'][-1]
print(f"MAE: {mae_value}")
```

```
MAE: 0.025766462087631226
```

In [32]:

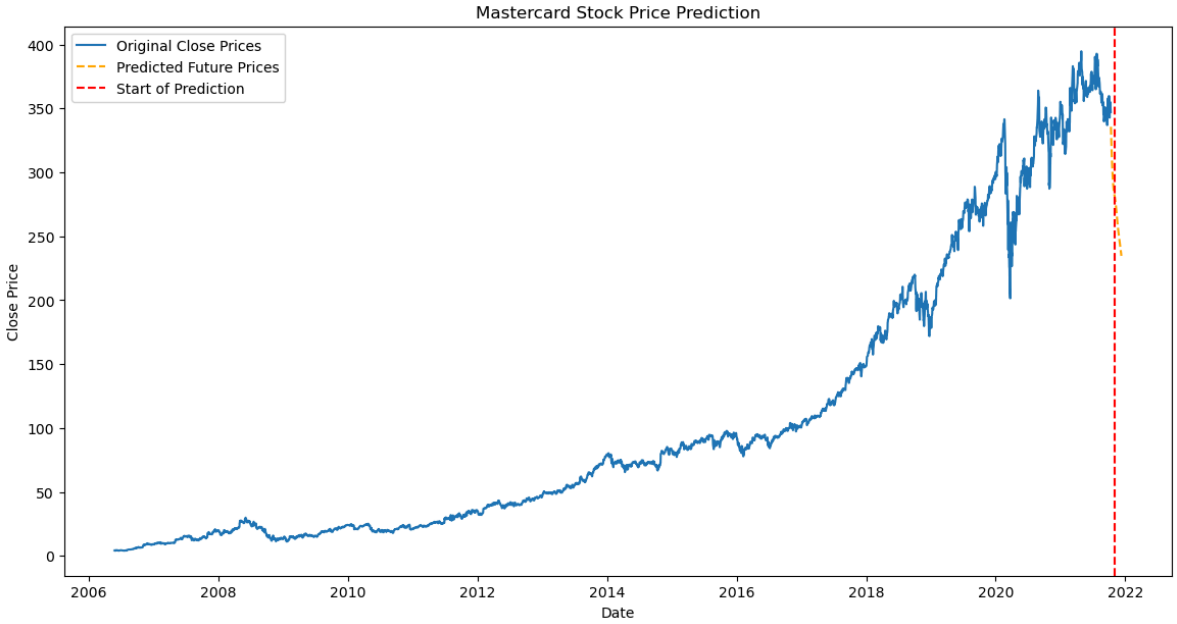
```
# Number of future days to predict (for November and December 2021)
future_days = 60  # Approximately 2 months

# Get the last 'lag' days from the dataset to start predicting future value
last_data = df['Close'].values[-80:].reshape(-1, 1)
last_data_scaled = sc.transform(last_data)

# Create an empty list to store future predictions
future_predictions = []

# Use the last 'lag' days to predict the next 'future_days' values
for _ in range(future_days):
    # Prepare the input data
    input_data = last_data_scaled[-80:]
    input_data = np.reshape(input_data, (1, input_data.shape[0], 1))

    # Predict the next value
    predicted_value = model.predict(input_data)

    # Append the predicted value to the future_predictions list
    future_predictions.append(predicted_value[0, 0])

    # Append the predicted value to the last_data_scaled array for the next
    last_data_scaled = np.append(last_data_scaled, predicted_value[0, 0])
    last_data_scaled = last_data_scaled.reshape(-1, 1)

# Inverse transform the predicted values to get them back to the original s
future_predictions = sc.inverse_transform(np.array(future_predictions).resh

# Create a DataFrame for future predictions
future_dates = pd.date_range(start=df.index[-1], periods=future_days + 1, c
future_df = pd.DataFrame(future_predictions, index=future_dates, columns=['

# Extract values for November and December 2021
nov_dec_predictions = future_df[(future_df.index.month == 11) | (future_df.

# Print the predicted values for November and December 2021
print(nov_dec_predictions)

# Plot the results
plt.figure(figsize=(14, 7))
plt.plot(df['Close'], label='Original Close Prices')
plt.plot(future_df, label='Predicted Future Prices', color='orange', linest
plt.axvline(x=pd.to_datetime('2021-11-01'), color='r', linestyle='--', labe
plt.title('Mastercard Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Close Price')
plt.legend()
plt.show()
```

```
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 27ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 26ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 17ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 33ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 47ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 34ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 29ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 16ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```

```
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
1/1 [==============================] - 0s 31ms/step
```

	Predicted Close
2021-11-01	282.210083
2021-11-02	281.291351
2021-11-03	280.372345
2021-11-04	279.424042
2021-11-05	278.430054
2021-11-06	277.417694
2021-11-07	276.336548
2021-11-08	275.172882
2021-11-09	273.911865
2021-11-10	272.585938
2021-11-11	271.177917
2021-11-12	269.685852
2021-11-13	268.137390
2021-11-14	266.570953
2021-11-15	265.015930
2021-11-16	263.489258
2021-11-17	261.994446
2021-11-18	260.539886
2021-11-19	259.127106
2021-11-20	257.757568
2021-11-21	256.432526
2021-11-22	255.149826
2021-11-23	253.906036
2021-11-24	252.696854
2021-11-25	251.518295
2021-11-26	250.367386
2021-11-27	249.243744
2021-11-28	248.141083
2021-11-29	247.051361
2021-11-30	245.965973
2021-12-01	244.880020
2021-12-02	243.787964
2021-12-03	242.684998
2021-12-04	241.569305
2021-12-05	240.440933
2021-12-06	239.300293
2021-12-07	238.150787
2021-12-08	236.997437
2021-12-09	235.845230
2021-12-10	234.696930

```
C:\Users\Asus\AppData\Local\Temp\ipykernel_20708\2045876864.py:31: FutureW
arning:

Argument `closed` is deprecated in favor of `inclusive`.
```


Mastercard Stock Price Prediction
Original Close Prices
Predicted Future Prices
Start of Prediction
Close Price
Date
0
50
100
150
200
250
300
350
400
2006
2008
2010
2012
2014
2016
2018
2020
2022

In [ ]: