In [1]:
```python
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

a) Join input files performance.csv and info.txt, and produce a data frame with Date, Index1, Index2, Index3 and Stock A. The values are monthly returns.

In [2]:
```python
# Extract the data from performance 2023 and info 2023
performance = pd.read_csv('performance 2023.csv')
info = pd.read_csv('info 2023.txt',sep='\s+')

# Merge two DataFrames
merge_data = pd.merge(performance, info, on='ID')

# Pivot the Merged Data
merge_data_pivot = merge_data.pivot(index = 'Date', columns='Name', values = 'Performance')

# Rename the pivot data columns
merge_data_pivot.rename(columns={'Index 1': 'Index 1','Index 2': 'Index 2', 'Index 3': 'Index 3', 'Stock A': 'Stock A'}, inplace=True)

merge_data_pivot.head()
```

Out[2]:

Name	Index 1	Index 2	Index 3	Stock A
Date				
01/01/1990	-0.068817	-0.011883	-0.019140	-0.091892
01/01/1991	0.041518	0.012362	0.010910	0.229506
01/01/1992	-0.019900	-0.013604	0.049300	0.083338
01/01/1993	0.007046	0.019176	-0.000886	0.046225
01/01/1994	0.032501	0.013502	0.046439	0.035714

b). Split the dataset into training and testing per 80/20 split. Use the training dataset, regress the Stock A returns against all 3 indices together, and interpret the results. Please set seed to make reproducible outputs.

In [3]:
```python
# Set seeds
np.random.seed(100)

# Set the X and Y trains
X = merge_data_pivot[['Index 1','Index 2','Index 3']]
y = merge_data_pivot['Stock A']

# Set the train group and the test group
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=100)

# Fit a linear regression model
lr = LinearRegression()
lr.fit(X_train, y_train)

coefficients = pd.DataFrame(lr.coef_, X.columns, columns=['Coefficient'])
print(coefficients)
```

```
         Coefficient
Name
Index 1     1.781004
Index 2    -0.330732
Index 3    -0.286995
```

In [4]:
```python
# Add an OLS regression
# Add a constant (intercept) to the independent variables
X_train_with_constant = sm.add_constant(X_train)

# Fit the linear regression model using statsmodels' OLS
ols_model = sm.OLS(y_train, X_train_with_constant)
ols_results = ols_model.fit()

# Prepare the predict data
y_train_predict = lr.predict(X_train)
y_test_predict = lr.predict(X_test)

# Print the summary
print(ols_results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                Stock A   R-squared:                       0.334
Model:                            OLS   Adj. R-squared:                  0.327
Method:                 Least Squares   F-statistic:                     51.76
Date:                Mon, 14 Oct 2024   Prob (F-statistic):           3.78e-27
Time:                        18:02:47   Log-Likelihood:                 316.37
No. Observations:                 314   AIC:                            -624.7
Df Residuals:                     310   BIC:                            -609.7
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0011      0.005      0.213      0.832      -0.009       0.012
Index 1        1.7810      0.214      8.312      0.000       1.359       2.203
Index 2       -0.3307      0.478     -0.693      0.489      -1.270       0.609
Index 3       -0.2870      0.205     -1.401      0.162      -0.690       0.116
==============================================================================
Omnibus:                       94.803   Durbin-Watson:                   1.860
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              916.303
Skew:                           0.929   Prob(JB):                    1.06e-199
Kurtosis:                      11.160   Cond. No.                         96.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

c). What might be the potential problems with this regression, and how to identify the problems?

1. The data has three independent variables, which is possible to occur multicollinearity. To identify the problem, we need to run the variance inflation factor test to see if the independent variables greater than 5. If VIFs are greater than 5, it suggests that the multicollinearity is a problem.
2. R-square is 0.334, which means 33.4% of the variation in the dependent variable, and the model has a low fit capacity. To identify the problem, it's necessary to add more variables to the model.
3. The p-value for index 2 and index 3 are higher than 0.05, indicating these two variables are not statistically significant at the 5% significance level. To identify the problem, either remove these two variables or add other new variables.
4. The J-B value is quite high with a near 0 probability, which means the data the resideuals are not normally distributed. To identify the problem, we need to use Q-Q Plot to check the normality of the residuals.

d) Use Lasso model to re-run the regression. Interpret the new coefficients, and explain why it's different from the results from linear regression.

