In [1]:
```
# Quant Project 2: Linear Regression Modeling for predicting the Stock P

# Step 1: Download the data from Yahoo Finance
# Step 2: Some Feature Engineering (to build new features) - Technical I
# Step 3: Run Linear Regression Model
# Step 4: Check how the model is performed (Actual vs Predicted)
# Step 5: Test for all the assumptions
# Step 6: Check the prediction
```

In [2]:
```
# Install:
# !pip install yfinance or
# pip install yfinance or
# conda install yfinance
```

In [3]:
```
import yfinance as yf
```

In [4]:
```
# Step 1: Download the data from Yahoo Finance
tickers = ['AAPL', 'AMZN', 'MSFT', 'QQQ', '^GSPC']
df = yf.download(tickers, start = '2020-01-01', end = '2024-12-31')['Clo
df
```

```
YF.download() has changed argument auto_adjust default to True

[*********************100%***********************]  5 of 5 completed
```

Out[4]:

Ticker	AAPL	AMZN	MSFT	QQQ	^GSPC
Date					
2020-01-02	72.716064	94.900497	153.323242	209.325882	3257.850098
2020-01-03	72.009132	93.748497	151.414139	207.408524	3234.850098
2020-01-06	72.582916	95.143997	151.805481	208.744888	3246.280029
2020-01-07	72.241539	95.343002	150.421371	208.715851	3237.179932
2020-01-08	73.403648	94.598503	152.817322	210.284607	3253.050049
...	...	...	...	...	...
2024-12-23	254.989655	225.059998	434.379028	522.091431	5974.069824
2024-12-24	257.916443	229.050003	438.450836	529.170898	6040.040039
2024-12-26	258.735504	227.050003	437.233276	528.811401	6037.589844
2024-12-27	255.309296	223.750000	429.668457	521.781860	5970.839844
2024-12-30	251.923019	221.300003	423.979858	514.842224	5906.939941

1257 rows × 5 columns

In [5]:
```
# Step 2: Perform Feature Engineering

# Lesson: To predict AAPL Stock price, we have to consider yesterday's p
# The market is not open yet so we don't know what's the price today

# Considering Yesterday's Value
df['AAPL(t-1)'] = df['AAPL'].shift(1)
df['AMZN(t-1)'] = df['AMZN'].shift(1)
df['MSFT(t-1)'] = df['MSFT'].shift(1)
df['QQQ(t-1)'] = df['QQQ'].shift(1)
df['^GSPC(t-1)'] = df['^GSPC'].shift(1)

# Moving Avg (MA): Technical Indicator - It helps you understand the sho
df['AAPL_MA_5'] = df['AAPL'].rolling(window=5).mean()
df['AMZN_MA_5'] = df['AMZN'].rolling(window=5).mean()
df['MSFT_MA_5'] = df['MSFT'].rolling(window=5).mean()
df['QQQ_MA_5'] = df['QQQ'].rolling(window=5).mean()
df['^GSPC_MA_5'] = df['^GSPC'].rolling(window=5).mean()

# Set Y Variable - Next day
df['Target'] = df['AAPL'].shift(-1)

df = df.dropna()
```

In [6]:
```
df.columns
```

Out[6]:
```
Index(['AAPL', 'AMZN', 'MSFT', 'QQQ', '^GSPC', 'AAPL(t-1)', 'AMZN(t-
1)',
       'MSFT(t-1)', 'QQQ(t-1)', '^GSPC(t-1)', 'AAPL_MA_5', 'AMZN_MA_5',
       'MSFT_MA_5', 'QQQ_MA_5', '^GSPC_MA_5', 'Target'],
      dtype='object', name='Ticker')
```

In [7]:
```
# Y = Intercept + B1*X1 + B2*X2 + B3*X3
```

In [8]:
```
# Step 3: Fit a Linear Regression Model
# Set X and Y variable for Linear Regression Model - Ordinary Least Squa

import statsmodels.api as sm

X = df[['AAPL(t-1)', 'AMZN(t-1)',
       'MSFT(t-1)', 'QQQ(t-1)', '^GSPC(t-1)', 'AAPL_MA_5', 'AMZN_MA_5',
       'MSFT_MA_5', 'QQQ_MA_5', '^GSPC_MA_5']]

Y = df['Target']

X_const = sm.add_constant(X) # Intercept Term

# Train the Model
model = sm.OLS(Y, X_const).fit()

# Summary
model.summary()
```

Out[8]:

OLS Regression Results

Dep. Variable:	Target	**R-squared:**	0.993
Model:	OLS	**Adj. R-squared:**	0.992
Method:	Least Squares	**F-statistic:**	1.644e+04
Date:	Sun, 27 Apr 2025	**Prob (F-statistic):**	0.00
Time:	03:46:07	**Log-Likelihood:**	-3385.6
No. Observations:	1252	**AIC:**	6793.
Df Residuals:	1241	**BIC:**	6850.
Df Model:	10		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
const	0.5116	1.147	0.446	0.656	-1.739	2.762
AAPL(t-1)	0.4682	0.081	5.764	0.000	0.309	0.628
AMZN(t-1)	0.0647	0.068	0.958	0.338	-0.068	0.197
MSFT(t-1)	-0.0165	0.052	-0.314	0.753	-0.119	0.086
QQQ(t-1)	0.0091	0.102	0.089	0.929	-0.190	0.208
^GSPC(t-1)	0.0053	0.007	0.734	0.463	-0.009	0.019
AAPL_MA_5	0.5158	0.082	6.279	0.000	0.355	0.677
AMZN_MA_5	-0.0594	0.069	-0.866	0.386	-0.194	0.075
MSFT_MA_5	0.0281	0.053	0.531	0.595	-0.076	0.132
QQQ_MA_5	-0.0099	0.104	-0.096	0.924	-0.213	0.194
^GSPC_MA_5	-0.0057	0.007	-0.776	0.438	-0.020	0.009

Omnibus:	26.037	**Durbin-Watson:**	0.808
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	49.707
Skew:	-0.085	**Prob(JB):**	1.61e-11
Kurtosis:	3.961	**Cond. No.**	6.88e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 6.88e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

In [9]:
```
# P value < 0.05 = Variable is significant => Keep that variable
# P value > 0.05 = Variable is not significant => Drop that variable
```

In [10]:
```
# Set X and Y variable for Linear Regression Model - Ordinary Least Squa

import statsmodels.api as sm

X = df[['AAPL(t-1)', '^GSPC(t-1)']] # Dropping AAPL_MA_5
Y = df['Target']
X_const = sm.add_constant(X) # Intercept Term

# Train the Model
model = sm.OLS(Y, X_const).fit()

# Summary
model.summary()
```

Out[10]:

OLS Regression Results

Dep. Variable:	Target	**R-squared:**	0.992
Model:	OLS	**Adj. R-squared:**	0.992
Method:	Least Squares	**F-statistic:**	7.762e+04
Date:	Sun, 27 Apr 2025	**Prob (F-statistic):**	0.00
Time:	03:46:08	**Log-Likelihood:**	-3425.0
No. Observations:	1252	**AIC:**	6856.
Df Residuals:	1249	**BIC:**	6871.
Df Model:	2		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
const	-0.7197	0.751	-0.959	0.338	-2.192	0.753
AAPL(t-1)	0.9840	0.007	140.621	0.000	0.970	0.998
^GSPC(t-1)	0.0008	0.000	2.114	0.035	5.82e-05	0.002

Omnibus:	40.865	**Durbin-Watson:**	1.041
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	103.863
Skew:	-0.036	**Prob(JB):**	2.79e-23
Kurtosis:	4.409	**Cond. No.**	3.08e+04

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.08e+04. This might indicate that there are
strong multicollinearity or other numerical problems.

