In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error, mean_absolute_percentage_error,r2_scor
```

Data Preparation

In [2]:
```
data = pd.read_csv("Apple.csv")
print(data.head())
```

```
         Date     Close     Volume      Open      High       Low
0  02/28/2020   $273.36  106721200   $257.26   $278.41   $256.37
1  02/27/2020   $273.52   80151380    $281.1      $286   $272.96
2  02/26/2020   $292.65   49678430   $286.53   $297.88    $286.5
3  02/25/2020   $288.08   57668360   $300.95   $302.53   $286.13
4  02/24/2020   $298.18   55548830   $297.26   $304.18   $289.23
```

In [3]:
```
viz = data.copy()
```

In [4]:
```
data['Date'] = pd.to_datetime(data['Date']).astype(int)
```

In [5]:
```
data
```

Out[5]:

	Date	Close	Volume	Open	High	Low
0	1582848000000000000	$273.36	106721200	$257.26	$278.41	$256.37
1	1582761600000000000	$273.52	80151380	$281.1	$286	$272.96
2	1582675200000000000	$292.65	49678430	$286.53	$297.88	$286.5
3	1582588800000000000	$288.08	57668360	$300.95	$302.53	$286.13
4	1582502400000000000	$298.18	55548830	$297.26	$304.18	$289.23
...	...	...	...	...	...	...
2513	1267747200000000000	$31.2786	224647427	$30.7057	$31.3857	$30.6614
2514	1267660800000000000	$30.1014	89591907	$29.8971	$30.1314	$29.8043
2515	1267574400000000000	$29.9043	92846488	$29.8486	$29.9814	$29.7057
2516	1267488000000000000	$29.8357	141486282	$29.99	$30.1186	$29.6771
2517	1267401600000000000	$29.8557	137312041	$29.3928	$29.9286	$29.35

2518 rows × 6 columns

In [6]:
```
print(data.columns)
```

```
Index(['Date', ' Close', ' Volume', ' Open', ' High', ' Low'], dtype='object')
```

In [7]:
```
dollar_columns = [' Close',' Open',' High',' Low']

def remove_dollars_and_convert(value):
    if isinstance(value, str):
        return float(value.replace('$', '').replace(',', ''))
    return value

for column in dollar_columns:
    data[column] = data[column].apply(remove_dollars_and_convert)
```

In [8]:
```
print(data)
```

```
                    Date     Close     Volume      Open      High       Low
0     1582848000000000000  273.3600  106721200  257.2600  278.4100  256.3700
1     1582761600000000000  273.5200   80151380  281.1000  286.0000  272.9600
2     1582675200000000000  292.6500   49678430  286.5300  297.8800  286.5000
3     1582588800000000000  288.0800   57668360  300.9500  302.5300  286.1300
4     1582502400000000000  298.1800   55548830  297.2600  304.1800  289.2300
...                   ...       ...        ...       ...       ...       ...
2513  1267747200000000000   31.2786  224647427   30.7057   31.3857   30.6614
2514  1267660800000000000   30.1014   89591907   29.8971   30.1314   29.8043
2515  1267574400000000000   29.9043   92846488   29.8486   29.9814   29.7057
2516  1267488000000000000   29.8357  141486282   29.9900   30.1186   29.6771
2517  1267401600000000000   29.8557  137312041   29.3928   29.9286   29.3500

[2518 rows x 6 columns]
```

In [9]:
```
data.isnull().sum()
```

Out[9]:
```
Date      0
 Close    0
 Volume   0
 Open     0
 High     0
 Low      0
dtype: int64
```

In [10]:
```
data.shape
```

Out[10]:
```
(2518, 6)
```

In [11]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2518 entries, 0 to 2517
Data columns (total 6 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Date     2518 non-null   int64
 1    Close   2518 non-null   float64
 2    Volume  2518 non-null   int64
 3    Open    2518 non-null   float64
 4    High    2518 non-null   float64
 5    Low     2518 non-null   float64
dtypes: float64(4), int64(2)
memory usage: 118.2 KB
```

Train_test Split

In [12]:
```
train, test = train_test_split(data, test_size = 0.2)
```

In [13]:
```
test_pred = test.copy()
```

In [14]:
```
train.head(10)
```

Out[14]:

	Date	Close	Volume	Open	High	Low
2042	1326758400000000000	60.6714	60589792	60.6000	60.8557	60.4228
633	1503360000000000000	159.7800	21564890	158.2300	160.0000	158.0200
2198	1307059200000000000	49.0628	77951813	49.0257	49.3328	48.8586
21	1580256000000000000	324.3400	54149930	324.4500	327.8500	321.3800
2348	1288224000000000000	43.6057	137111911	43.9928	44.0000	42.9857
997	1457913600000000000	102.5200	25058280	101.9100	102.9100	101.7800
2245	1301270400000000000	50.0628	77053503	50.4500	50.6171	50.0628
1548	1388707200000000000	77.2828	98046999	78.9800	79.1000	77.2043
370	1536278400000000000	221.3000	37418910	221.8500	225.3700	220.7100
2416	1279843200000000000	37.1343	133236640	36.7271	37.1971	36.6114

In [15]:
```
test.head(10)
```

Out[15]:

	Date	Close	Volume	Open	High	Low
925	1466726400000000000	93.4000	75219710	92.9100	94.6550	92.6500
2310	1292976000000000000	46.4514	66449697	46.3371	46.5314	46.2214
77	1572998400000000000	257.2400	18966120	256.7700	257.4900	255.3650
1111	1443484800000000000	109.0600	73230280	112.8300	113.5100	107.8600
1526	1391558400000000000	73.2271	82072855	72.3657	73.6114	72.3217
933	1465862400000000000	97.4600	31890740	97.3200	98.4750	96.7500
402	1532390400000000000	193.0000	18680930	192.4500	193.6601	192.0500
1062	1449532800000000000	118.2300	34275420	117.5200	118.6000	116.8600
2095	1319760000000000000	57.8500	80540134	57.5714	58.0500	57.5014
721	1492473600000000000	141.2000	14676420	141.4100	142.0400	141.1100

In [16]:
```
x_train = train[[' Open', ' High', ' Low', ' Volume']]
x_test = test[[' Open', ' High', ' Low', ' Volume']]
```