In [5]:
```python
from sklearn.linear_model import Lasso
from sklearn.preprocessing import StandardScaler
```

In [6]:
```python
# Set seeds
np.random.seed(100)

# Set the train groups and test groups, and split the data
X = merge_data_pivot[['Index 1','Index 2','Index 3']]
y = merge_data_pivot['Stock A']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 100)

# Set the alpha for the lasso regression
lasso = Lasso(alpha=0.00001, random_state=100)

# Fit the lasso regression
lasso.fit(X_train, y_train)

# Get the coefficients
lasso_coefficients = pd.DataFrame(lasso.coef_, X_train.columns, columns=['Lasso Coefficient'])

# Predict on both training and test sets
y_train_predicted = lasso.predict(X_train)
y_test_predicted = lasso.predict(X_test)

# Evaluate the model
print(lasso_coefficients)
```

```
         Lasso Coefficient
Name
Index 1           1.739653
Index 2          -0.226095
Index 3          -0.250815
```

Difference:

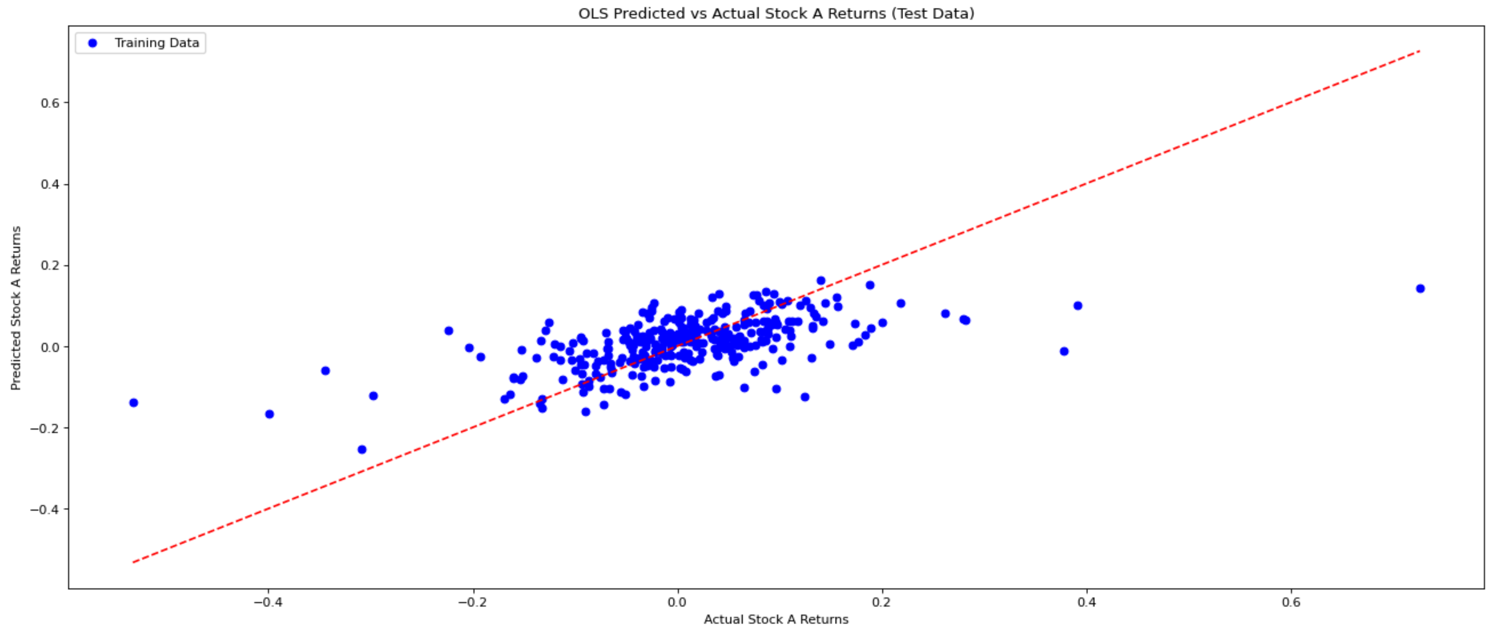
The lasso regression has the penalty parameters, which will squeeze the independent variables to filter the most related one. Therefore, the coefficient of Index 1 will be smaller than the linear regression result, and Index 2, 3 results will more close to 0. As the alpha parameter greater, the Index 2 and index 3 will become closer to 0, as they are not highly correlated to the Stock A, and the Index 1 will tend to 0, as the penalty become greater. The Lasso regression introduces bias to the model, but this can improve the model's performance on unseen data by reducing variance.

e). Use the fitted models from b) and d), predict Stock A returns in testing dataset, and plot the fitted values against the actual values. Which model provides a better fit, and what metric do you use to measure the fitness?