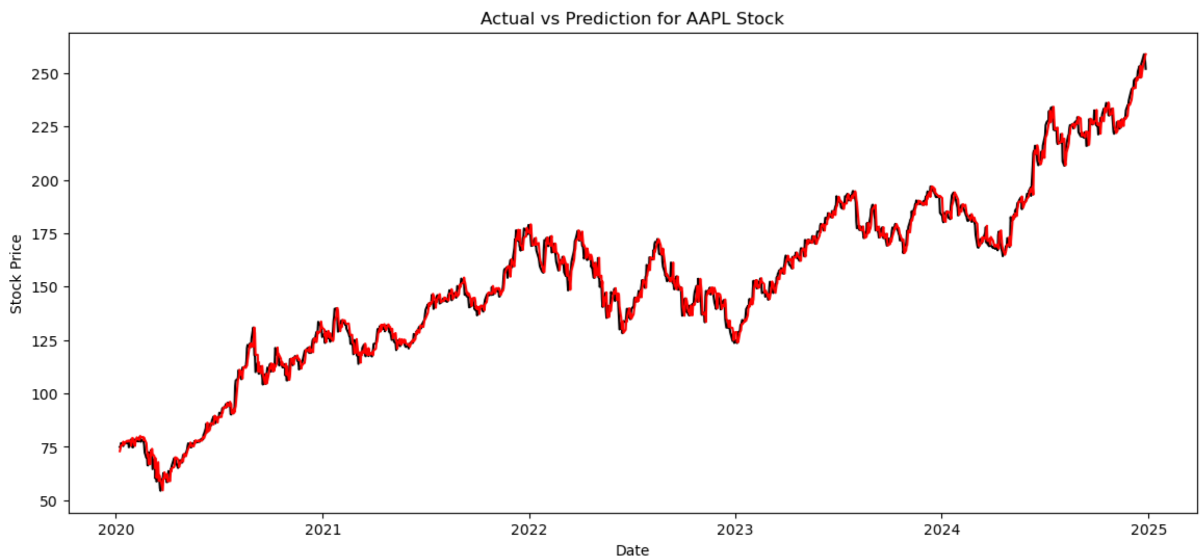
In [11]:
```
import pandas as pd
df_train_predict = pd.DataFrame()
df_train_predict['Actual'] = df['Target']
df_train_predict['Predicted'] = model.predict(X_const)
df_train_predict
```

Out[11]:

	Actual	Predicted
Date		
2020-01-08	74.962807	72.982845
2020-01-09	75.132271	74.139196
2020-01-10	76.737411	75.690917
2020-01-13	75.701218	75.850115
2020-01-14	75.376801	77.447996
...	...	...
2024-12-20	254.989655	249.547433
2024-12-23	257.916443	254.218745
2024-12-24	258.735504	255.020359
2024-12-26	255.309296	257.953658
2024-12-27	251.923019	258.757641

1252 rows × 2 columns

In [12]:
```
# Plot between Actual vs Predicted Value

import matplotlib.pyplot as plt

plt.figure(figsize = (14,6))
plt.plot(df_train_predict.index, df_train_predict['Actual'], label = 'Ac
plt.plot(df_train_predict.index, df_train_predict['Predicted'], label =
plt.title("Actual vs Prediction for AAPL Stock")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.show()
```



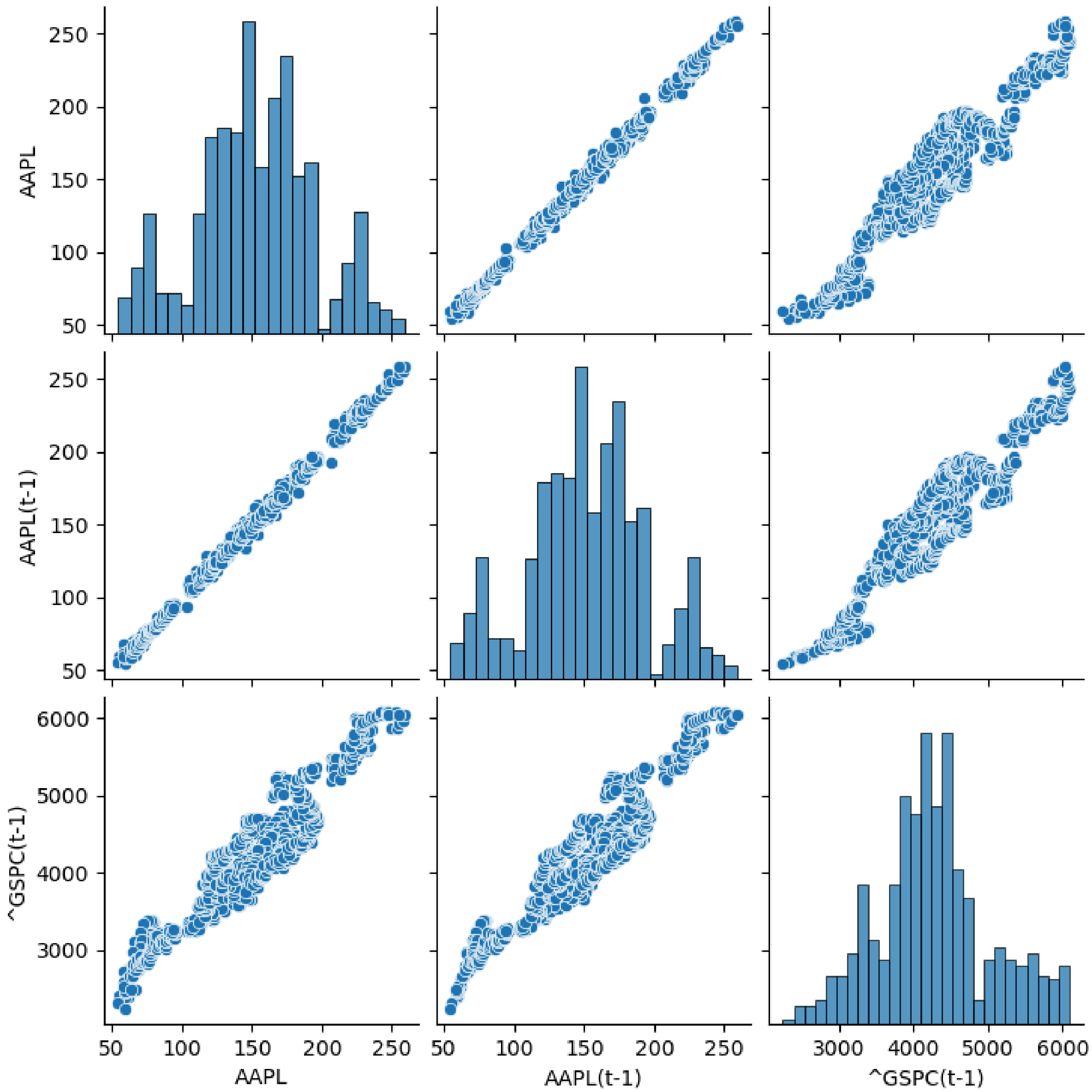
In [13]:
```
# Catch is: We need to still test LR Assumptions
# We need to check how the model is performing on test data
```

In [14]:
```
# Linear Regression:  (Given the dataset here)
# Step 1: Train the Model
# Step 2: Test the Model

# 2 + 2 = 4 - Trained
# 2 + 2 = 4 - Testing
# 2 + 9 = 11 => Good Model
```