In [17]:
```
y_train = train[' Close']
y_test = test[' Close']
```

In [18]:
```
x_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 2014 entries, 2042 to 2386
Data columns (total 4 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0    Open    2014 non-null   float64
 1    High    2014 non-null   float64
 2    Low     2014 non-null   float64
 3    Volume  2014 non-null   int64  
dtypes: float64(3), int64(1)
memory usage: 78.7 KB
```

Linear Regression

In [64]:
```
model_lnr = LinearRegression()
model_lnr.fit(x_train, y_train)
```

Out[64]:

▾ LinearRegression

LinearRegression()

In [65]:
```
y_pred = model_lnr.predict(x_test)
```

In [66]:
```
result = model_lnr.predict([[262.000000, 267.899994, 250.029999, 11896100]])
print(result)
```

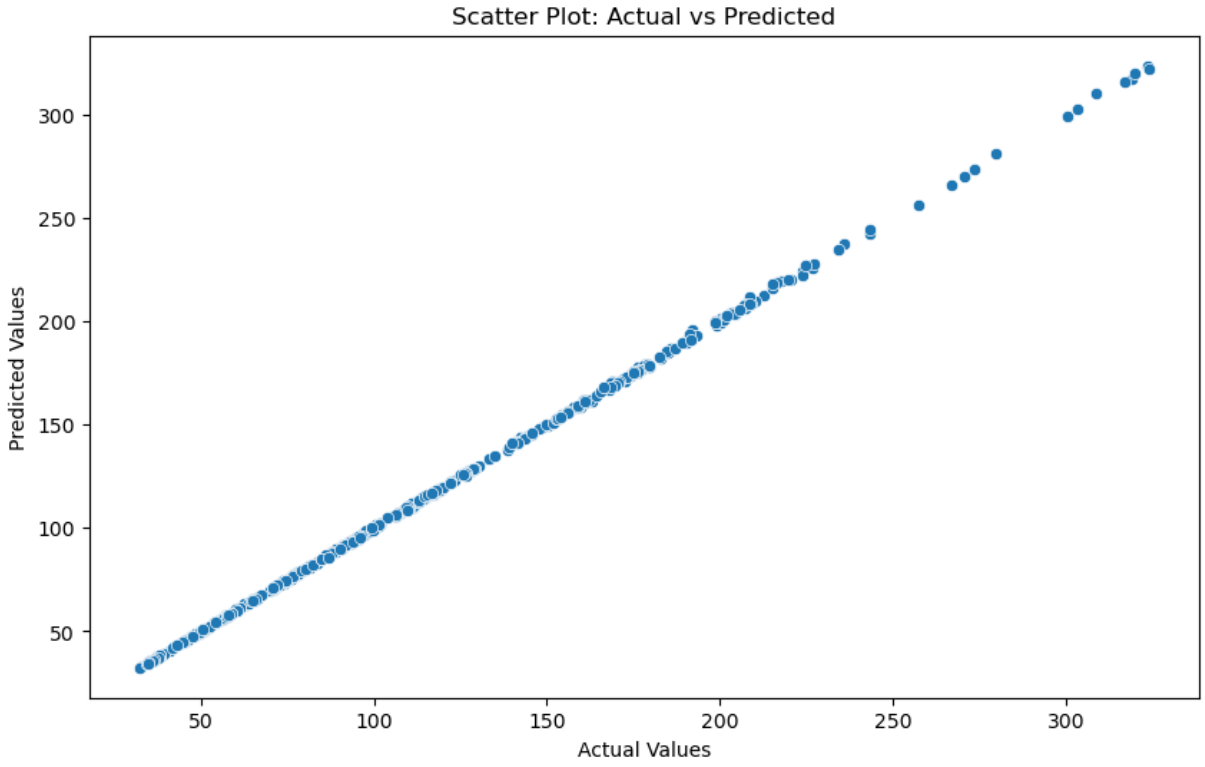
```
[257.58652307]

/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/sklearn/base.py:464: UserWarning: X does not
have valid feature names, but LinearRegression was fitted with feature names
  warnings.warn(
```

Model Evaluation

In [67]:
```
print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 0.411
RMSE 0.641
MAE 0.428
MAPE 0.004
R2 Score :  1.0
```

In [68]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



In [23]:
```
print(y_train)
```

```
2042     60.6714
633     159.7800
2198     49.0628
21      324.3400
2348     43.6057
          ...
1307    109.4100
10      324.8700
1766     64.4014
1433     91.8600
2386     36.9671
Name:  Close, Length: 2014, dtype: float64
```

In [24]:
```
from sklearn.metrics import mean_absolute_error

mae = mean_absolute_error(y_test, y_pred)
mae
```

Out[24]:
```
0.4283861076527482
```

In [25]:
```
y_train.dtype
```

Out[25]:
```
dtype('float64')
```

In [26]:
```
x_train.shape
```

Out[26]:
```
(2014, 4)
```

In [27]:
```
y_train.shape
```

Out[27]:
```
(2014,)
```

Model Visualization

In [28]:
```
def style():
    plt.figure(facecolor='black', figsize=(15,10))
    ax = plt.axes()

    ax.tick_params(axis='x', colors='white')
    ax.tick_params(axis='y', colors='white')

    ax.spines['left'].set_color('white')

    ax.spines['bottom'].set_color('white')
    ax.set_facecolor("black")
```

In [29]:
```
viz['Date']=pd.to_datetime(viz['Date'],format='%m/%d/%Y')
```

In [30]:
```
data = pd.DataFrame(viz[['Date',' Close']])
data=data.reset_index()
data=data.drop('index',axis=1)
data.set_index('Date', inplace=True)
data = data.asfreq('D')
data
```