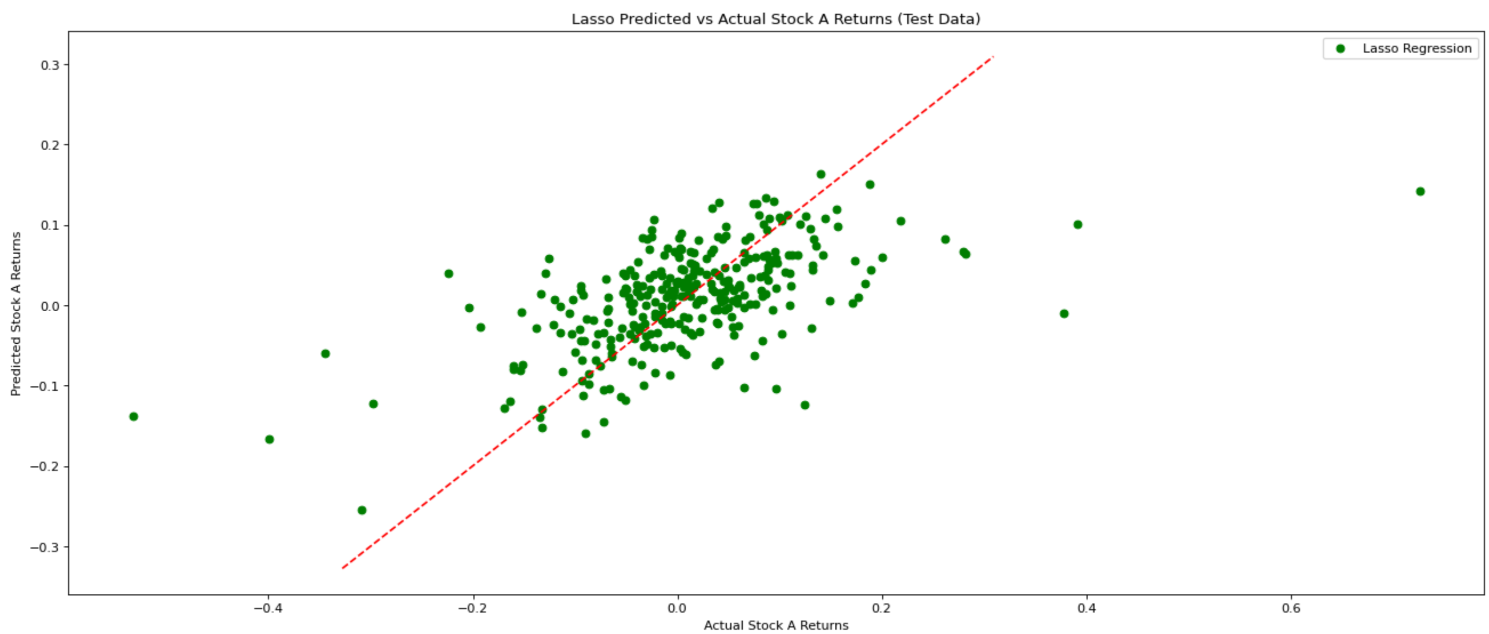
In [7]:
```python
from sklearn.metrics import mean_squared_error, r2_score
```

In [8]:
```python
# Plot Test vs Actual values
plt.figure(figsize=(20, 8), dpi=80)
plt.scatter(y_train, y_train_predict, color="blue", label="Training Data")
```