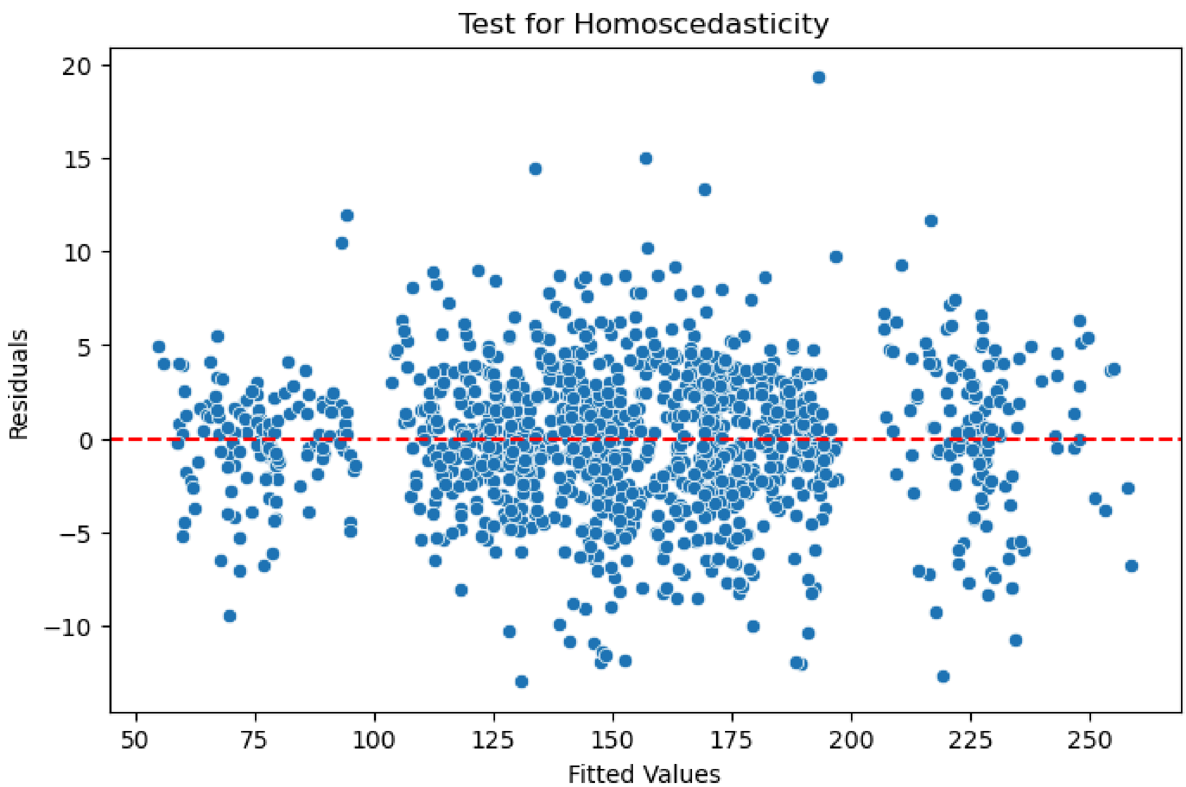
In [15]:
```
# Assumptions of Linear Regression:

# 1) Linearity between dependent and independent variable - Met

import seaborn as sns
df = df[['AAPL', 'AAPL(t-1)', '^GSPC(t-1)']]
sns.pairplot(df)

# AAPL & AAPL(t-1) has linear relationship
# AAPL & S&P500 has linear relationship
# AAPL & AAPL_MA_5 has linear relationship
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: F
utureWarning: use_inf_as_na option is deprecated and will be removed in
a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: F
utureWarning: use_inf_as_na option is deprecated and will be removed in
a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119: F
utureWarning: use_inf_as_na option is deprecated and will be removed in
a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

Out[15]:
```
<seaborn.axisgrid.PairGrid at 0x1718410d0>
```


AAPL
AAPL(t-1)
^GSPC(t-1)
50
100
150
200
250
3000
4000
5000
6000

In [16]:
```
# 2) Homoscedasticity: Fitting Residual with the predicted value

residual = model.resid # Actual - Predicted
fitted = model.fittedvalues  # Predicted Y Value

plt.figure(figsize = (8,5))
sns.scatterplot(x = fitted, y = residual)
plt.axhline(0, color = 'red', linestyle = '--')
plt.title('Test for Homoscedasticity')
plt.xlabel('Fitted Values')
plt.ylabel('Residuals')
plt.show()

# Since it's a tube like structure => It is homoscedastic -> Assumption 
# If it was funnel like structure => It is heteroscedastic
```



In [17]:
```
X_const.columns
```

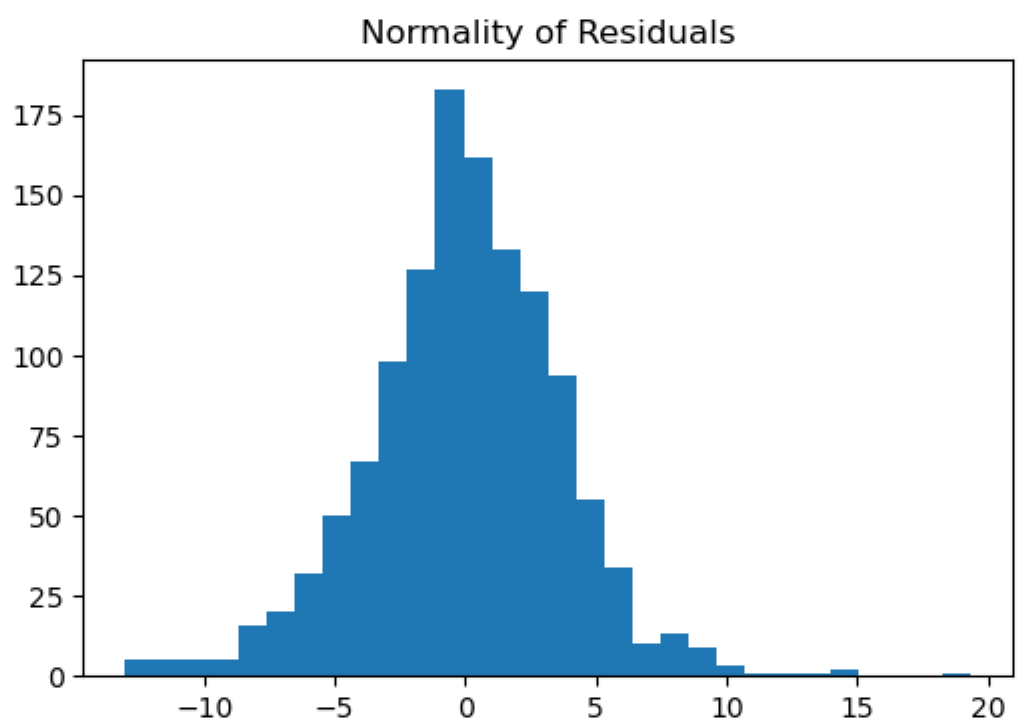
Out[17]:
```
Index(['const', 'AAPL(t-1)', '^GSPC(t-1)'], dtype='object')
```

In [18]:
```
# 3) Multicollinearity => VIF (Variance Inflation Factor) => For indepen

# Rule of thumb for VIF
# VIF < 1 => No Multicollinearity
# VIF < 10 => Moderate Multicollinearity
# VIF > 10 => Strong Multicollinearity