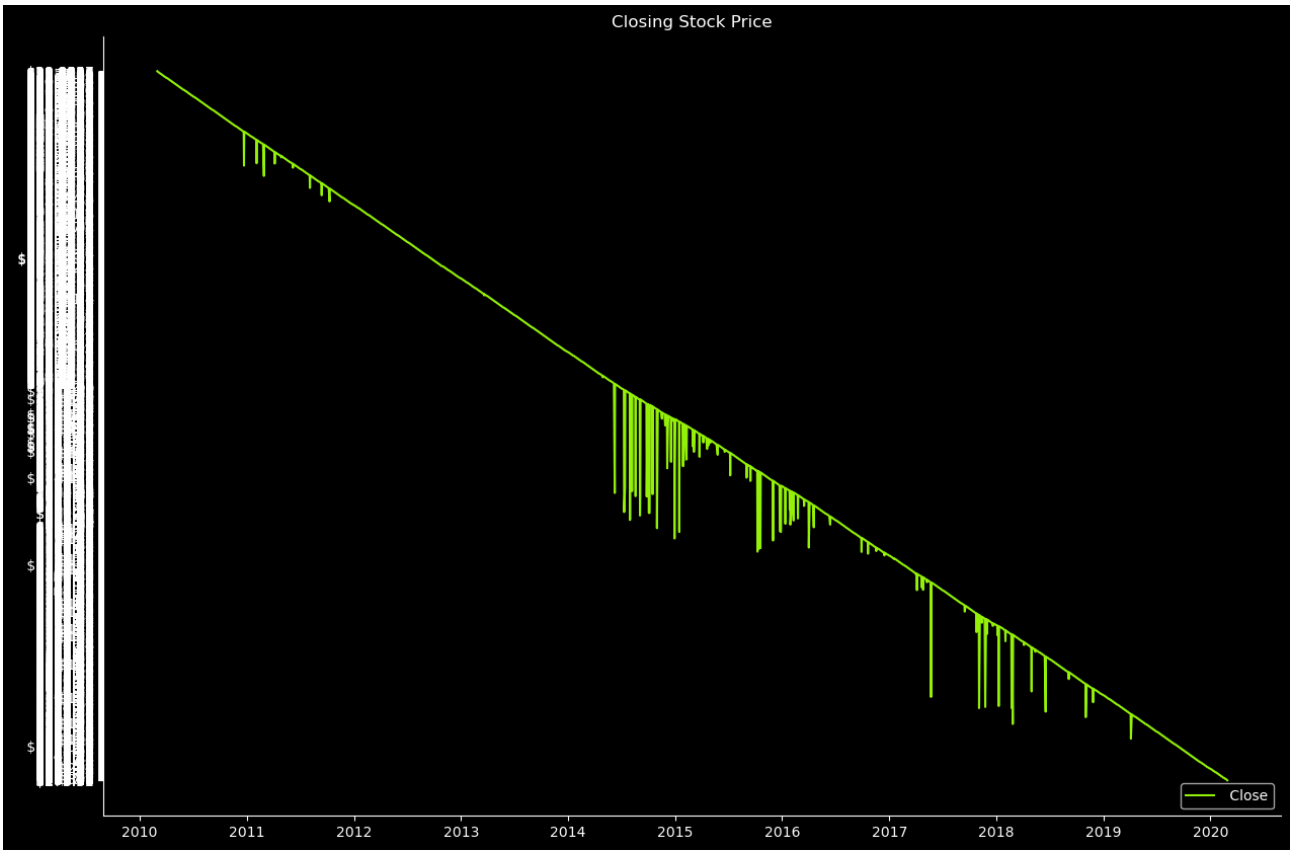
Out[30]:

Date	Close
2010-03-01	$29.8557
2010-03-02	$29.8357
2010-03-03	$29.9043
2010-03-04	$30.1014
2010-03-05	$31.2786
...	...
2020-02-24	$298.18
2020-02-25	$288.08
2020-02-26	$292.65
2020-02-27	$273.52
2020-02-28	$273.36

3652 rows × 1 columns

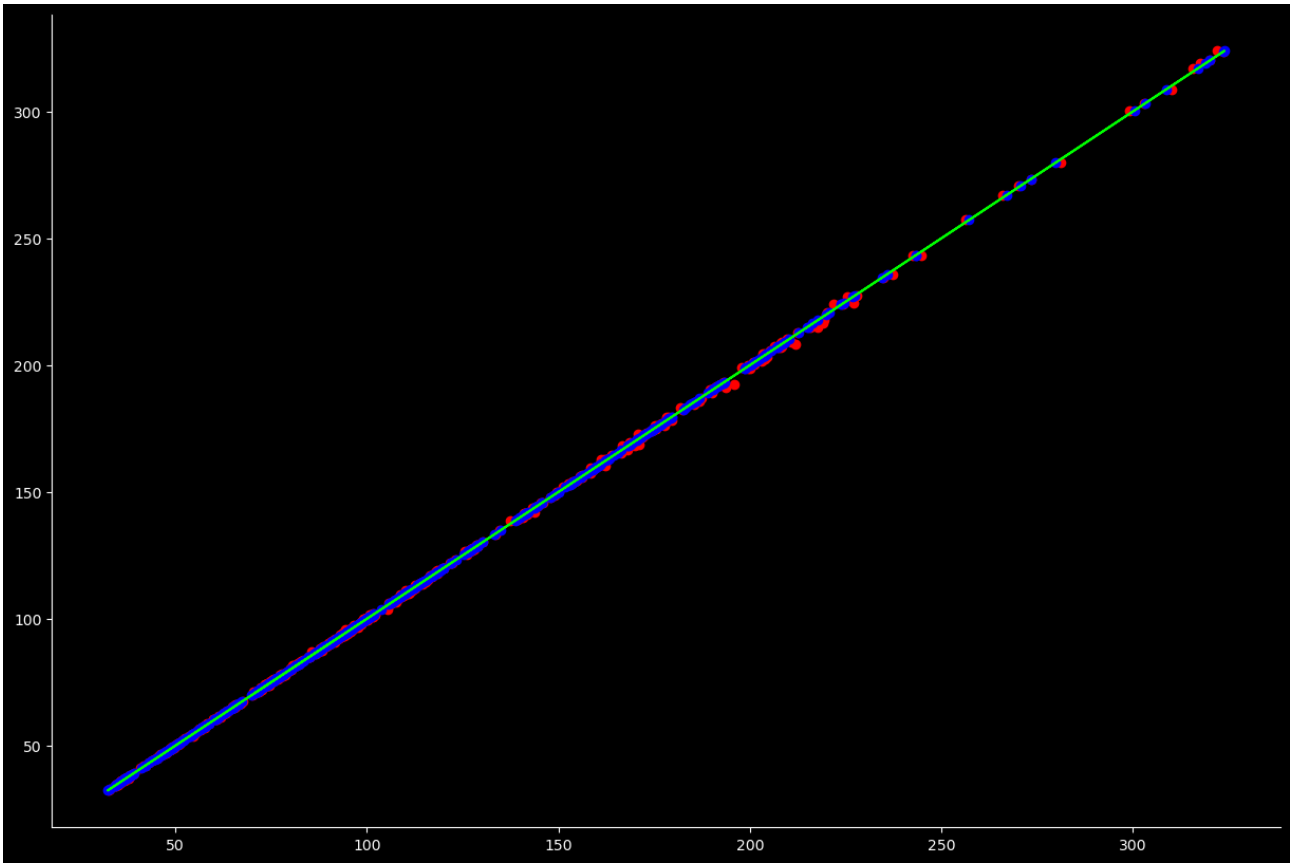
In [31]:
```
style()

plt.title('Closing Stock Price', color="white")
plt.plot(viz.Date, viz[' Close'], color="#94F008")
plt.legend([" Close"], loc ="lower right", facecolor='black', labelcolor='white')
```