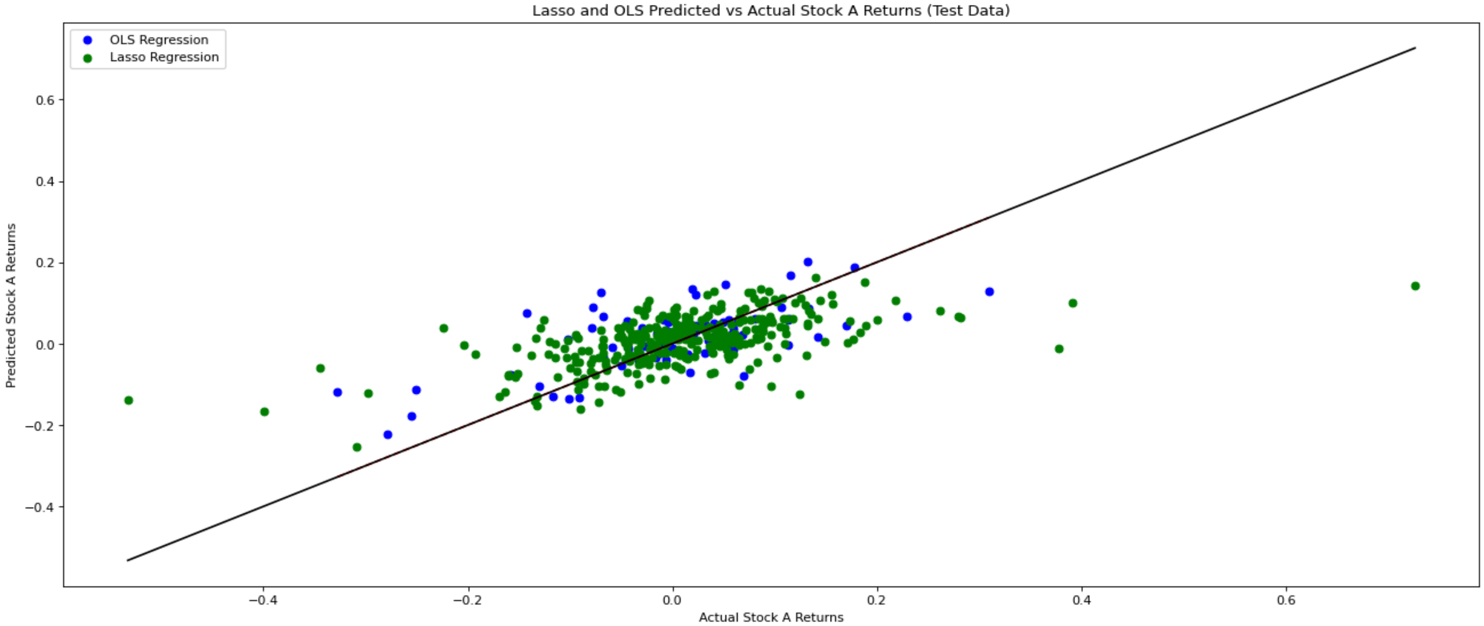
```python
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], color='red', linestyle='--')
plt.xlabel("Actual Stock A Returns")
plt.ylabel("Predicted Stock A Returns")
plt.title("OLS Predicted vs Actual Stock A Returns (Test Data)")
plt.legend()
plt.show()
```



In [9]:
```python
# Plot Test vs Actual values
plt.figure(figsize=(20, 8), dpi=80)
plt.scatter(y_train, y_train_predict, color="Green", label="Lasso Regression")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.xlabel("Actual Stock A Returns")
plt.ylabel("Predicted Stock A Returns")
plt.title("Lasso Predicted vs Actual Stock A Returns (Test Data)")
plt.legend()
plt.show()
```



In [10]:
```python
# Plot Predicted vs Actual values (Test Data)
plt.figure(figsize=(20, 8), dpi=80)
plt.scatter(y_test, y_test_predicted, color="blue", label="OLS Regression")
plt.scatter(y_train, y_train_predict, color="Green", label="Lasso Regression")
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], color='red', linestyle='--')
plt.plot([min(y_train), max(y_train)], [min(y_train), max(y_train)], color='Black', linestyle='-')
plt.xlabel("Actual Stock A Returns")
plt.ylabel("Predicted Stock A Returns")
plt.title("Lasso and OLS Predicted vs Actual Stock A Returns (Test Data)")
plt.legend()
plt.show()
```



In [11]:

```python
# Calculate MSE for both models
mse_ols = mean_squared_error(y_train, y_train_predict)
mse_lasso = mean_squared_error(y_test, y_test_predicted)

# Calculate R-squared for both models
r2_ols = r2_score(y_train, y_train_predict)
r2_lasso = r2_score(y_test, y_test_predicted)

print(f"OLS Model - MSE: {mse_ols}, R-squared: {r2_ols}")
print(f"Lasso Model - MSE: {mse_lasso}, R-squared: {r2_lasso}")
```

```
OLS Model - MSE: 0.007805346129705988, R-squared: 0.33374624456221824
Lasso Model - MSE: 0.006143705732064331, R-squared: 0.43076957854985276
```

Conclusion:

1. The lasso model has a lower MSE, indicating that Lasso prediction are closer to the actual stock A return
2. The Lasso model has a higher R-squared, indicating that Lasso model explains more of the variance in Stock A return
3. From the plot, we can also find out Lasso regression test data are closer to the actual Stock A return

Lasso regression provides a better fit compared to OLS linear regression, based on both the lower MSE and higher R-squared. The Lasso model also appears to generalize better to the test data, which is reflected in the more accurate predictions seen in the plots.