# VIF = Condition is Met

from statsmodels.stats.outliers_influence import variance_inflation_fact

vif = pd.DataFrame()
vif['Features'] = X_const.columns
vif['VIF'] = [variance_inflation_factor(X_const.values, i) for i in rang
vif = vif[1:]
vif
```

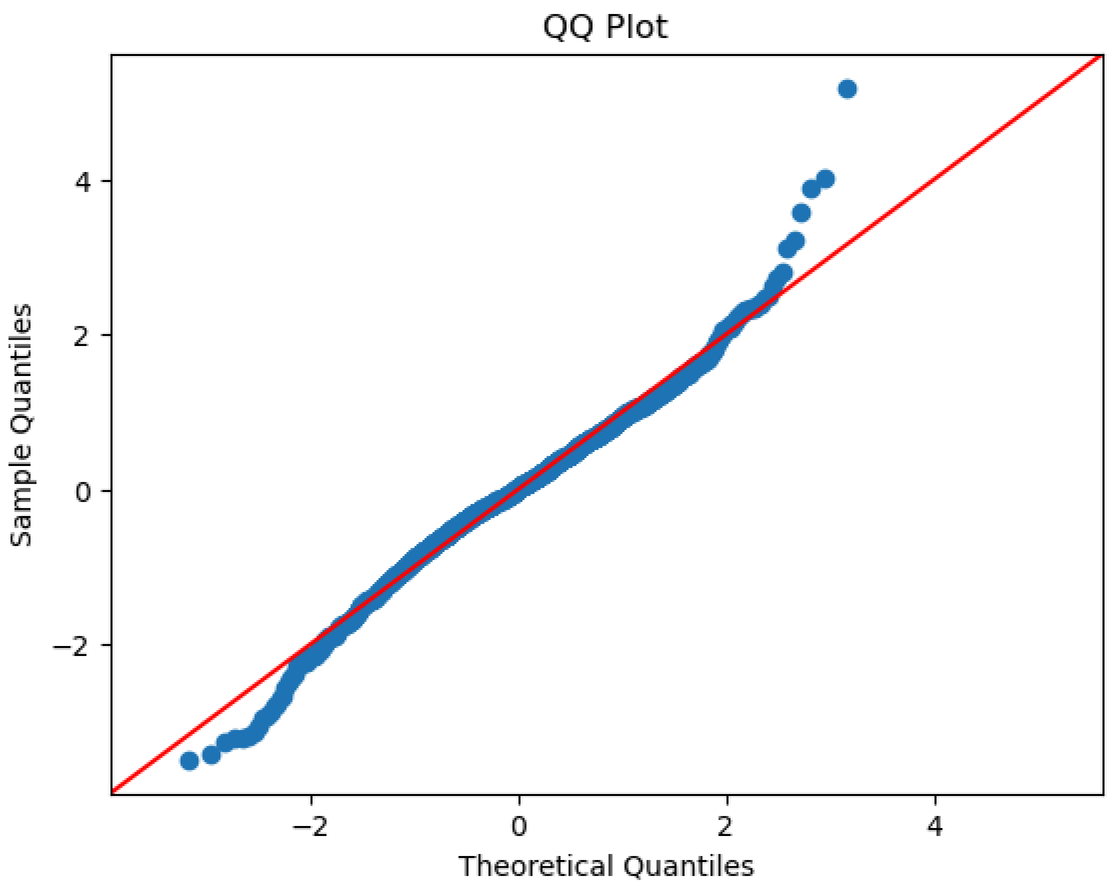
Out[18]:

	Features	VIF
1	AAPL(t-1)	7.634911
2	^GSPC(t-1)	7.634911

In [19]:
```
# 4) Assumption: Normality of Residual => 1) Visual Test (Histogram) or

plt.figure(figsize = (6,4))
plt.hist(residual, bins = 30)
plt.title("Normality of Residuals")
plt.show()
```



In [20]:
```
# QQ Plot for testing Normality of Residuals
import statsmodels.api as sm
sm.qqplot(residual, line = '45', fit = True)
plt.title('QQ Plot')
plt.show()
```



In [21]:
```
# Test 5: Auto correlation of Residual:  Durbin Watson Test

from statsmodels.stats.stattools import durbin_watson
dw = durbin_watson(residual)
dw # p value

# p value < 0.05 => Autocorrelation b/w residual is there
# p value > 0.05 => Autocorrelation b/w residual is not there

# Our 5th condition is met
```

Out[21]: 1.0410072978607603

In [22]:
```
# All the 5 conditions of Linear Regression Modela are MET
# Linearity
# Homoscedasticity
# VIF
# Normality of Residuals
# Auto correlation of Residual
```

In [23]:
```
# Predict the Stock Price for the Year 2025
```

In [24]:
```
# Step 1: Download the data from Yahoo Finance
tickers = ['AAPL', '^GSPC']
df = yf.download(tickers, start = '2025-01-01', end = '2025-03-31')['Clo
df.head()
```

```
[*********************100%***********************]  2 of 2 completed
```

Out[24]:

Ticker	AAPL	^GSPC
Date		
2025-01-02	243.582199	5868.549805
2025-01-03	243.092728	5942.470215
2025-01-06	244.730927	5975.379883
2025-01-07	241.944000	5909.029785
2025-01-08	242.433441	5918.250000

In [25]:
```
# Step 2: Data Eng
df['AAPL(t-1)'] = df['AAPL'].shift(1)
df['^GSPC(t-1)'] = df['^GSPC'].shift(1)
df = df.dropna()
df.head()
```

Out[25]:

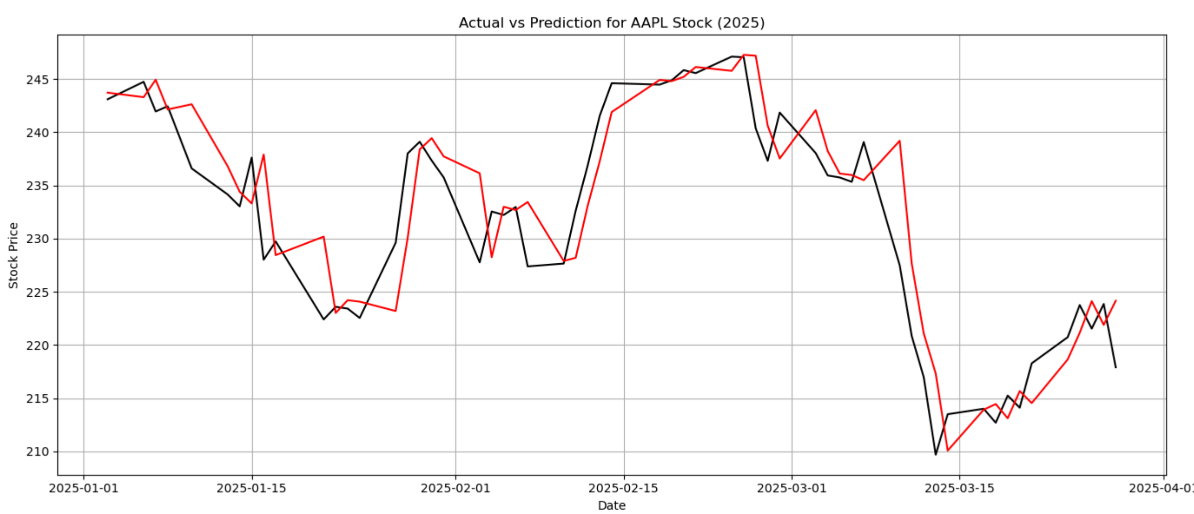
Ticker	AAPL	^GSPC	AAPL(t-1)	^GSPC(t-1)
Date				
2025-01-03	243.092728	5942.470215	243.582199	5868.549805
2025-01-06	244.730927	5975.379883	243.092728	5942.470215
2025-01-07	241.944000	5909.029785	244.730927	5975.379883
2025-01-08	242.433441	5918.250000	241.944000	5909.029785
2025-01-10	236.589874	5827.040039	242.433441	5918.250000

In [26]:
```
X_test = df[['AAPL(t-1)', '^GSPC(t-1)']]
X_test = sm.add_constant(X_test)

df_result = pd.DataFrame()
df_result['Actual'] = df['AAPL']
df_result['Predicted'] = model.predict(X_test)
df_result.head()
```

Out[26]:

	Actual	Predicted
Date		
2025-01-03	243.092728	243.710053
2025-01-06	244.730927	243.288152
2025-01-07	241.944000	244.926750
2025-01-08	242.433441	242.130768
2025-01-10	236.589874	242.619834

In [27]:
```
# Plot between Actual vs Predicted Value

import matplotlib.pyplot as plt

plt.figure(figsize = (14,6))
plt.plot(df_result.index, df_result['Actual'], label = 'Actual', color =
plt.plot(df_result.index, df_result['Predicted'], label = 'Predicted', c
plt.title("Actual vs Prediction for AAPL Stock (2025)")
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [28]:
```
# Risk Metrics
# Calculate rmse and mse
# rmse = root mean square error => Sq root(Avg((A - P)^2))
# mse = mean square error => Avg((A - P)^2)

from sklearn.metrics import mean_squared_error
import numpy as np

# Calculate mse
mse = mean_squared_error(df_result['Actual'], df_result['Predicted'])
rmse = np.sqrt(mse)
print(rmse, mse)
```

```
4.165605166643048 17.35226640436326
```

In [29]:
```
# Conclusion: It's a decent Model but not 100% Accurate
# Lesson: Stock data in general have lot of non linearities
# It's extremely tough to use simple linear regression model just to cap
# That's why in the industry it's common to use ML Models which are grea
```