Out[31]: <matplotlib.legend.Legend at 0x131ea7e50>



In [32]:
```
style()

plt.scatter(y_pred, y_test, color='red', marker='o')
plt.scatter(y_test, y_test, color='blue')
plt.plot(y_test, y_test, color='lime')
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x1343f6650>]
```



In [33]:
```
test_pred['Close_Prediction'] = y_pred
test_pred
```

Out[33]:

	Date	Close	Volume	Open	High	Low	Close_Prediction
925	1466726400000000000	93.4000	75219710	92.9100	94.6550	92.6500	94.106255
2310	1292976000000000000	46.4514	66449697	46.3371	46.5314	46.2214	46.404254
77	1572998400000000000	257.2400	18966120	256.7700	257.4900	255.3650	256.306260
1111	1443484800000000000	109.0600	73230280	112.8300	113.5100	107.8600	109.599013
1526	1391558400000000000	73.2271	82072855	72.3657	73.6114	72.3217	73.328380
...	...	...	...	...	...	...	...
483	1522195200000000000	166.4800	41464880	167.2500	170.0200	165.1900	167.895080
1725	1366675200000000000	58.0186	150492275	57.7128	58.3401	56.9728	57.666882
1104	1444262400000000000	109.5000	61747260	110.1900	110.1900	108.2100	108.695814
1171	1436140800000000000	126.0000	27972950	124.9400	126.2300	124.8500	125.902757
1604	1381708800000000000	70.8628	65216780	69.9757	71.0828	69.9071	70.805943

504 rows × 7 columns

In [34]:
```
test_pred[[' Close', 'Close_Prediction']].describe().T
```

Out[34]:

	count	mean	std	min	25%	50%	75%	max
Close	504.0	113.936761	59.980312	32.378600	66.338550	99.30500	155.642500	323.870000
Close_Prediction	504.0	113.972551	60.016081	32.498897	66.263182	99.16654	155.663686	323.668038

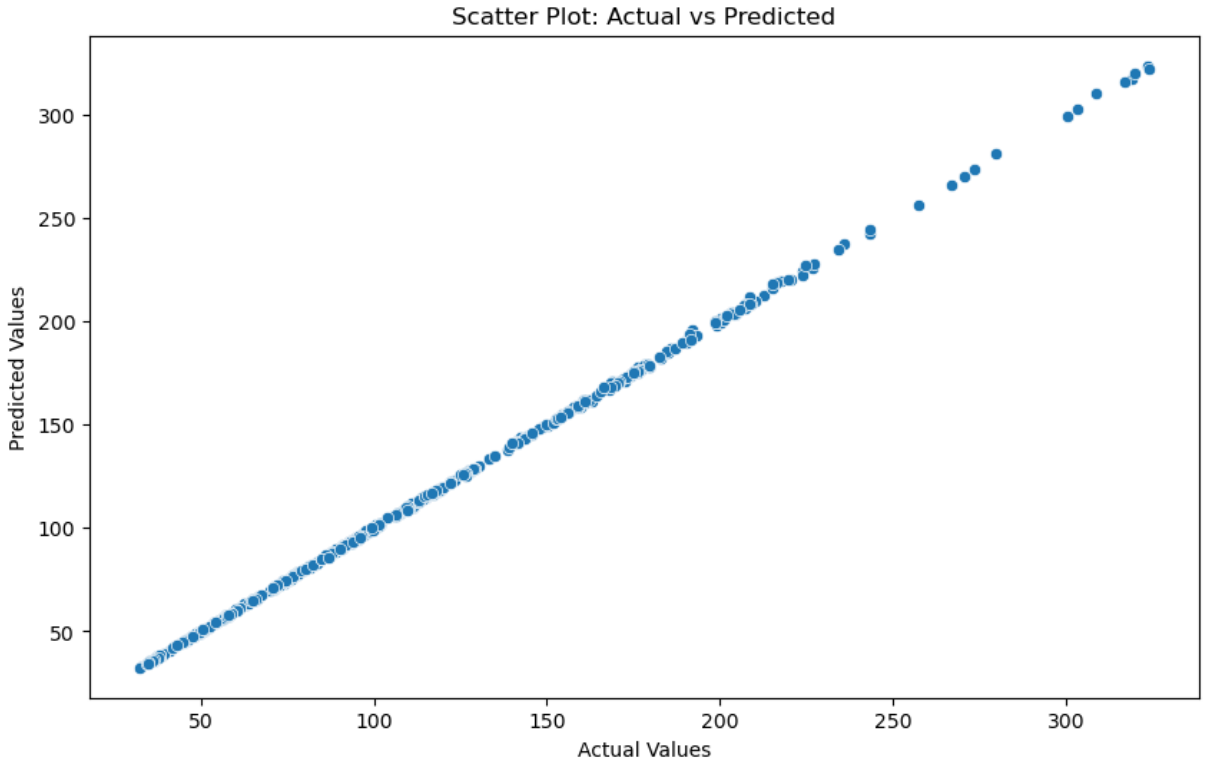
In [ ]:

Ridge

In [62]:
```
from sklearn.linear_model import Ridge

alpha_value = 1.0  # You can adjust the regularization strength (alpha) as needed
model_ridge = Ridge(alpha=alpha_value)

model_ridge.fit(x_train, y_train)

# Make predictions on the test set
y_pred= model_ridge.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 0.411
RMSE 0.641
MAE 0.428
MAPE 0.004
R2 Score :  1.0
```