Step 1: Download the data

In [30]:
```
# Step 1: Download the data from Yahoo Finance
tickers = ['AAPL', 'AMZN', 'MSFT', 'QQQ', '^GSPC']
df = yf.download(tickers, start = '2020-01-01', end = '2025-04-01')['Clo
df
```

```
[*********************100%***********************]  5 of 5 completed
```

Out[30]:

Ticker	AAPL	AMZN	MSFT	QQQ	^GSPC
Date					
2020-01-02	72.716072	94.900497	153.323257	209.325882	3257.850098
2020-01-03	72.009125	93.748497	151.414139	207.408463	3234.850098
2020-01-06	72.582901	95.143997	151.805511	208.744888	3246.280029
2020-01-07	72.241531	95.343002	150.421402	208.715836	3237.179932
2020-01-08	73.403641	94.598503	152.817352	210.284592	3253.050049
...	...	...	...	...	...
2025-03-25	223.750000	205.710007	395.160004	493.459991	5776.649902
2025-03-26	221.529999	201.130005	389.970001	484.380005	5712.200195
2025-03-27	223.850006	201.360001	390.579987	481.619995	5693.310059
2025-03-28	217.899994	192.720001	378.799988	468.940002	5580.939941
2025-03-31	222.130005	190.259995	375.390015	468.920013	5611.850098

1318 rows × 5 columns

Step 2: Feature Engineering

In [31]:
```
# Step 2: Perform Feature Engineering

# Lesson: To predict AAPL Stock price, we have to consider yesterday's p
# The market is not open yet so we don't know what's the price today

# Considering Yesterday's Value
df['AAPL(t-1)'] = df['AAPL'].shift(1)
df['AMZN(t-1)'] = df['AMZN'].shift(1)
df['MSFT(t-1)'] = df['MSFT'].shift(1)
df['QQQ(t-1)'] = df['QQQ'].shift(1)
df['^GSPC(t-1)'] = df['^GSPC'].shift(1)

# Moving Avg (MA): Technical Indicator - It helps you understand the sho
df['AAPL_MA_5'] = df['AAPL'].rolling(window=5).mean()
df['AMZN_MA_5'] = df['AMZN'].rolling(window=5).mean()
df['MSFT_MA_5'] = df['MSFT'].rolling(window=5).mean()
df['QQQ_MA_5'] = df['QQQ'].rolling(window=5).mean()
df['^GSPC_MA_5'] = df['^GSPC'].rolling(window=5).mean()

# Set Y Variable - Next day
df['Target'] = df['AAPL'].shift(-1)

df = df.dropna()
df
```

Out[31]:

Ticker	AAPL	AMZN	MSFT	QQQ	^GSPC	AAPL(t-1)	AMZN(t-1)	MS
Date								
2020-01-08	73.403641	94.598503	152.817352	210.284592	3253.050049	72.241531	95.343002	150
2020-01-09	74.962807	95.052498	154.726517	212.066467	3274.699951	73.403641	94.598503	152
2020-01-10	75.132263	94.157997	154.010559	211.524124	3265.350098	74.962807	95.052498	154
2020-01-13	76.737419	94.565002	155.862411	213.964478	3288.129883	75.132263	94.157997	154
2020-01-14	75.701210	93.472000	154.764664	213.121902	3283.149902	76.737419	94.565002	155
...	...	...	...	...	...	...	...	
2025-03-24	220.729996	203.259995	393.079987	490.660004	5767.569824	218.270004	196.210007	391
2025-03-25	223.750000	205.710007	395.160004	493.459991	5776.649902	220.729996	203.259995	393
2025-03-26	221.529999	201.130005	389.970001	484.380005	5712.200195	223.750000	205.710007	395
2025-03-27	223.850006	201.360001	390.579987	481.619995	5693.310059	221.529999	201.130005	389
2025-03-28	217.899994	192.720001	378.799988	468.940002	5580.939941	223.850006	201.360001	390

1313 rows × 16 columns

Step 3: Lasso Regression

In [32]:

```
# Step 1: Import all the required libraries
# Step 2: Define Features and Target Variables
# Step 3: Train Test Split
# Step 4: Apply Lasso Regression
# Step 5: Get Intercept and Coeff for Lasso Regression
# Step 6: Predict using Lasso Regression
# Step 7: Create a dataframe with Actual and Predicted Values
# Step 8: Plot Actual & Predicted Values
# Step 9: Evaluate the Model - R square, mse, rmse
```

In [33]:
```
# Step 1: Import all the required libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import Lasso
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

In [34]:
```
# Step 2: Define Features and Target Variables
X = df[['AAPL(t-1)', 'AMZN(t-1)',
       'MSFT(t-1)', 'QQQ(t-1)', '^GSPC(t-1)', 'AAPL_MA_5', 'AMZN_MA_5',
       'MSFT_MA_5', 'QQQ_MA_5', '^GSPC_MA_5']]

Y = df['Target']
```

In [35]:
```
# Step 3: Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.
```

In [36]:
```
# Step 4: Apply Lasso Regression
from sklearn.linear_model import Lasso

lasso = Lasso(alpha = 10)
lasso.fit(X_train, Y_train) # Train the model
```

Out[36]:
```
▾    Lasso
Lasso(alpha=10)
```

In [37]:
```
# Step 5: Get Intercept and Coeff for Lasso Regression

coefficients = lasso.coef_
coefficients

intercept = lasso.intercept_
intercept

coeff_df = pd.DataFrame({'Feature':X.columns, 'Coefficients':coefficient
coeff_df
```

Out[37]:

	Feature	Coefficients
0	AAPL(t-1)	0.581944
1	AMZN(t-1)	-0.000000
2	MSFT(t-1)	0.006772
3	QQQ(t-1)	0.000000
4	^GSPC(t-1)	0.002424
5	AAPL_MA_5	0.357012
6	AMZN_MA_5	-0.000000
7	MSFT_MA_5	0.000000
8	QQQ_MA_5	0.000000
9	^GSPC_MA_5	0.000000

In [38]:
```
# Step 6: Predict using Lasso Regression
y_pred = lasso.predict(X_test)
y_pred
```

Out[38]:
```
array([244.97062955, 244.14187203, 244.49286543, 242.12254896,
       241.94525568, 237.64656806, 235.39680802, 234.4421395 ,
       236.41167126, 230.29317533, 230.628106  , 225.81117937,
       225.6990322 , 225.29119603, 224.71082941, 229.66071838,
       235.87109987, 237.40235799, 237.20880604, 236.07879459,
       230.90535526, 233.31316351, 232.87306976, 232.78270648,
       229.34479861, 229.62205729, 232.84644281, 235.87506325,
       239.97955867, 242.95179599, 243.7952358 , 244.73776086,
       245.53375196, 245.24265519, 246.2272409 , 245.76111671,
       241.2790264 , 238.95558724, 241.20544064, 237.88342177,
       236.1587917 , 236.14789353, 235.4311948 , 236.90691876,
       228.6184177 , 223.31377031, 219.32108851, 213.02741802,
       214.63303211, 214.4365347 , 213.36832544, 215.34396182,
       214.98515768, 217.93365214, 220.4096761 , 222.65236517,
       221.86526039, 223.14729878])
```

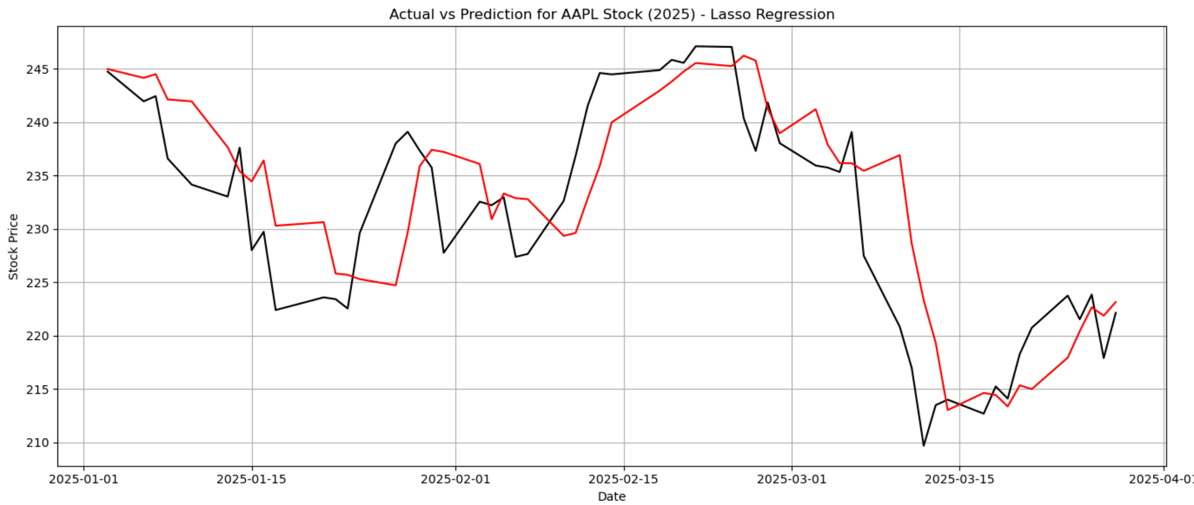
In [39]:
```
# Step 7: Create a dataframe with Actual and Predicted Values