In [63]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

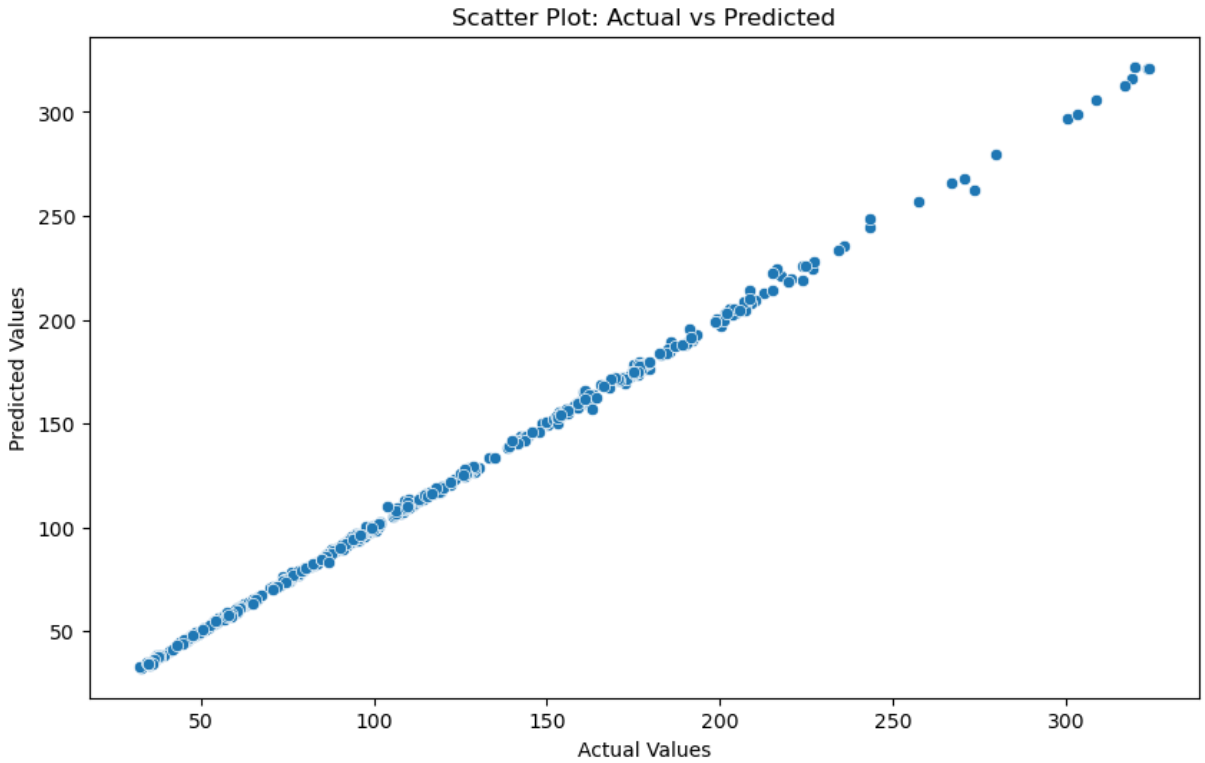


Lasso

In [60]:
```
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Create a Lasso Regression model
alpha_value = 1.0  # You can adjust the regularization strength (alpha) as needed
model_lasso = Lasso(alpha=alpha_value)

# Train the Lasso Regression model
model_lasso.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model_lasso.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 2.184
RMSE 1.478
MAE 0.908
MAPE 0.008
R2 Score :  0.999

/Users/lokeshkollepara/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_coordinate_descent.py:
628: ConvergenceWarning: Objective did not converge. You might want to increase the number of iterations,
check the scale of the features or consider increasing regularisation. Duality gap: 1.654e+03, tolerance:
7.452e+02
  model = cd_fast.enet_coordinate_descent(
```

In [61]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

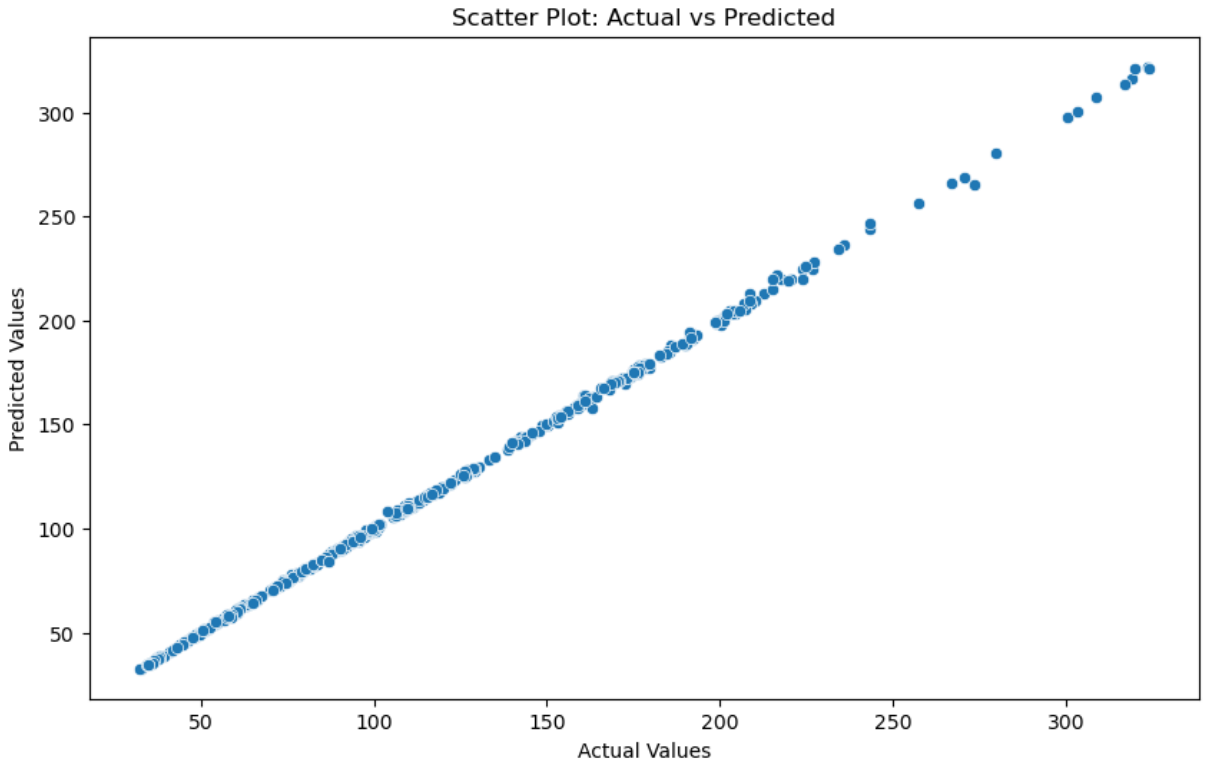


Support Vector Regressor

In [56]:
```
from sklearn.svm import SVR
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

# Create a Support Vector Regression model
model_svr = SVR(kernel='linear', C=1.0, epsilon=0.1)
# You can choose different kernels (e.g., 'linear', 'rbf', 'poly') and adjust parameters like C and epsilon

# Train the SVR model
model_svr.fit(x_train_scaled, y_train)

# Make predictions on the scaled test set
y_pred = model_svr.predict(x_test_scaled)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.138
RMSE 1.067
MAE 0.665
MAPE 0.006
R2 Score :  1.0
```