df_result = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
df_result
```

Out[39]:

Date	Actual	Predicted
2025-01-03	244.730927	244.970630
2025-01-06	241.944000	244.141872
2025-01-07	242.433441	244.492865
2025-01-08	236.589874	242.122549
2025-01-10	234.142563	241.945256
2025-01-13	233.023788	237.646568
2025-01-14	237.608749	235.396808
2025-01-15	228.009308	234.442139
2025-01-16	229.727417	236.411671
2025-01-17	222.395477	230.293175
2025-01-21	223.584167	230.628106
2025-01-22	223.414368	225.811179
2025-01-23	222.535324	225.699032
2025-01-24	229.607544	225.291196
2025-01-27	237.998322	224.710829
2025-01-28	239.097122	229.660718
2025-01-29	237.329056	235.871100
2025-01-30	235.740814	237.402358
2025-01-31	227.759583	237.208806
2025-02-03	232.544327	236.078795
2025-02-04	232.214691	230.905355
2025-02-05	232.963867	233.313164
2025-02-06	227.380005	232.873070
2025-02-07	227.649994	232.782706
2025-02-10	232.619995	229.344799
2025-02-11	236.869995	229.622057
2025-02-12	241.529999	232.846443
2025-02-13	244.600006	235.875063
2025-02-14	244.470001	239.979559
2025-02-18	244.869995	242.951796
2025-02-19	245.830002	243.795236
2025-02-20	245.550003	244.737761
2025-02-21	247.100006	245.533752

Date	Actual	Predicted
2025-02-24	247.039993	245.242655
2025-02-25	240.360001	246.227241
2025-02-26	237.300003	245.761117
2025-02-27	241.839996	241.279026
2025-02-28	238.029999	238.955587
2025-03-03	235.929993	241.205441
2025-03-04	235.740005	237.883422
2025-03-05	235.330002	236.158792
2025-03-06	239.070007	236.147894
2025-03-07	227.479996	235.431195
2025-03-10	220.839996	236.906919
2025-03-11	216.979996	228.618418
2025-03-12	209.679993	223.313770
2025-03-13	213.490005	219.321089
2025-03-14	214.000000	213.027418
2025-03-17	212.690002	214.633032
2025-03-18	215.240005	214.436535
2025-03-19	214.100006	213.368325
2025-03-20	218.270004	215.343962
2025-03-21	220.729996	214.985158
2025-03-24	223.750000	217.933652
2025-03-25	221.529999	220.409676
2025-03-26	223.850006	222.652365
2025-03-27	217.899994	221.865260
2025-03-28	222.130005	223.147299

In [40]:
```
# Step 8: Plot Actual & Predicted Values
import matplotlib.pyplot as plt

plt.figure(figsize = (14,6))
plt.plot(df_result.index, df_result['Actual'], label = 'Actual', color =
plt.plot(df_result.index, df_result['Predicted'], label = 'Predicted', c
plt.title("Actual vs Prediction for AAPL Stock (2025) - Lasso Regression
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [41]:
```
# Step 9: Evaluate the Model - R square, mse, rmse

from sklearn.metrics import r2_score, mean_squared_error

r2 = r2_score(Y_test, y_pred)
print("R square", r2)

mse = mean_squared_error(Y_test, y_pred)
print("mse",mse)

rmse = np.sqrt(mse)
print("rmse", rmse)
```

```
R square 0.6745742147077414
mse 33.88523686502021
rmse 5.8211027189889215
```

Step 3: Ridge Regression

In [42]:
```
# Step 1: Import all the required libraries
# Step 2: Define Features and Target Variables
# Step 3: Train Test Split
# Step 4: Apply Ridge Regression
# Step 5: Get Intercept and Coeff for Ridge Regression
# Step 6: Predict using Ridge Regression
# Step 7: Create a dataframe with Actual and Predicted Values
# Step 8: Plot Actual & Predicted Values
# Step 9: Evaluate the Model - R square, mse, rmse
```

In [43]:
```
# Step 1: Import all the required libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import Ridge
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

In [44]:
```
# Step 2: Define Features and Target Variables
X = df[['AAPL(t-1)', 'AMZN(t-1)',
       'MSFT(t-1)', 'QQQ(t-1)', '^GSPC(t-1)', 'AAPL_MA_5', 'AMZN_MA_5',
       'MSFT_MA_5', 'QQQ_MA_5', '^GSPC_MA_5']]

Y = df['Target']
```

In [45]:
```
# Step 3: Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.
```

In [46]:
```
# Step 4: Apply Ridge Regression
from sklearn.linear_model import Ridge

ridge = Ridge(alpha = 10)
ridge.fit(X_train, Y_train) # Train the model
```

Out[46]:
```
▾    Ridge
Ridge(alpha=10)
```

In [47]:
```
# Step 5: Get Intercept and Coeff for Lasso Regression