In [58]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

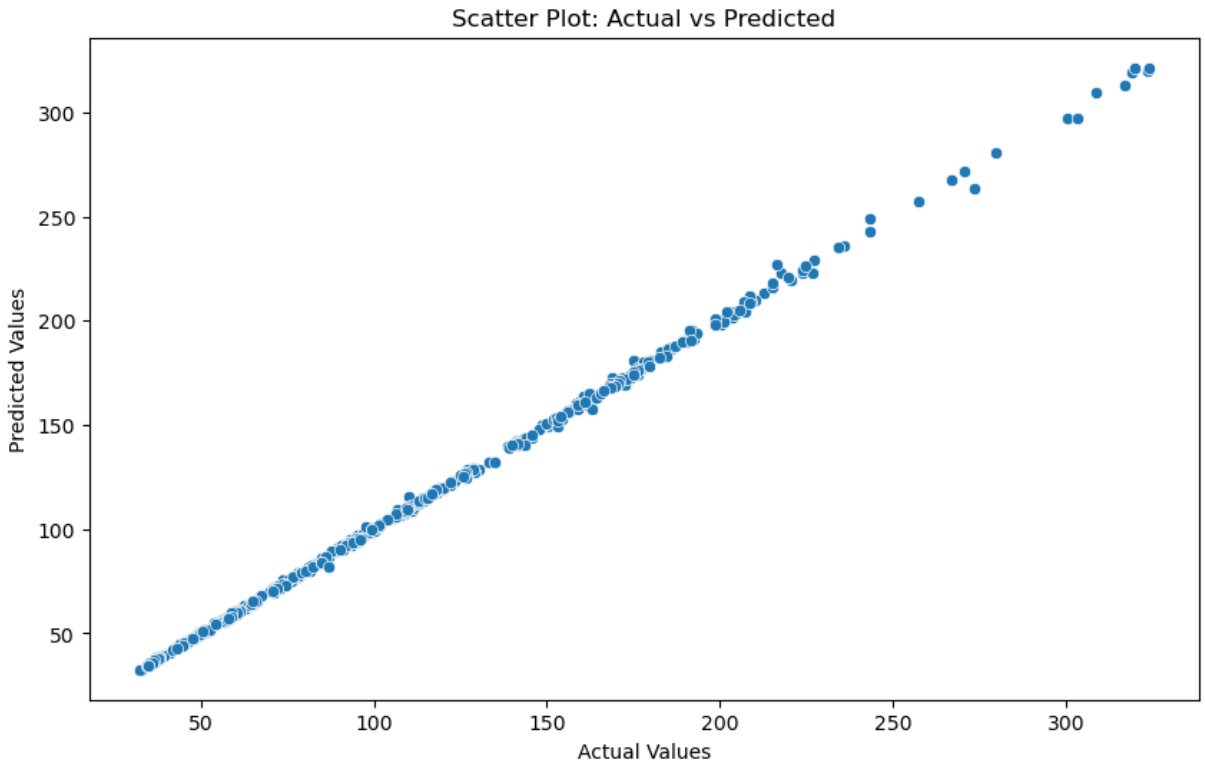


Decision Tree Regressor

In [54]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Create a Decision Tree Regressor model
model_dt = DecisionTreeRegressor(max_depth=None, random_state=42)
# You can adjust parameters like max_depth, min_samples_split, etc.

# Train the Decision Tree Regressor model
model_dt.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model_dt.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.868
RMSE 1.367
MAE 0.823
MAPE 0.007
R2 Score :  0.999
```

In [55]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

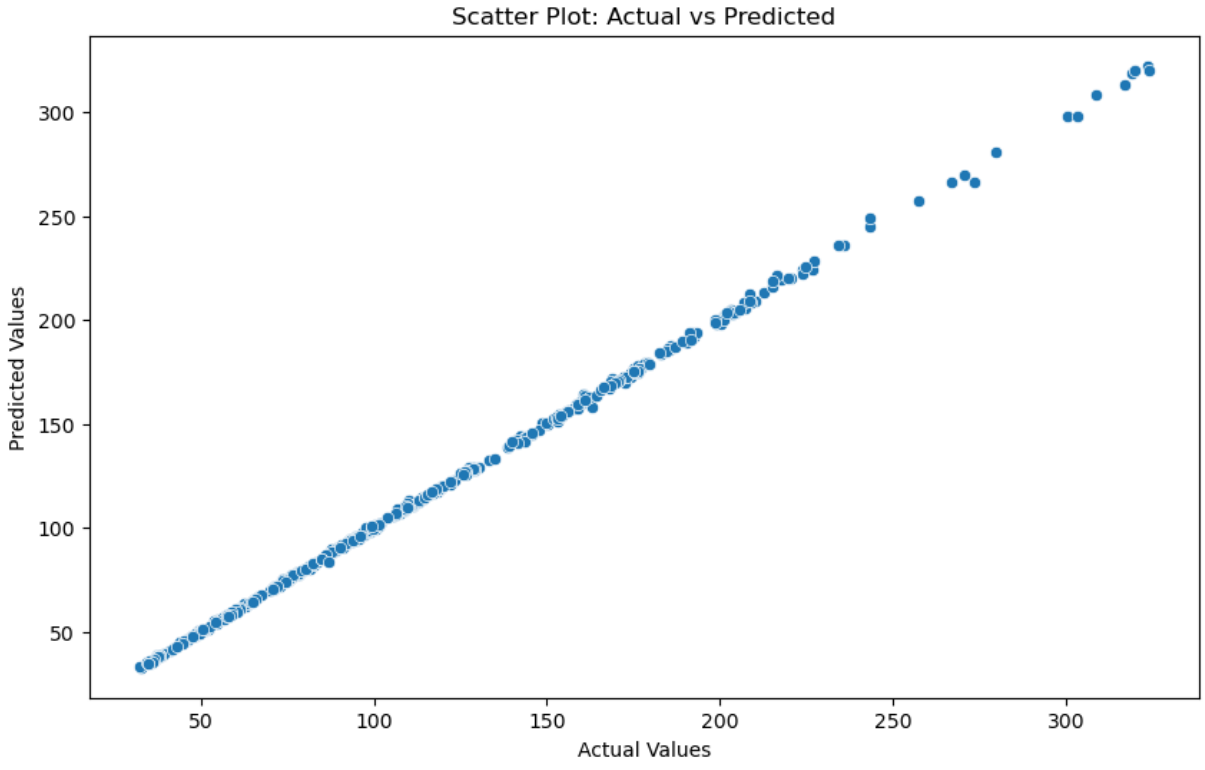


Random Forest Regressor

In [52]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Create a Random Forest Regressor model
model_rf = RandomForestRegressor(n_estimators=100, random_state=42)
# You can adjust parameters like n_estimators, max_depth, min_samples_split, etc.

# Train the Random Forest Regressor model
model_rf.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model_rf.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.06
RMSE 1.03
MAE 0.645
MAPE 0.006
R2 Score :  1.0
```

In [53]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

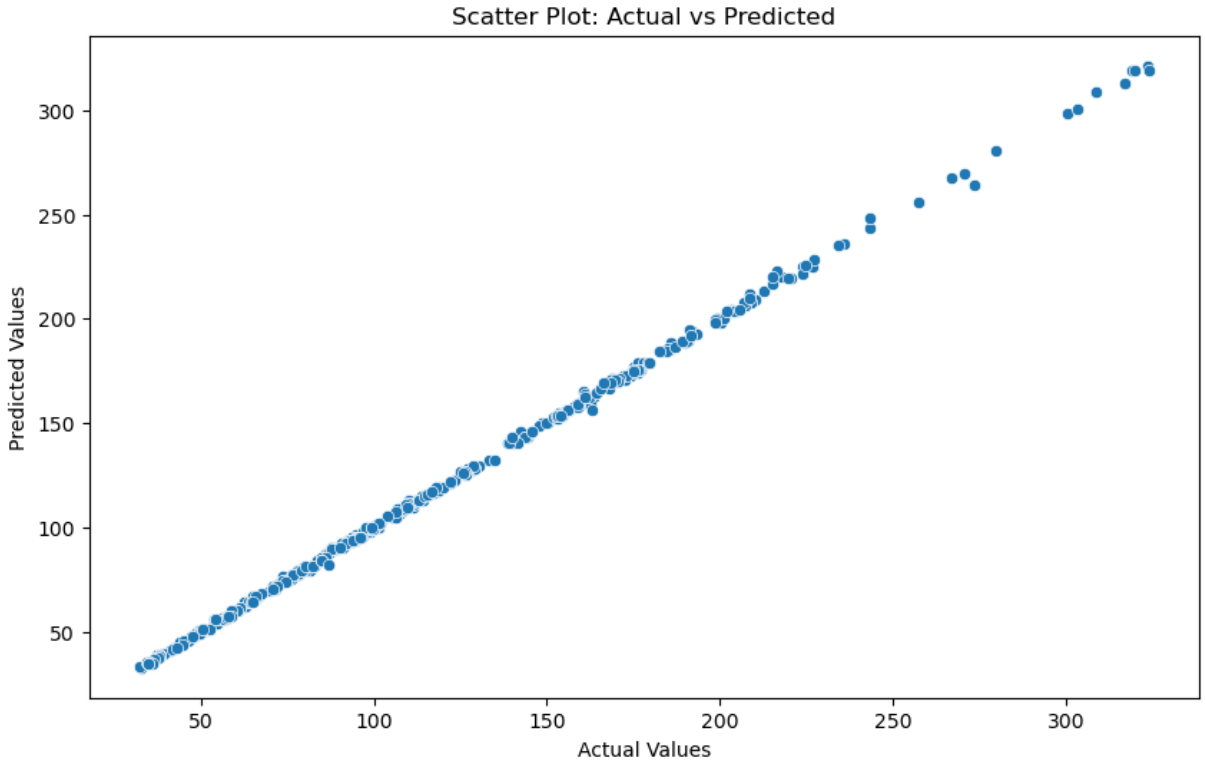


Gradient Boosting Regressorr

In [50]:
```
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import mean_squared_error

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Create a Gradient Boosting Regressor model
model_gb = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth=3, random_state=42)
# You can adjust parameters like n_estimators, learning_rate, max_depth, etc.

# Train the Gradient Boosting Regressor model
model_gb.fit(x_train, y_train)

# Make predictions on the test set
y_pred = model_gb.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.571
RMSE 1.253
MAE 0.833
MAPE 0.008
R2 Score :  1.0
```