coefficients = ridge.coef_
coefficients

intercept = ridge.intercept_
intercept

coeff_df = pd.DataFrame({'Feature':X.columns, 'Coefficients':coefficient
coeff_df
```

Out[47]:

	Feature	Coefficients
0	AAPL(t-1)	0.467920
1	AMZN(t-1)	0.062511
2	MSFT(t-1)	-0.014934
3	QQQ(t-1)	0.014701
4	^GSPC(t-1)	0.005419
5	AAPL_MA_5	0.513144
6	AMZN_MA_5	-0.058280
7	MSFT_MA_5	0.028290
8	QQQ_MA_5	-0.016678
9	^GSPC_MA_5	-0.005723

In [48]:
```
# Step 6: Predict using Ridge Regression
y_pred = ridge.predict(X_test)
y_pred
```

Out[48]:
```
array([245.78092384, 245.09454324, 245.22520298, 242.30061532,
       241.90655861, 237.72200134, 235.65987475, 234.65550246,
       236.2645113 , 230.53756046, 230.46140677, 226.30244673,
       225.43481612, 224.70533033, 224.07465189, 228.46169577,
       234.41769762, 236.11306963, 236.99902675, 235.92847769,
       231.15221923, 233.08234091, 232.18086518, 232.04598581,
       228.43509056, 229.15028688, 232.00060376, 234.59148493,
       238.90626501, 241.98055619, 243.09618537, 244.21685144,
       244.93516106, 244.43277803, 245.37562859, 245.09468031,
       241.63303234, 238.96887684, 241.16732896, 237.50065356,
       235.80470475, 236.35699835, 235.0933765 , 236.37303158,
       228.5334362 , 223.75011871, 219.93018804, 213.27525032,
       214.51622345, 214.02268134, 212.6421225 , 214.65164795,
       214.4942894 , 217.00006402, 220.06132319, 222.22316842,
       221.39486881, 222.41142609])
```

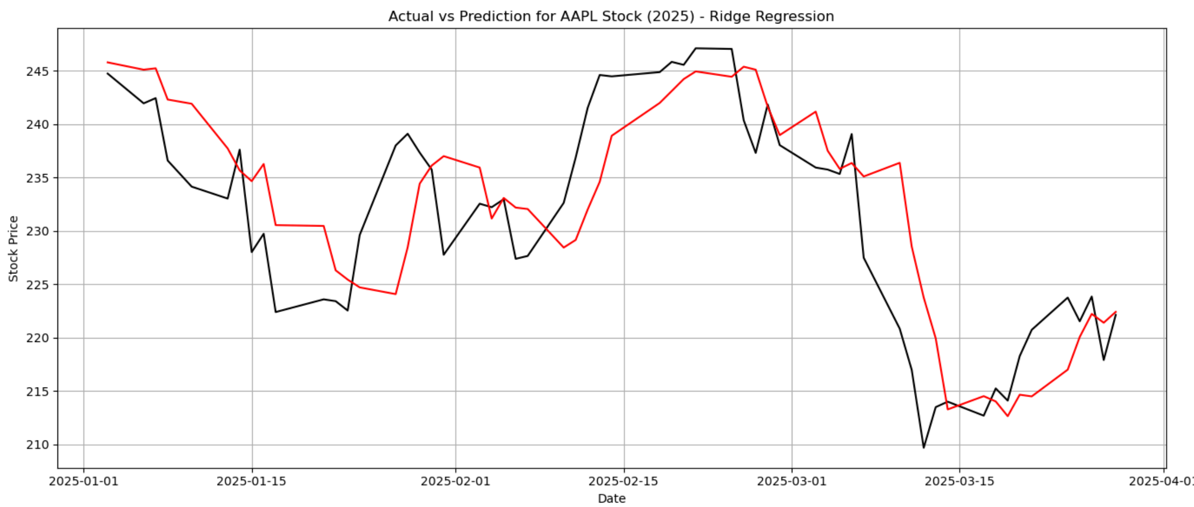
In [49]:
```
# Step 7: Create a dataframe with Actual and Predicted Values

df_result = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
df_result
```

Out[49]:

Date	Actual	Predicted
2025-01-03	244.730927	245.780924
2025-01-06	241.944000	245.094543
2025-01-07	242.433441	245.225203
2025-01-08	236.589874	242.300615
2025-01-10	234.142563	241.906559
2025-01-13	233.023788	237.722001
2025-01-14	237.608749	235.659875
2025-01-15	228.009308	234.655502
2025-01-16	229.727417	236.264511
2025-01-17	222.395477	230.537560
2025-01-21	223.584167	230.461407
2025-01-22	223.414368	226.302447
2025-01-23	222.535324	225.434816
2025-01-24	229.607544	224.705330
2025-01-27	237.998322	224.074652
2025-01-28	239.097122	228.461696
2025-01-29	237.329056	234.417698
2025-01-30	235.740814	236.113070
2025-01-31	227.759583	236.999027
2025-02-03	232.544327	235.928478
2025-02-04	232.214691	231.152219
2025-02-05	232.963867	233.082341
2025-02-06	227.380005	232.180865
2025-02-07	227.649994	232.045986
2025-02-10	232.619995	228.435091
2025-02-11	236.869995	229.150287
2025-02-12	241.529999	232.000604
2025-02-13	244.600006	234.591485
2025-02-14	244.470001	238.906265
2025-02-18	244.869995	241.980556
2025-02-19	245.830002	243.096185
2025-02-20	245.550003	244.216851
2025-02-21	247.100006	244.935161

Date	Actual	Predicted
2025-02-24	247.039993	244.432778
2025-02-25	240.360001	245.375629
2025-02-26	237.300003	245.094680
2025-02-27	241.839996	241.633032
2025-02-28	238.029999	238.968877
2025-03-03	235.929993	241.167329
2025-03-04	235.740005	237.500654
2025-03-05	235.330002	235.804705
2025-03-06	239.070007	236.356998
2025-03-07	227.479996	235.093377
2025-03-10	220.839996	236.373032
2025-03-11	216.979996	228.533436
2025-03-12	209.679993	223.750119
2025-03-13	213.490005	219.930188
2025-03-14	214.000000	213.275250
2025-03-17	212.690002	214.516223
2025-03-18	215.240005	214.022681
2025-03-19	214.100006	212.642123
2025-03-20	218.270004	214.651648
2025-03-21	220.729996	214.494289
2025-03-24	223.750000	217.000064
2025-03-25	221.529999	220.061323
2025-03-26	223.850006	222.223168
2025-03-27	217.899994	221.394869
2025-03-28	222.130005	222.411426

In [50]:
```
# Step 8: Plot Actual & Predicted Values
import matplotlib.pyplot as plt

plt.figure(figsize = (14,6))
plt.plot(df_result.index, df_result['Actual'], label = 'Actual', color =
plt.plot(df_result.index, df_result['Predicted'], label = 'Predicted', c
plt.title("Actual vs Prediction for AAPL Stock (2025) - Ridge Regression
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [51]:
```
# Step 9: Evaluate the Model - R square, mse, rmse

from sklearn.metrics import r2_score, mean_squared_error

r2 = r2_score(Y_test, y_pred)
print("R square", r2)

mse = mean_squared_error(Y_test, y_pred)
print("mse",mse)

rmse = np.sqrt(mse)
print("rmse", rmse)
```

```
R square 0.6552348970022761
mse 35.89895978088886
rmse 5.991574065376215
```

In [52]:
```
# Elastic Net - Lasso + Ridge
```

Elastic Net Regression

In [53]:
```
# Step 1: Import all the required libraries
# Step 2: Define Features and Target Variables
# Step 3: Train Test Split
# Step 4: Apply Elastic Net Regression
# Step 5: Get Intercept and Coeff for Elastic Net Regression
# Step 6: Predict using Elastic Net Regression
# Step 7: Create a dataframe with Actual and Predicted Values
# Step 8: Plot Actual & Predicted Values
# Step 9: Evaluate the Model - R square, mse, rmse
```

In [54]:
```
# Step 1: Import all the required libraries
import numpy as np
import pandas as pd
from sklearn.linear_model import ElasticNet
from sklearn.model_selection import train_test_split
from sklearn.metrics import r2_score, mean_squared_error
```

In [55]:
```
# Step 2: Define Features and Target Variables
X = df[['AAPL(t-1)', 'AMZN(t-1)',
       'MSFT(t-1)', 'QQQ(t-1)', '^GSPC(t-1)', 'AAPL_MA_5', 'AMZN_MA_5',
       'MSFT_MA_5', 'QQQ_MA_5', '^GSPC_MA_5']]

Y = df['Target']
```

In [56]:
```
# Step 3: Train Test Split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.
```

In [57]:
```
# Step 4: Apply Elastic Net Regression
from sklearn.linear_model import ElasticNet

elastic_net = ElasticNet(alpha = 1, l1_ratio = 0.5)
elastic_net.fit(X_train, Y_train) #Train the model

# alpha = 1, alpha control the strength of regularization (higher alpha
# l1_ratio = 0.5 => applying 50% lasso and 50% as ridge regression - alp
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/linear_model/_coord
inate_descent.py:631: ConvergenceWarning: Objective did not converge. Y
ou might want to increase the number of iterations, check the scale of
the features or consider increasing regularisation. Duality gap: 6.651e
+03, tolerance: 2.210e+02
  model = cd_fast.enet_coordinate_descent(
```

Out[57]:
```
▾ ElasticNet
ElasticNet(alpha=1)
```

In [58]:
```
# Step 5: Get Intercept and Coeff for Elastic Net Regression

coefficients = elastic_net.coef_
coefficients

intercept = elastic_net.intercept_
intercept

coeff_df = pd.DataFrame({'Feature':X.columns, 'Coefficients':coefficient
coeff_df
```

Out[58]:

	Feature	Coefficients
0	AAPL(t-1)	0.540711
1	AMZN(t-1)	0.000184
2	MSFT(t-1)	0.012382
3	QQQ(t-1)	0.001336
4	^GSPC(t-1)	0.006426
5	AAPL_MA_5	0.434639
6	AMZN_MA_5	0.000000
7	MSFT_MA_5	0.000000
8	QQQ_MA_5	0.000000
9	^GSPC_MA_5	-0.006563

In [59]:
```
# Step 6: Predict using Elastic Net Regression
y_pred = elastic_net.predict(X_test)
y_pred
```

Out[59]:
```
array([245.24396357, 244.59964945, 244.89092635, 242.15797064,
       241.95571833, 237.49063938, 235.35854924, 234.34653895,
       236.3824075 , 230.26519883, 230.34216011, 225.57386174,
       225.22821502, 224.6611276 , 224.00575885, 228.45231449,
       234.92258163, 236.54455909, 236.68557633, 235.49290764,
       230.39026177, 232.65563861, 232.24679213, 232.12156123,
       228.54269742, 228.95471801, 232.061062  , 234.92619326,
       239.26090045, 242.27392112, 243.28147121, 244.34780228,
       245.2023884 , 244.65580724, 245.69300704, 245.25122179,
       241.26356601, 238.64696241, 241.15226305, 237.56407733,
       235.71518696, 236.18023474, 235.2196472 , 236.79729831,
       228.29988761, 223.12761119, 219.31304829, 212.74660596,
       214.44710659, 214.19648411, 212.77294772, 214.80226349,
       214.46969326, 217.27520537, 220.03475214, 222.25432696,
       221.37013928, 222.58864462])
```

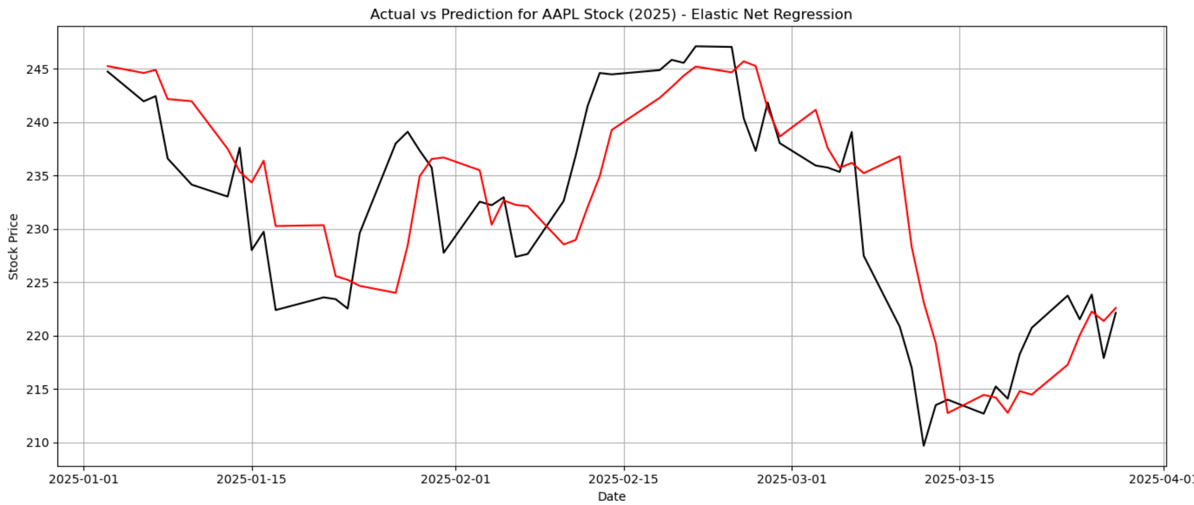
In [60]:
```
# Step 7: Create a dataframe with Actual and Predicted Values
df_result = pd.DataFrame({'Actual': Y_test, 'Predicted': y_pred})
df_result
```

Out[60]:

Date	Actual	Predicted
2025-01-03	244.730927	245.243964
2025-01-06	241.944000	244.599649
2025-01-07	242.433441	244.890926
2025-01-08	236.589874	242.157971
2025-01-10	234.142563	241.955718
2025-01-13	233.023788	237.490639
2025-01-14	237.608749	235.358549
2025-01-15	228.009308	234.346539
2025-01-16	229.727417	236.382408
2025-01-17	222.395477	230.265199
2025-01-21	223.584167	230.342160
2025-01-22	223.414368	225.573862
2025-01-23	222.535324	225.228215
2025-01-24	229.607544	224.661128
2025-01-27	237.998322	224.005759
2025-01-28	239.097122	228.452314
2025-01-29	237.329056	234.922582
2025-01-30	235.740814	236.544559
2025-01-31	227.759583	236.685576
2025-02-03	232.544327	235.492908
2025-02-04	232.214691	230.390262
2025-02-05	232.963867	232.655639
2025-02-06	227.380005	232.246792
2025-02-07	227.649994	232.121561
2025-02-10	232.619995	228.542697
2025-02-11	236.869995	228.954718
2025-02-12	241.529999	232.061062
2025-02-13	244.600006	234.926193
2025-02-14	244.470001	239.260900
2025-02-18	244.869995	242.273921
2025-02-19	245.830002	243.281471
2025-02-20	245.550003	244.347802
2025-02-21	247.100006	245.202388

Date	Actual	Predicted
2025-02-24	247.039993	244.655807
2025-02-25	240.360001	245.693007
2025-02-26	237.300003	245.251222
2025-02-27	241.839996	241.263566
2025-02-28	238.029999	238.646962
2025-03-03	235.929993	241.152263
2025-03-04	235.740005	237.564077
2025-03-05	235.330002	235.715187
2025-03-06	239.070007	236.180235
2025-03-07	227.479996	235.219647
2025-03-10	220.839996	236.797298
2025-03-11	216.979996	228.299888
2025-03-12	209.679993	223.127611
2025-03-13	213.490005	219.313048
2025-03-14	214.000000	212.746606
2025-03-17	212.690002	214.447107
2025-03-18	215.240005	214.196484
2025-03-19	214.100006	212.772948
2025-03-20	218.270004	214.802263
2025-03-21	220.729996	214.469693
2025-03-24	223.750000	217.275205
2025-03-25	221.529999	220.034752
2025-03-26	223.850006	222.254327
2025-03-27	217.899994	221.370139
2025-03-28	222.130005	222.588645

In [61]:
```
# Step 8: Plot Actual & Predicted Values
import matplotlib.pyplot as plt

plt.figure(figsize = (14,6))
plt.plot(df_result.index, df_result['Actual'], label = 'Actual', color =
plt.plot(df_result.index, df_result['Predicted'], label = 'Predicted', c
plt.title("Actual vs Prediction for AAPL Stock (2025) - Elastic Net Regr
plt.xlabel("Date")
plt.ylabel("Stock Price")
plt.grid(True)
plt.tight_layout()
plt.show()
```



In [62]:
```
# Step 9: Evaluate the Model - R square, mse, rmse
from sklearn.metrics import r2_score, mean_squared_error

r2 = r2_score(Y_test, y_pred)
print("R square", r2)

mse = mean_squared_error(Y_test, y_pred)
print("mse",mse)

rmse = np.sqrt(mse)
print("rmse", rmse)
```

```
R square 0.6638572335134292
mse 35.00115165315263
rmse 5.916177114755155
```

In [63]:
```
# Performance for All our Models
```

In [64]:
```
# OLS
# R-squared:0.993
# mse 17.35226637043703
# rmse 4.165605162570864
```

In [65]:
```
# Lasso Regression
# R square 0.6745742009648297
# mse 33.88523829601259
# rmse 5.82110284190312
```

In [66]:
```
# Ridge Regression
# R square 0.6552348986780858
# mse 35.898959606393746
# rmse 5.9915740508145054
```

In [67]:
```
# Elastic Net Regression
# R square 0.6638572193358945
# mse 35.0011531294005
# rmse 5.9161772395188175
```

In [68]:
```
## Thank You!
```