In [51]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

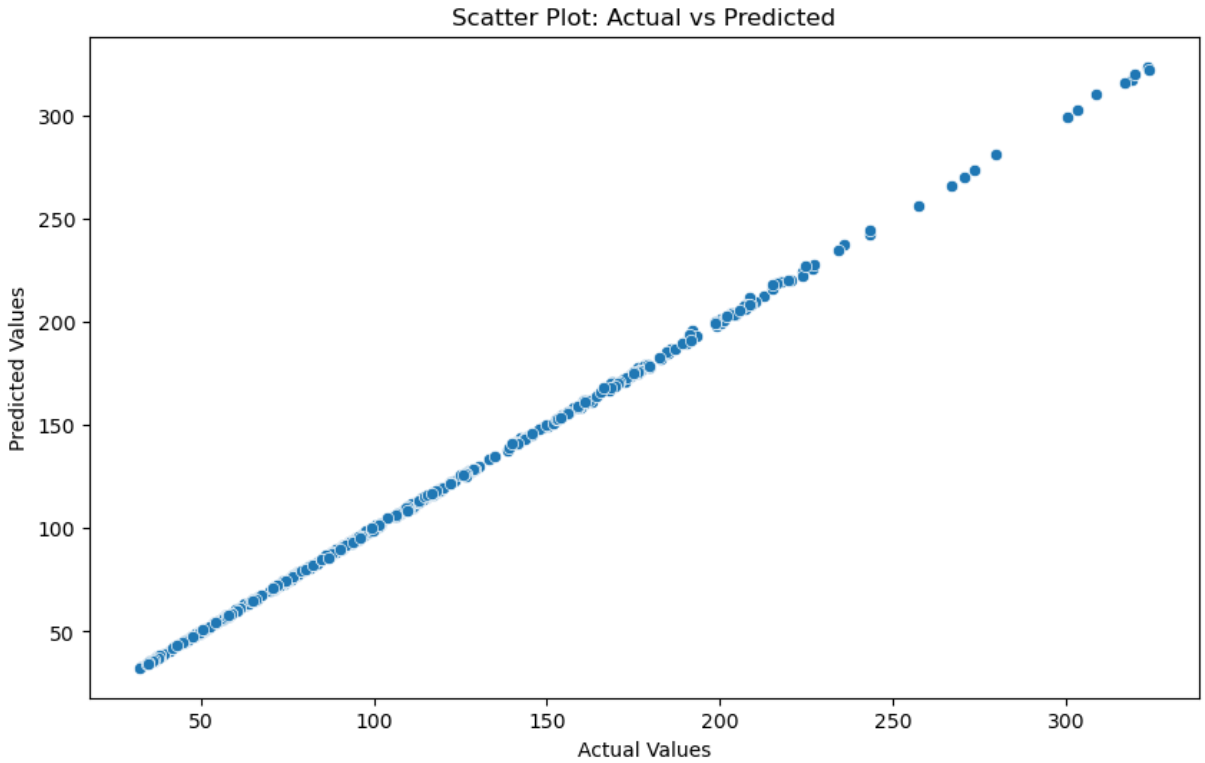


OLS(Ordinary Least Squares)

In [41]:
```
import statsmodels.api as sm
from sklearn.metrics import mean_squared_error
import numpy as np

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Add a constant term to the features matrix for the intercept
x_train_ols = sm.add_constant(x_train)
x_test_ols = sm.add_constant(x_test)

# Create an OLS model
model_ols = sm.OLS(y_train, x_train_ols)

# Train the OLS model
result_ols = model_ols.fit()

# Make predictions on the test set
y_pred_ols = result_ols.predict(x_test_ols)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.571
RMSE 1.253
MAE 0.833
MAPE 0.008
R2 Score :  1.0
```

In [49]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_ols)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```

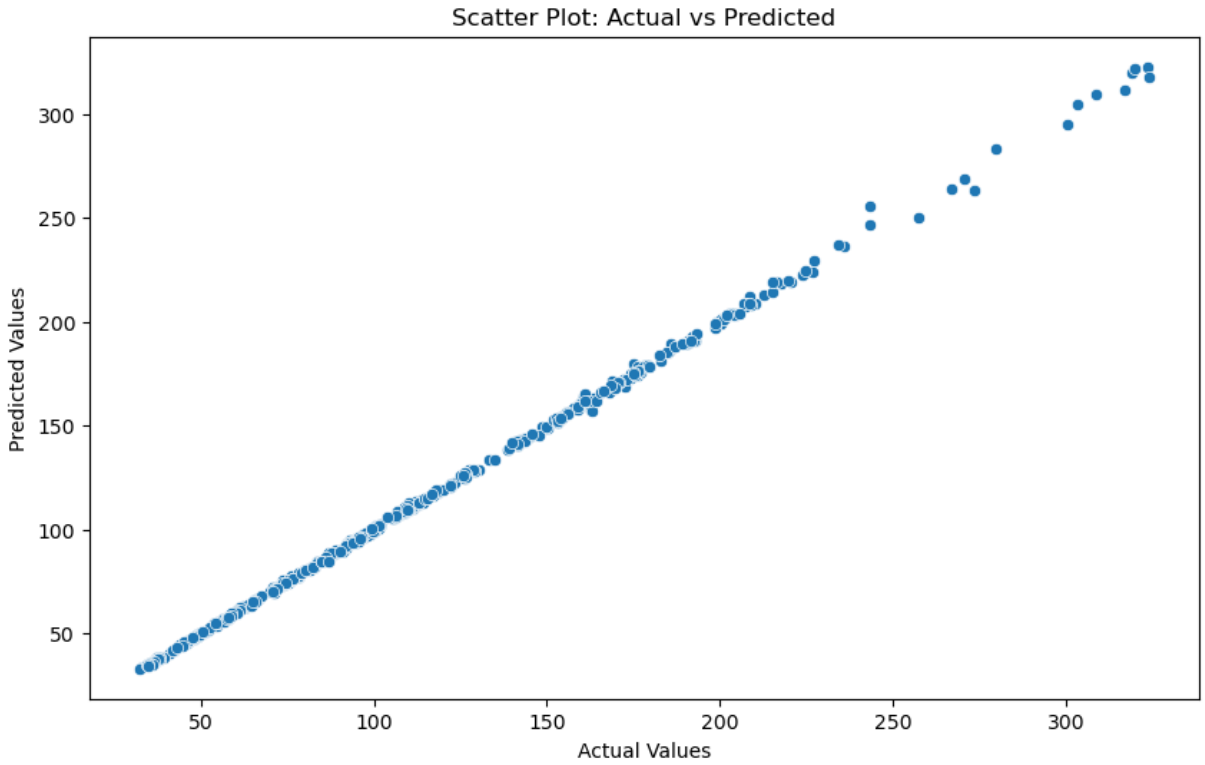


Xgboost

In [46]:
```
import xgboost as xgb
from sklearn.metrics import mean_squared_error

# Assuming x_train, y_train, x_test are your training features, training labels, and test features

# Create an XGBoost regression model
model_xgboost = xgb.XGBRegressor()

# Train the XGBoost model
model_xgboost.fit(x_train, y_train)

# Make predictions on the test set
y_pred_xgboost = model_xgboost.predict(x_test)

print("MSE",round(mean_squared_error(y_test,y_pred), 3))
print("RMSE",round(np.sqrt(mean_squared_error(y_test,y_pred)), 3))
print("MAE",round(mean_absolute_error(y_test,y_pred), 3))
print("MAPE",round(mean_absolute_percentage_error(y_test,y_pred), 3))
print("R2 Score : ", round(r2_score(y_test,y_pred), 3))
```

```
MSE 1.571
RMSE 1.253
MAE 0.833
MAPE 0.008
R2 Score :  1.0
```

In [48]:
```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test, y=y_pred_xgboost)
plt.title('Scatter Plot: Actual vs Predicted')
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.show()
```



In [ ]: