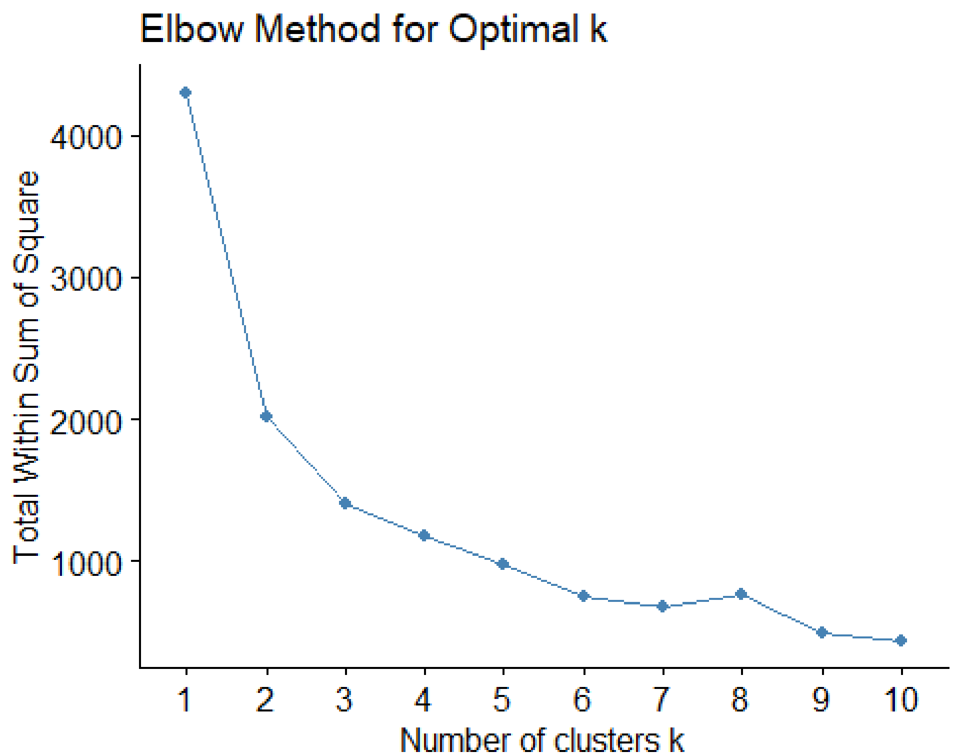
HEDGE_FUND_EDA

Nandan Soni

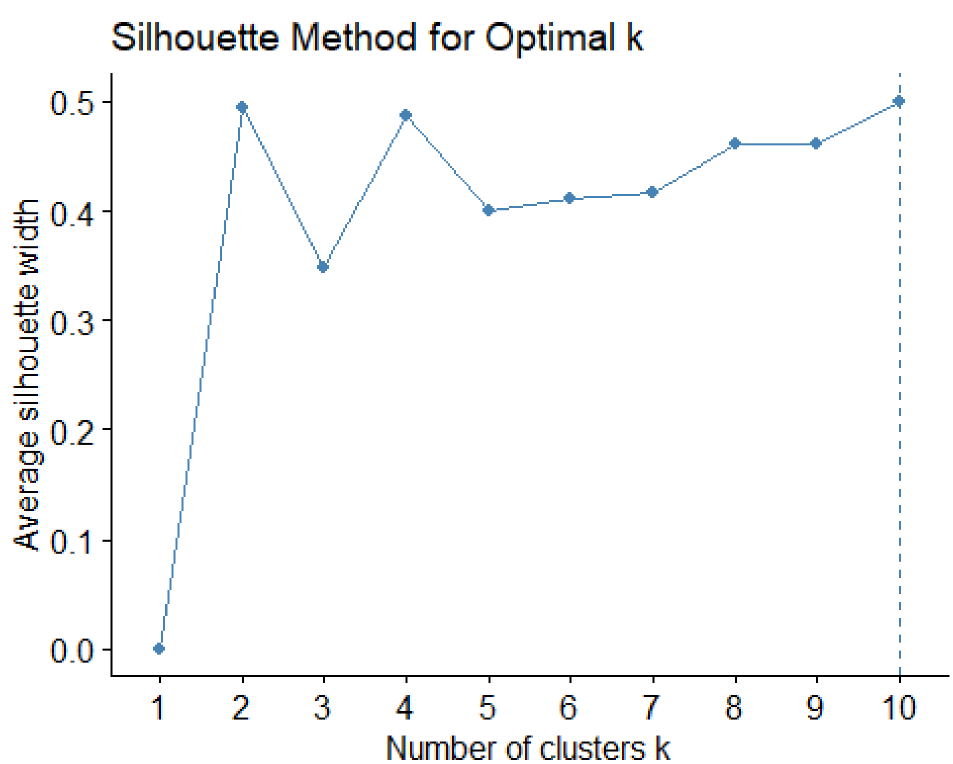
2024-12-19

```
library(tidyverse)

## Warning: package 'tidyverse' was built under R version 4.3.3

## Warning: package 'ggplot2' was built under R version 4.3.3

## ── Attaching core tidyverse packages ──────────────────────── tidyverse
2.0.0 ──
## ✔ dplyr     1.1.3     ✔ readr     2.1.4
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ ggplot2   3.5.0     ✔ tibble    3.2.1
## ✔ lubridate 1.9.2     ✔ tidyr     1.3.0
## ✔ purrr     1.0.2
## ── Conflicts ──────────────────────────────────────────
tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all
conflicts to become errors

library(ggplot2)
require(caret)

## Loading required package: caret

## Warning: package 'caret' was built under R version 4.3.2

## Loading required package: lattice
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift

require(factoextra)

## Loading required package: factoextra

## Warning: package 'factoextra' was built under R version 4.3.3

## Welcome! Want to learn more? See two factoextra-related books at
https://goo.gl/ve3WBa

require(useful)
```

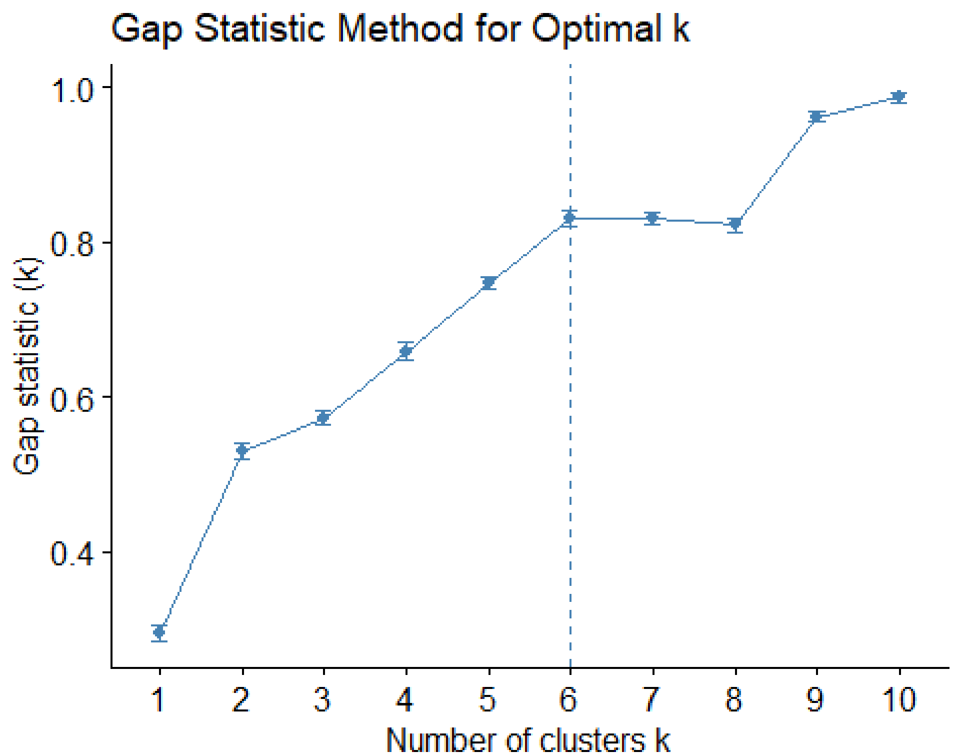
```
## Loading required package: useful

## Warning: package 'useful' was built under R version 4.3.3

library(stats)

setwd("D:/3 rd/401")
data <- read.csv("hedge_fund.csv")

str(data)

## 'data.frame':    478 obs. of  10 variables:
##  $ Date                 : int  31048 31079 31107 31138 31168 31199 31229
31260 31291 31321 ...
##  $ M2V                  : num  1.8 1.8 1.8 1.79 1.79 ...
##  $ UNRATE               : num  7.3 7.2 7.2 7.3 7.2 7.4 7.4 7.1 7.1 7.1 ...
##  $ CPIAUCNS             : num  106 106 106 107 107 ...
##  $ PPIACO               : num  103 103 103 103 104 ...
##  $ FEDFUNDS             : num  8.35 8.5 8.58 8.27 7.97 7.53 7.88 7.9 7.92
7.99 ...
##  $ Adj.Close            : num  180 181 181 180 190 ...
##  $ GOLD                 : chr  "325.3" "321.9" "345.4" "338.9" ...
##  $ WTISPLC..crude.oil..: num  25.6 27.3 28.2 28.8 27.6 ...
##  $ PERMIT..Units..000s : int  1660 1662 1727 1664 1709 1716 1697 1808 1916
1743 ...

data$GOLD <- as.numeric(gsub("[^0-9.]", "", data$GOLD))

data$PERMIT..Units..000s <- as.numeric(data$PERMIT..Units..000s)

scaled_data <- scale(data)

pca_result <- prcomp(scaled_data, center = TRUE, scale. = TRUE)

data_cluster <- data %>%
  select(-Date) %>%
  na.omit() %>%
  scale()
str(data_cluster)

##  num [1:478, 1:9] 0.0977 0.0977 0.0977 0.0839 0.0839 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : chr [1:478] "1" "2" "3" "4" ...
##   ..$ : chr [1:9] "M2V" "UNRATE" "CPIAUCNS" "PPIACO" ...
##  - attr(*, "scaled:center")= Named num [1:9] 1.77 5.78 195 160.84 3.44 ...
##   ..- attr(*, "names")= chr [1:9] "M2V" "UNRATE" "CPIAUCNS" "PPIACO" ...
##  - attr(*, "scaled:scale")= Named num [1:9] 0.29 1.67 54.6 45.37 2.76 ...
##   ..- attr(*, "names")= chr [1:9] "M2V" "UNRATE" "CPIAUCNS" "PPIACO" ...

fviz_nbclust(data_cluster, kmeans, method = "wss") +
  labs(title = "Elbow Method for Optimal k")
```



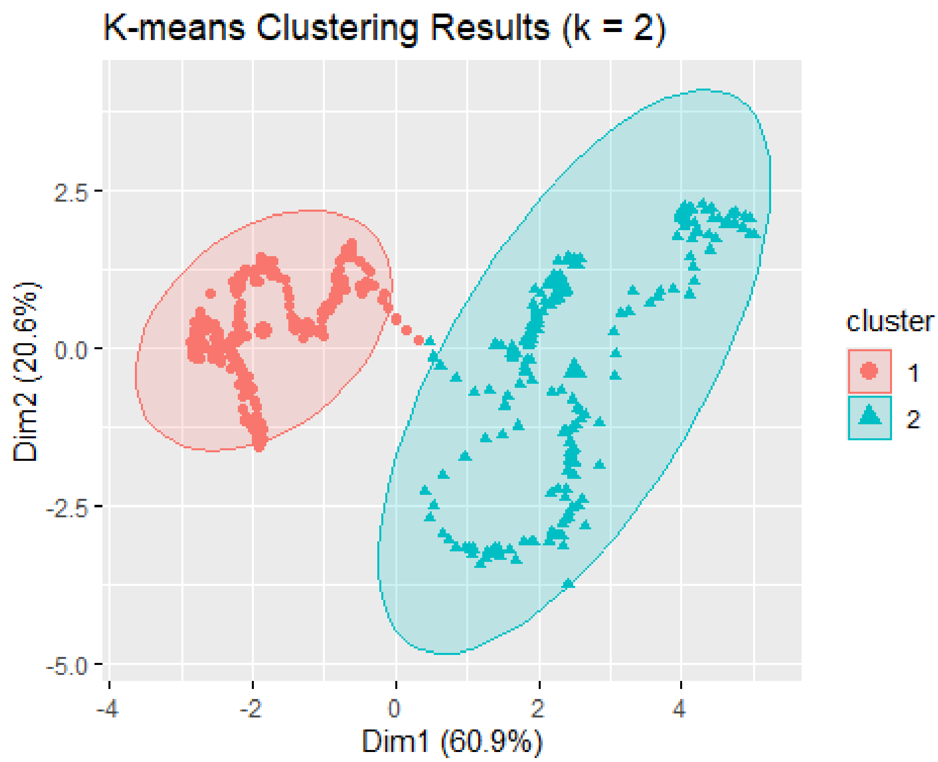

```
fviz_nbclust(data_cluster, kmeans, method = "silhouette") +
  labs(title = "Silhouette Method for Optimal k")
```



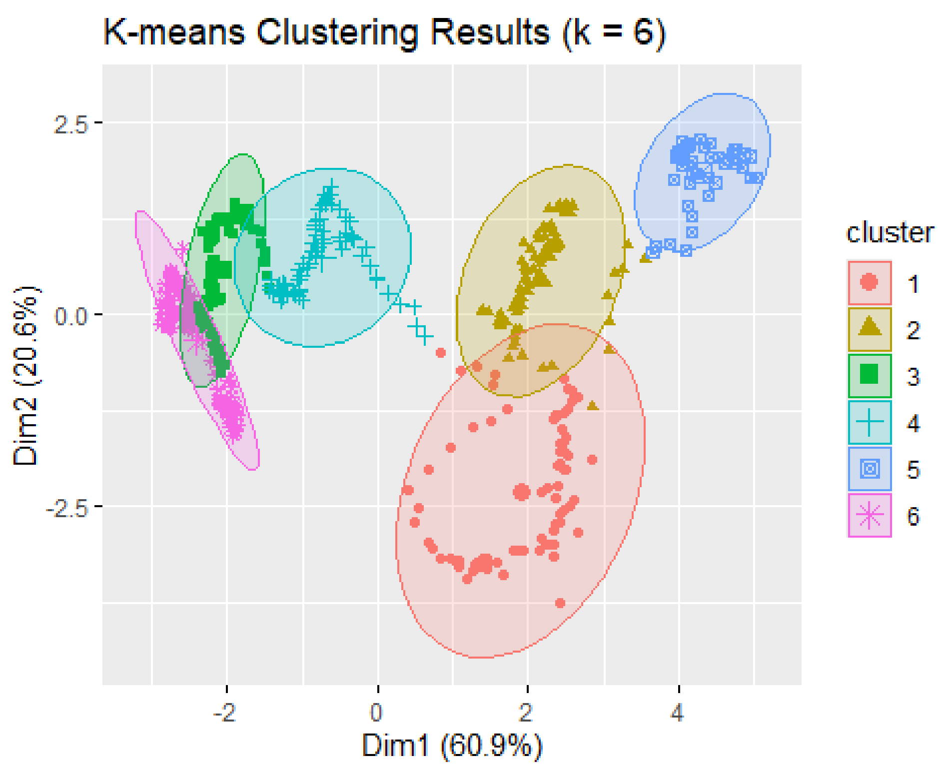
```r
fviz_nbclust(data_cluster, kmeans, method = "gap_stat", nboot = 20) +
  labs(title = "Gap Statistic Method for Optimal k")
```



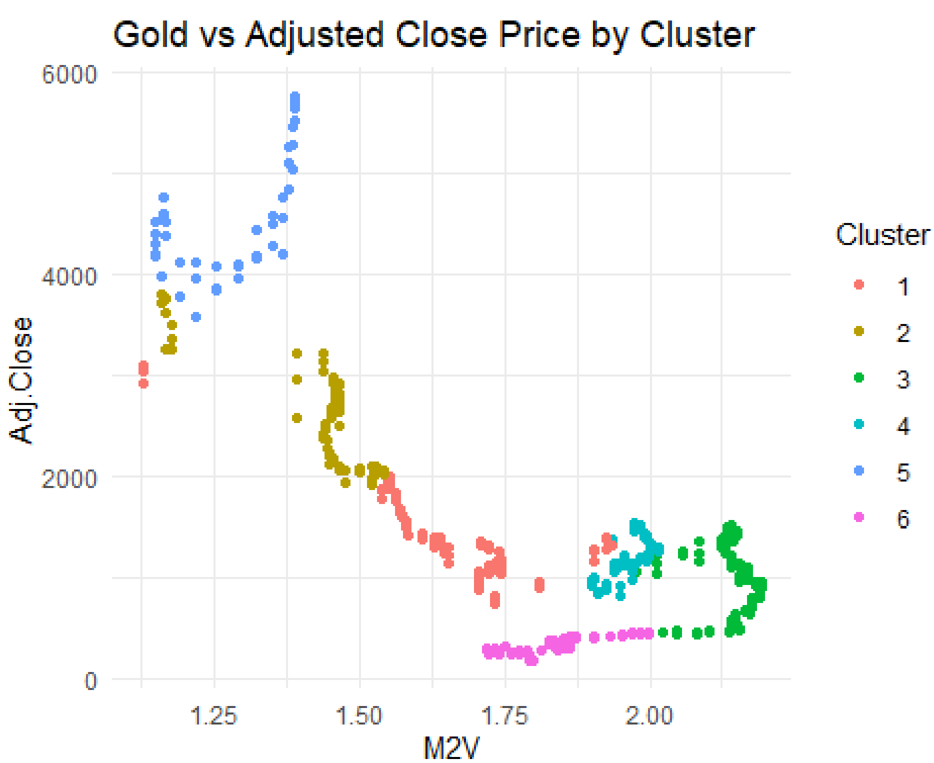

```r
k2 <- kmeans(data_cluster, centers = 2, nstart = 25)
k6 <- kmeans(data_cluster, centers = 6, nstart = 25)




fviz_cluster(k2, data = data_cluster, geom = "point", ellipse.type = "norm")
+
  labs(title = "K-means Clustering Results (k = 2)")
```




```r
fviz_cluster(k6, data = data_cluster, geom = "point", ellipse.type = "norm")
+
  labs(title = "K-means Clustering Results (k = 6)")
```



```
data$Cluster <- k6$cluster


head(data)

##    Date   M2V UNRATE CPIAUCNS PPIACO FEDFUNDS Adj.Close  GOLD
## 1 31048 1.799    7.3    105.5  103.4     8.35    179.63 325.3
## 2 31079 1.799    7.2    106.0  103.3     8.50    181.18 321.9
## 3 31107 1.799    7.2    106.4  103.1     8.58    180.66 345.4
## 4 31138 1.795    7.3    106.9  103.3     8.27    179.83 338.9
## 5 31168 1.795    7.2    107.3  103.5     7.97    189.55 345.7
## 6 31199 1.795    7.4    107.6  103.3     7.53    191.85 340.4
##   WTISPLC..crude.oil.. PERMIT..Units..000s Cluster
## 1               25.641                1660       6
## 2               27.271                1662       6
## 3               28.238                1727       6
## 4               28.805                1664       6
## 5               27.623                1709       6
## 6               27.143                1716       6

ggplot(data, aes(x = M2V, y = Adj.Close, color = factor(Cluster))) +
  geom_point() +
  labs(title = "Gold vs Adjusted Close Price by Cluster",
       color = "Cluster") +
  theme_minimal()
```



Project Code

December 18, 2024

[29]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from pandas.plotting import register_matplotlib_converters
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
import warnings
warnings.filterwarnings('ignore')
pd.pandas.set_option('display.max.columns',None)
plt.style.use('fivethirtyeight')
```

[30]:
```
data = pd.read_csv('F:/Okanagan College/3rd Semester/DSCI 401/Project/Final␣
↪Data.csv')
data['Date'] = pd.to_datetime(data['Date'])
```

Rename

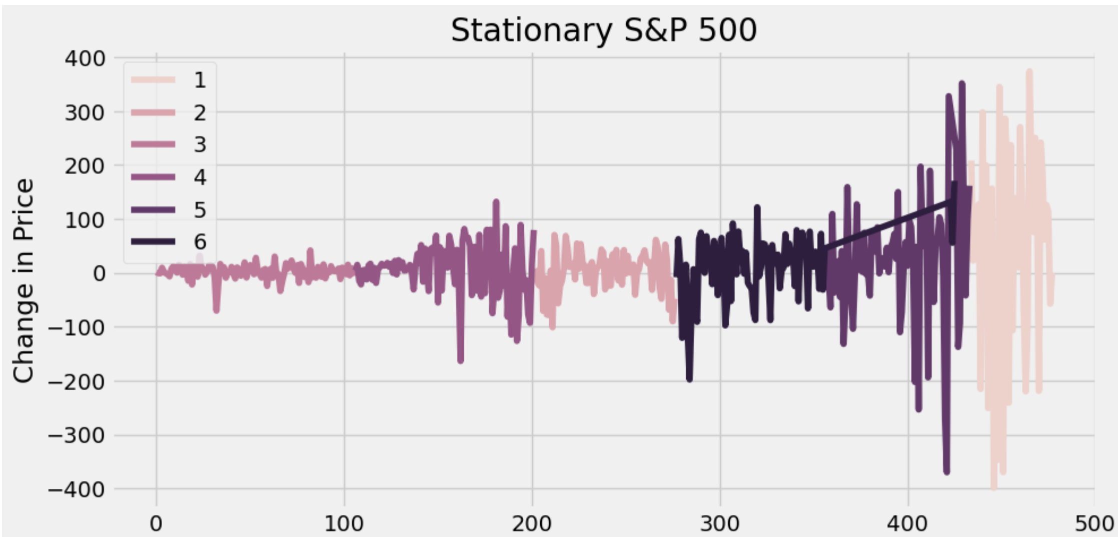
[31]:
```
data.rename(columns={'DATE': 'Date','Adj.Close':'SP500','WTISPLC..crude.oil..':
↪'OIL','PERMIT..Units..000s':'PERMITS','CPIAUCNS':'CPI','PPIACO':'PPI'},␣
↪inplace=True)
data.tail()
```

[31]:
```
           Date    M2V  UNRATE      CPI      PPI  FEDFUNDS        SP500  \
473  2024-06-01  1.385     4.1  314.175  255.914      5.33  5460.479980
474  2024-07-01  1.389     4.3  314.540  257.326      5.33  5522.299805
475  2024-08-01  1.389     4.2  314.796  255.394      5.33  5648.399902
476  2024-09-01  1.389     4.1  315.301  252.737      5.13  5762.479980
477  2024-10-01  1.389     4.1  315.664  253.452      4.83  5705.450195

       GOLD    OIL  PERMITS  Cluster         DXY
473  2352.1  79.77     1454        1  105.870003
474  2326.3  81.80     1406        1  104.099998
475  2395.3  76.68     1470        1  101.699997
476  2468.0  70.24     1425        1  100.779999
477  2567.1  71.99     1419        1  103.980003
```

0.1 First Difference

[32]:
```
first_diffs = data['SP500'].values[1:] - data['SP500'].values[:-1]
first_diffs = np.concatenate([first_diffs, [0]])
data['FirstDifference'] = first_diffs
```

[33]:
```
plt.figure(figsize=(10, 5))
plt.title('Stationary S&P 500')
sns.lineplot(data=data, x=data.index, y='FirstDifference', hue='Cluster')
plt.ylabel('Change in Price')

plt.legend(loc='upper left', bbox_to_anchor=(0, 1))
plt.show()
```



0.2 Subsetting Different Phases Based On Clusters

[34]:
```
phase_1 = data[data['Cluster']==1].reset_index(drop=True)
phase_1 = phase_1.drop(columns='Cluster',axis=1)

phase_2 = data[data['Cluster']==2].reset_index(drop=True)
phase_2 = phase_2.drop(columns='Cluster',axis=1)

phase_3 = data[data['Cluster']==3].reset_index(drop=True)
phase_3 = phase_3.drop(columns='Cluster',axis=1)

phase_4 = data[data['Cluster']==4].reset_index(drop=True)
phase_4 = phase_4.drop(columns='Cluster',axis=1)

phase_5 = data[data['Cluster']==5].reset_index(drop=True)
```

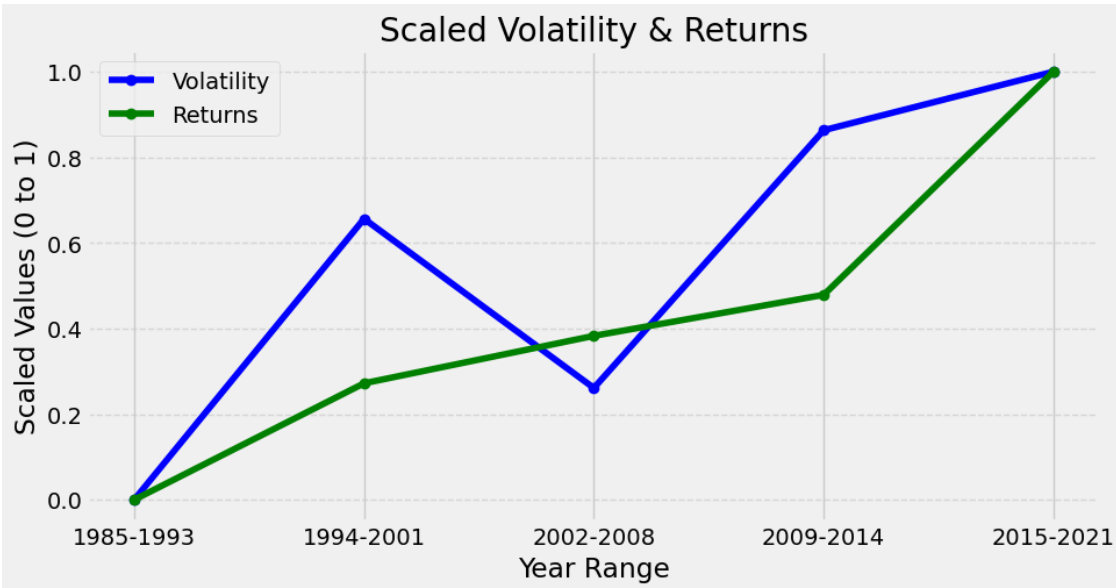
```
phase_5 = phase_5.drop(columns='Cluster',axis=1)

phase_6 = data[data['Cluster']==6].reset_index(drop=True)
phase_6 = phase_6.drop(columns='Cluster',axis=1)
```

[163]:
```
datasets = [phase_1, phase_2, phase_3, phase_4, phase_5, phase_6]

i=1
for x in datasets:
    print(f'Phase {i} Mean: {x.SP500.mean()}')
    print(f'Phase {i} Std: {x.SP500.std()}')
    i=i+1
```

```
Phase 1 Mean: 4505.272705068182
Phase 1 Std: 550.622871120827
Phase 2 Mean: 1184.0263940546665
Phase 2 Std: 189.1365529688897
Phase 3 Mean: 318.7554709990566
Phase 3 Std: 81.29431989539988
Phase 4 Mean: 933.6120818458334
Phase 4 Std: 352.4023389472756
Phase 5 Mean: 2578.4877154324327
Phase 5 Std: 494.34155616854997
Phase 6 Mean: 1400.4174650132531
Phase 6 Std: 437.84225758861453
```

[184]:
```
std = [81.29431989539988, 352.4023389472756, 189.1365529688897, 437.
 ↪84225758861453, 494.34155616854997]
mean = [318.7554709990566, 933.6120818458334, 1184.0263940546665, 1400.
 ↪4174650132531, 2578.4877154324327]
x = ["1985-1993", "1994-2001", "2002-2008", "2009-2014", "2015-2021"]



def min_max_scale(values):
    min_val = min(values)
    max_val = max(values)
    return [(v - min_val) / (max_val - min_val) for v in values]

scaled_std = min_max_scale(std)
scaled_mean = min_max_scale(mean)

plt.figure(figsize=(10, 5))

plt.plot(x, scaled_std, marker='o', label='Volatility', color='blue')
plt.plot(x, scaled_mean, marker='o', label='Returns', color='green')
```

```
plt.title("Scaled Volatility & Returns")
plt.xlabel("Year Range")
plt.ylabel("Scaled Values (0 to 1)")


plt.legend()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

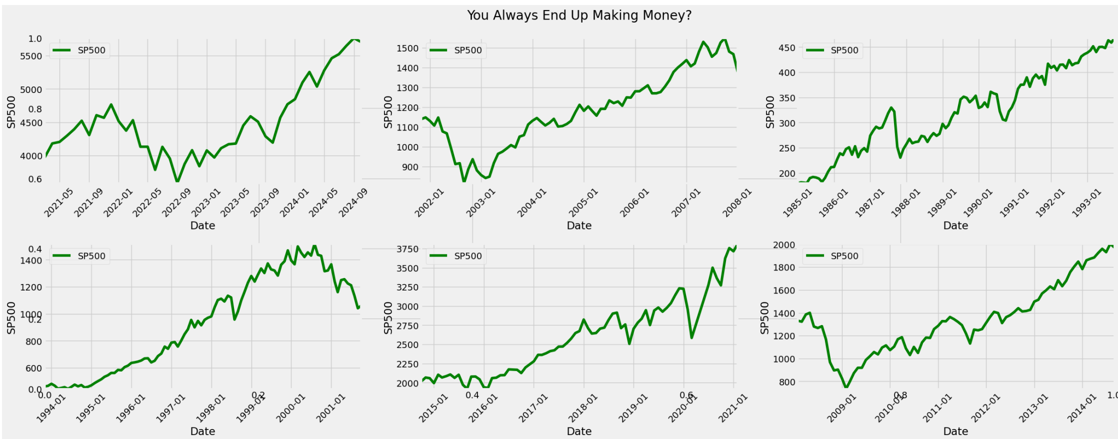


0.2.1 95% CI for phase duration is 6.309 – 6.491

0.3 Price around different phases

[149]:
```
import matplotlib.dates as mdates

datasets = [phase_1, phase_2, phase_3, phase_4, phase_5, phase_6.head(80)]
titles = ['Phase 1', 'Phase 2', 'Phase 3', 'Phase 4', 'Phase 5', 'Phase 6']

fig = plt.figure(figsize=(25, 10))
plt.title('You Always End Up Making Money?\n')

for i, (data, title) in enumerate(zip(datasets, titles)):
    ax = fig.add_subplot(2, 3, i + 1)
    ax.plot(data['Date'], data['SP500'], label='SP500', color='green')
    ax.set_xlabel('Date')
    ax.set_ylabel('SP500')
    ax.grid(True)
    ax.legend()
```

```
    ax.set_xlim(data['Date'].min(), data['Date'].max())
    ax.set_ylim(data['SP500'].min(), data['SP500'].max())

    ax.xaxis.set_major_formatter(mdates.DateFormatter('%Y-%m'))
    ax.xaxis.set_major_locator(mdates.AutoDateLocator())
    plt.setp(ax.xaxis.get_majorticklabels(), rotation=45)


plt.tight_layout()
plt.show()
```



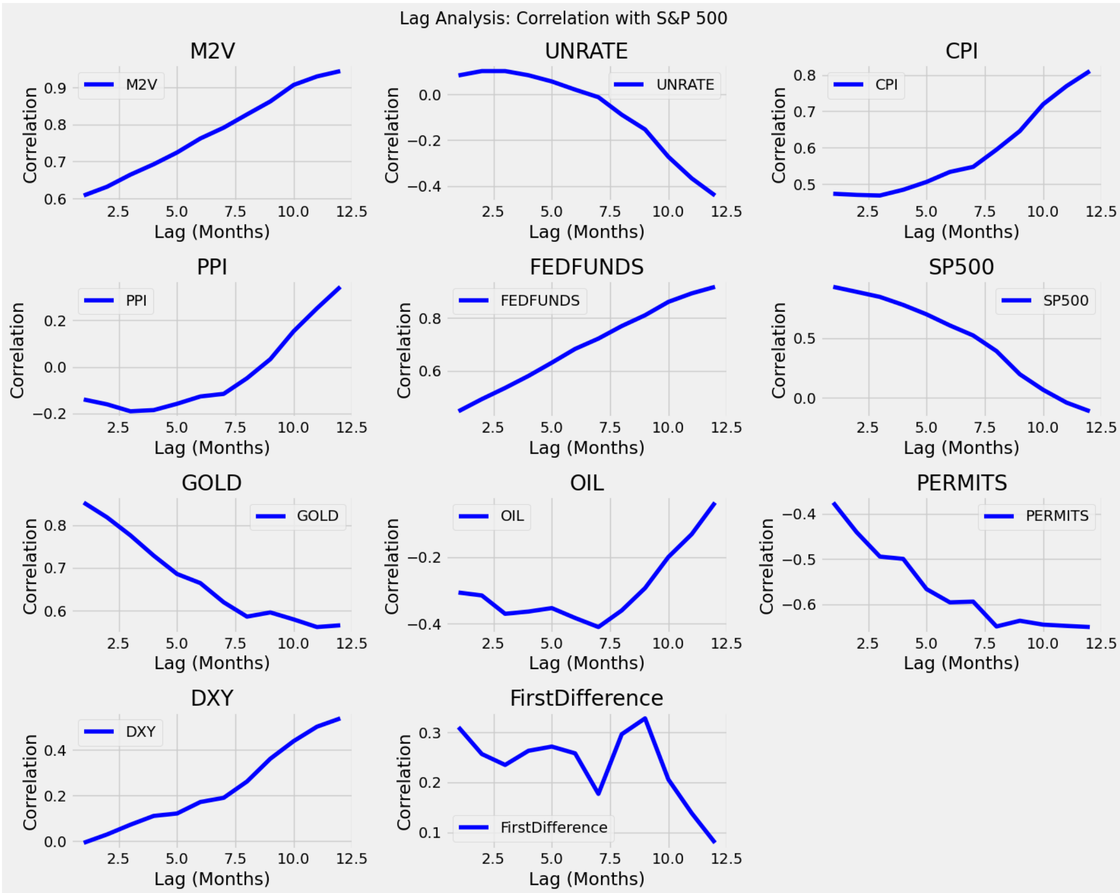
[35]:
```
def lag_analysis(dataframe, target_column, lag_range):
    lagged_correlations = {}
    data = dataframe.copy()

    for col in data.columns:
        if col != 'Date':  # Skip the Date column
            correlations = []
            for lag in range(1, lag_range + 1):
                data[f'{col}_lag{lag}'] = data[col].shift(lag)
                corr = data[target_column].corr(data[f'{col}_lag{lag}'])
                correlations.append(corr)
            lagged_correlations[col] = correlations

    return lagged_correlations
```

[36]:
```
datasets = [phase_1,phase_2,phase_3,phase_4,phase_5,phase_6]
```

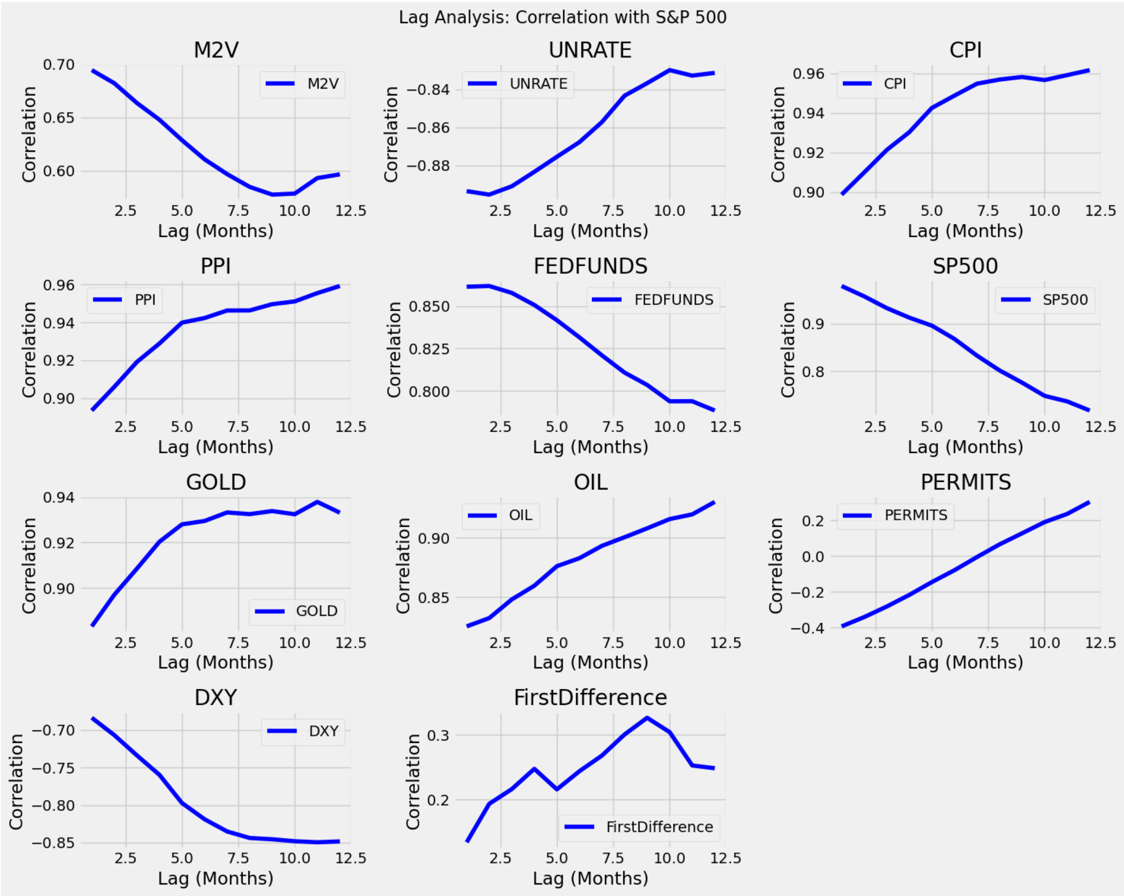
[37]:
```
import matplotlib.pyplot as plt

rows = 4
```

```python
cols = 3
j = 1
for x in datasets:
    lag_range = 12
    lagged_correlations = lag_analysis(x, target_column='SP500',␣
 ↪lag_range=lag_range)

    print(j)
    # Plot the results using subplots
    features = list(lagged_correlations.keys())
    num_features = len(features)

    fig, axes = plt.subplots(rows, cols, figsize=(15, 12))
    fig.suptitle('Lag Analysis: Correlation with S&P 500', fontsize=16)

    for i, (feature, correlations) in enumerate(lagged_correlations.items()):
        if i >= rows * cols:  # Check if the number of features exceeds␣
 ↪available subplots
            break
        row, col = divmod(i, cols)
        ax = axes[row, col]
        ax.plot(range(1, lag_range + 1), correlations, label=feature, color='b')
        ax.set_title(f'{feature}')
        ax.set_xlabel('Lag (Months)')
        ax.set_ylabel('Correlation')
        ax.grid(True)
        ax.legend()

    # Hide unused subplots (if any)
    for i in range(num_features, rows * cols):
        fig.delaxes(axes.flatten()[i])

    plt.tight_layout()
    plt.show()

    j += 1  # Increment j
```

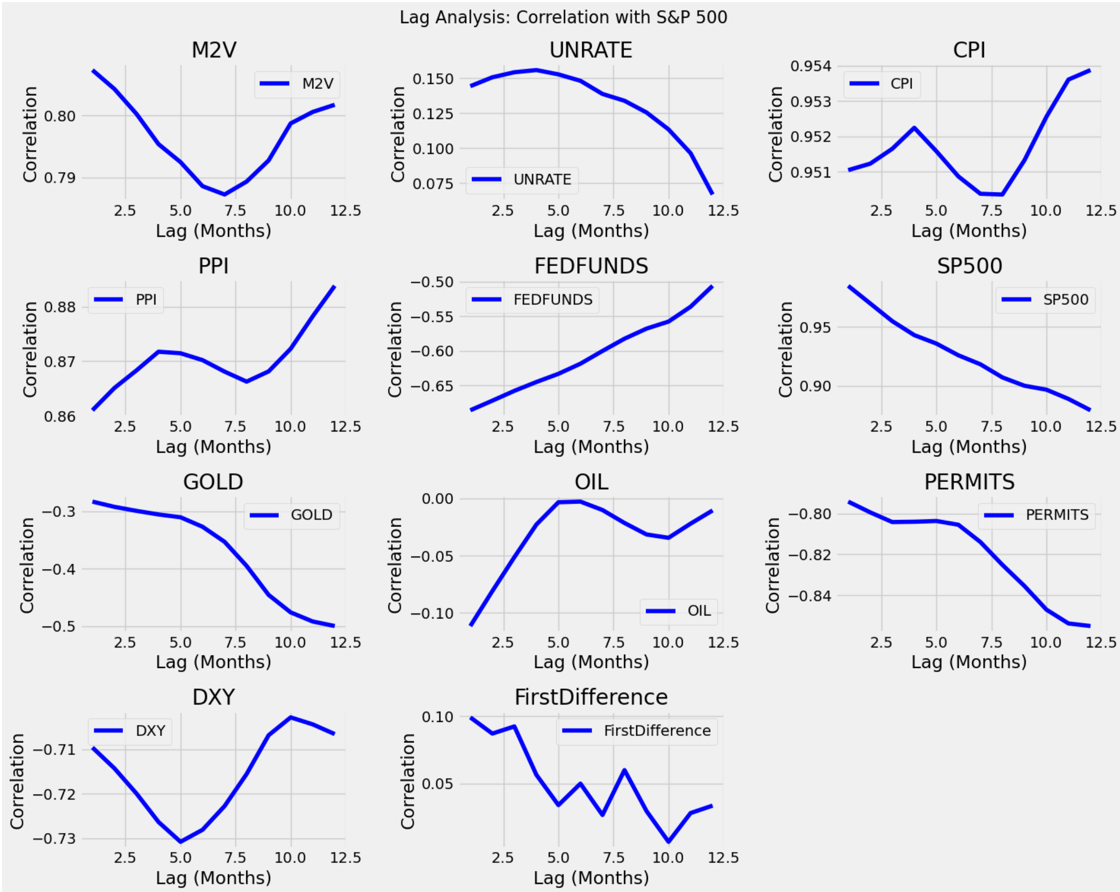
```
1
```


Lag Analysis: Correlation with S&P 500
M2V
UNRATE
CPI
PPI
FEDFUNDS
SP500
GOLD
OIL
PERMITS
DXY
FirstDifference
Correlation
Lag (Months)

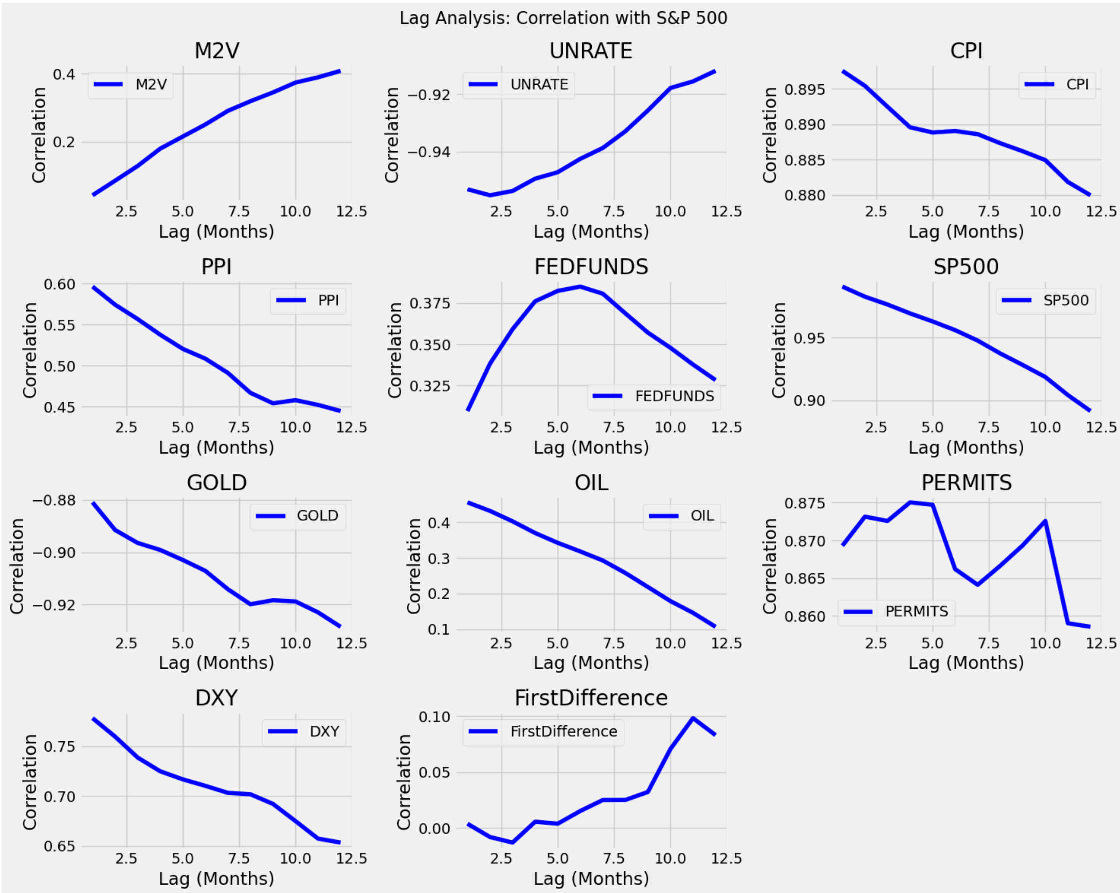
2


Lag Analysis: Correlation with S&P 500
M2V
UNRATE
CPI
PPI
FEDFUNDS
SP500
GOLD
OIL
PERMITS
DXY
FirstDifference
Correlation
Lag (Months)

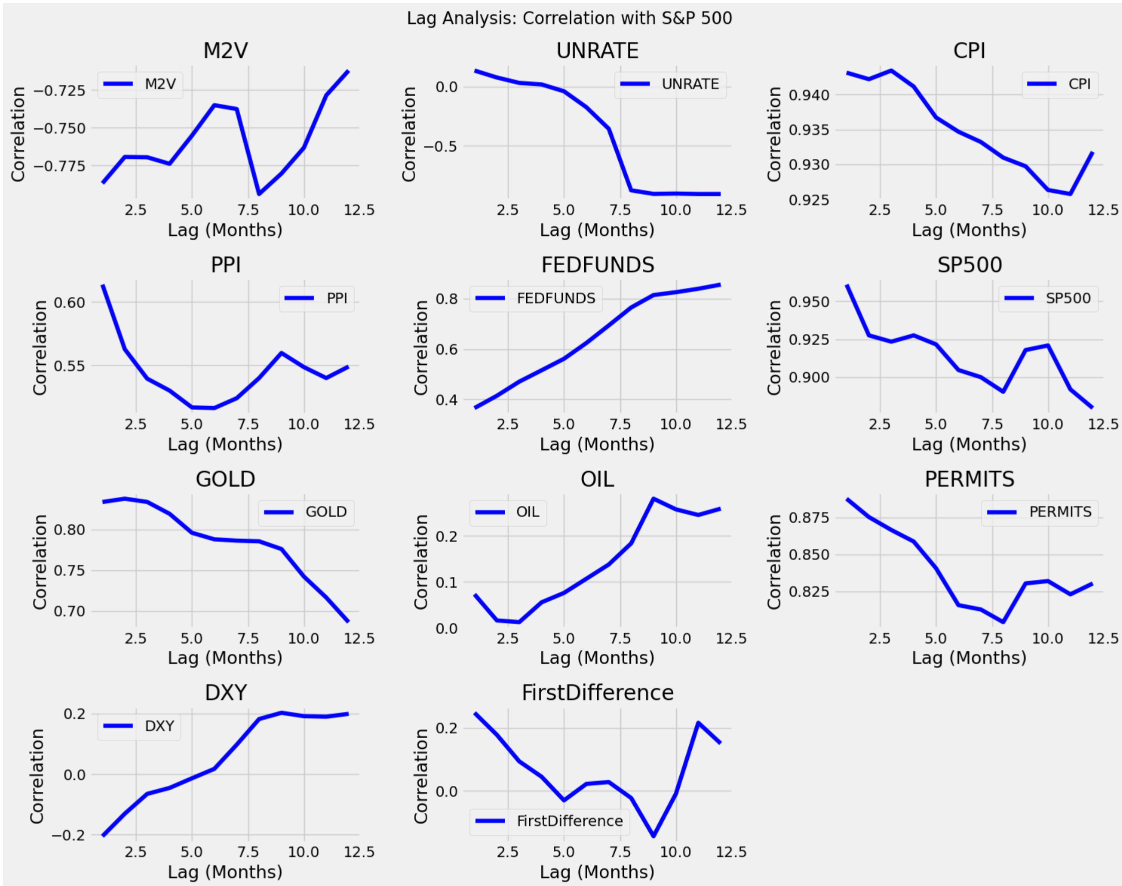
3


Lag Analysis: Correlation with S&P 500
M2V
UNRATE
CPI
PPI
FEDFUNDS
SP500
GOLD
OIL
PERMITS
DXY
FirstDifference
Correlation
Lag (Months)

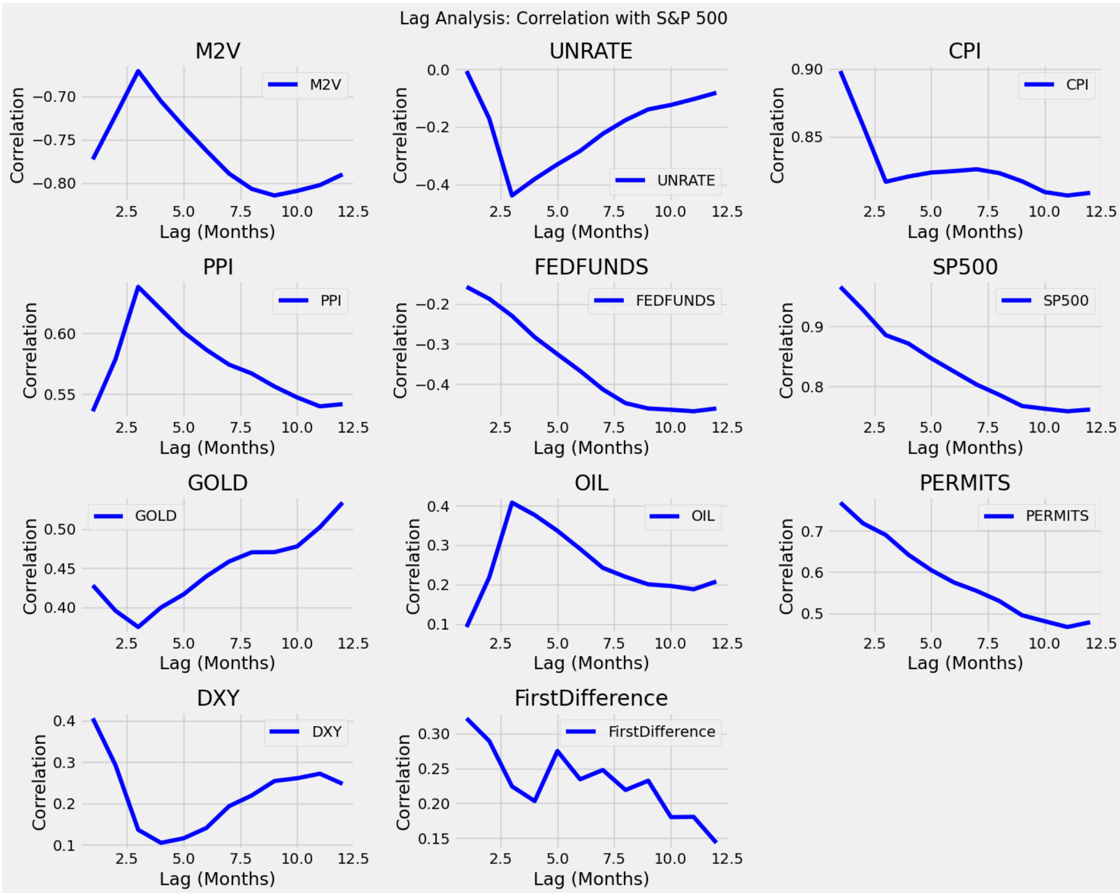
4


Lag Analysis: Correlation with S&P 500
M2V
UNRATE
CPI
PPI
FEDFUNDS
SP500
GOLD
OIL
PERMITS
DXY
FirstDifference
Correlation
Lag (Months)

5


Lag Analysis: Correlation with S&P 500
M2V
UNRATE
CPI
PPI
FEDFUNDS
SP500
GOLD
OIL
PERMITS
DXY
FirstDifference
Correlation
Lag (Months)

6



[38]:
```python
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense
```

[39]:
```python
feature_lag_mapping = {
    'M2V': 12,        # Lag 2 for M2 velocity
    'UNRATE': 3,     # Lag 1 for Unemployment rate
    'CPI': 12,        # Lag 4 for CPI
    'PPI': 12,       # Lag 12 for PPI
    'FEDFUNDS': 12,  # Lag 8 for Fed Funds Rate
    'PERMITS': 1,    # Lag 2 for Building Permits
    'DXY': 12,        # Lag 4 for Dollar Index
    'GOLD': 1,       # Lag 1 for Gold
    'OIL': 12,       # Lag 12 for Oil
    'SP500': 1       # Lag 1 for S&P 500
}


def apply_lags(data, lag_mapping):
```

```
    lagged_data = data.copy()
    for feature, lag in lag_mapping.items():
        for i in range(1, lag + 1):
            lagged_data[f'{feature}_lag_{i}'] = data[feature].shift(i)
    return lagged_data


df_lagged = apply_lags(phase_1, feature_lag_mapping)
df_lagged = df_lagged.dropna().reset_index(drop=True)
df_lagged =␣
↪df_lagged[['Date','SP500','FirstDifference','M2V_lag_12','UNRATE_lag_3','CPI_lag_12',
                  ␣
↪'PPI_lag_12','FEDFUNDS_lag_12','PERMITS_lag_1','DXY_lag_12','GOLD_lag_1',
                    'OIL_lag_12','SP500_lag_1']]
```

[40]:
```
df_lagged.tail()
```

[40]:
```
          Date        SP500  FirstDifference  M2V_lag_12  UNRATE_lag_3  \
27 2024-06-01  5460.479980        61.819825       1.322           3.8
28 2024-07-01  5522.299805       126.100097       1.349           3.9
29 2024-08-01  5648.399902       114.080078       1.349           4.0
30 2024-09-01  5762.479980       -57.029785       1.349           4.1
31 2024-10-01  5705.450195         0.000000       1.368           4.3

    CPI_lag_12  PPI_lag_12  FEDFUNDS_lag_12  PERMITS_lag_1  DXY_lag_12  \
27     305.109     253.860             5.08         1399.0  102.910004
28     305.691     253.835             5.12         1454.0  101.860001
29     307.026     257.680             5.33         1406.0  103.620003
30     307.789     258.934             5.33         1470.0  106.169998
31     307.671     255.192             5.33         1425.0  106.660004

    GOLD_lag_1  OIL_lag_12  SP500_lag_1
27      2335.5       70.25  5277.509766
28      2352.1       76.07  5460.479980
29      2326.3       81.39  5522.299805
30      2395.3       89.43  5648.399902
31      2468.0       85.64  5762.479980
```

0.4 Data Split and Scaling

[41]:
```
features = [features for features in df_lagged.columns if features not in␣
↪['SP500','Date']]
```

[42]:
```
X = df_lagged.drop(columns=['SP500','Date'])
y = df_lagged['SP500']

train_size = int(len(X) * 0.8)
```

```
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

[43]:
```
X.tail()
```

[43]:
```
    FirstDifference  M2V_lag_12  UNRATE_lag_3  CPI_lag_12  PPI_lag_12  \
27        61.819825       1.322           3.8     305.109     253.860
28       126.100097       1.349           3.9     305.691     253.835
29       114.080078       1.349           4.0     307.026     257.680
30       -57.029785       1.349           4.1     307.789     258.934
31         0.000000       1.368           4.3     307.671     255.192

    FEDFUNDS_lag_12  PERMITS_lag_1  DXY_lag_12  GOLD_lag_1  OIL_lag_12  \
27             5.08         1399.0  102.910004      2335.5       70.25
28             5.12         1454.0  101.860001      2352.1       76.07
29             5.33         1406.0  103.620003      2326.3       81.39
30             5.33         1470.0  106.169998      2395.3       89.43
31             5.33         1425.0  106.660004      2468.0       85.64

    SP500_lag_1
27  5277.509766
28  5460.479980
29  5522.299805
30  5648.399902
31  5762.479980
```

0.5 Feature Scaling

[44]:
```
scaler = StandardScaler()
target = StandardScaler()


target.fit(np.array(y_train).reshape(-1, 1))
scaler.fit(X_train)


X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)


y_train = target.transform(np.array(y_train).reshape(-1, 1))
y_test = target.transform(np.array(y_test).reshape(-1, 1))
```

[45]:
```
X_train = pd.concat([df_lagged['SP500'].reset_index(drop=True),pd.
 ↪DataFrame(scaler.transform(X_train),columns=features)],axis=1)
```

[46]:
```
X_train.head()
```

[46]:
```
        SP500  FirstDifference  M2V_lag_12  UNRATE_lag_3  CPI_lag_12  \
0  4530.410156        -1.831169   -0.765490      1.769076   -1.751671
1  4131.930176        -0.087434   -0.971932      2.449490   -1.567918
2  4132.149902        -1.605013   -0.971932      1.088662   -1.387204
3  3785.379883         1.420089   -0.971932     -0.272166   -1.176104
4  4130.290039        -0.855036   -0.951288      0.408248   -1.065785

   PPI_lag_12  FEDFUNDS_lag_12  PERMITS_lag_1  DXY_lag_12  GOLD_lag_1  \
0   -1.978805        -0.777155       2.095554   -0.991909   -0.599082
1   -1.817149        -0.777155       2.238879   -1.295975   -0.214055
2   -1.426946        -0.783081       1.906969   -1.520517    0.677842
3   -1.203973        -0.771230       0.979132   -1.115095    0.542351
4   -1.039530        -0.759380       1.228064   -1.157197   -0.292035

   OIL_lag_12  SP500_lag_1
0   -1.390296     0.358384
1   -1.431688     0.807943
2   -1.197588    -0.336940
3   -0.776208    -0.336309
4   -0.700888    -1.332623
```

0.6 Initial Model Buiding

[227]:
```
model = Sequential()
model.add(Dense(64, input_dim=X_train_scaled.shape[1], activation='relu'))
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1))


model.compile(optimizer='adam', loss='mean_squared_error')


model.fit(X_train_scaled, y_train, epochs=45, batch_size=32,␣
 ↪validation_data=(X_test_scaled, y_test))


loss = model.evaluate(X_test_scaled, y_test)
print(f'Model Loss: {loss}')


predictions = model.predict(X_test_scaled)
```

```
Epoch 1/45
1/1              2s 2s/step - loss:
0.9987 - val_loss: 10.2041
Epoch 2/45
```

```
1/1            0s 88ms/step - loss:
0.9309 - val_loss: 9.4488
Epoch 3/45
1/1            0s 94ms/step - loss:
0.8722 - val_loss: 8.7513
Epoch 4/45
1/1            0s 84ms/step - loss:
0.8231 - val_loss: 8.1052
Epoch 5/45
1/1            0s 100ms/step - loss:
0.7781 - val_loss: 7.5063
Epoch 6/45
1/1            0s 96ms/step - loss:
0.7371 - val_loss: 6.9575
Epoch 7/45
1/1            0s 106ms/step - loss:
0.6992 - val_loss: 6.4513
Epoch 8/45
1/1            0s 97ms/step - loss:
0.6617 - val_loss: 5.9802
Epoch 9/45
1/1            0s 102ms/step - loss:
0.6262 - val_loss: 5.5888
Epoch 10/45
1/1            0s 110ms/step - loss:
0.5934 - val_loss: 5.3114
Epoch 11/45
1/1            0s 96ms/step - loss:
0.5643 - val_loss: 5.0767
Epoch 12/45
1/1            0s 101ms/step - loss:
0.5375 - val_loss: 4.8429
Epoch 13/45
1/1            0s 94ms/step - loss:
0.5123 - val_loss: 4.6087
Epoch 14/45
1/1            0s 93ms/step - loss:
0.4868 - val_loss: 4.3929
Epoch 15/45
1/1            0s 99ms/step - loss:
0.4613 - val_loss: 4.1834
Epoch 16/45
1/1            0s 96ms/step - loss:
0.4385 - val_loss: 3.9613
Epoch 17/45
1/1            0s 104ms/step - loss:
0.4182 - val_loss: 3.7178
Epoch 18/45
```

```
1/1            0s 115ms/step - loss:
0.3975 - val_loss: 3.4692
Epoch 19/45
1/1            0s 104ms/step - loss:
0.3776 - val_loss: 3.2085
Epoch 20/45
1/1            0s 107ms/step - loss:
0.3577 - val_loss: 2.9595
Epoch 21/45
1/1            0s 95ms/step - loss:
0.3407 - val_loss: 2.7167
Epoch 22/45
1/1            0s 98ms/step - loss:
0.3238 - val_loss: 2.4760
Epoch 23/45
1/1            0s 99ms/step - loss:
0.3073 - val_loss: 2.2593
Epoch 24/45
1/1            0s 105ms/step - loss:
0.2913 - val_loss: 2.0608
Epoch 25/45
1/1            0s 95ms/step - loss:
0.2764 - val_loss: 1.8501
Epoch 26/45
1/1            0s 91ms/step - loss:
0.2627 - val_loss: 1.6377
Epoch 27/45
1/1            0s 92ms/step - loss:
0.2494 - val_loss: 1.4435
Epoch 28/45
1/1            0s 99ms/step - loss:
0.2361 - val_loss: 1.2745
Epoch 29/45
1/1            0s 111ms/step - loss:
0.2236 - val_loss: 1.1315
Epoch 30/45
1/1            0s 97ms/step - loss:
0.2113 - val_loss: 1.0007
Epoch 31/45
1/1            0s 95ms/step - loss:
0.2004 - val_loss: 0.8781
Epoch 32/45
1/1            0s 87ms/step - loss:
0.1899 - val_loss: 0.7655
Epoch 33/45
1/1            0s 85ms/step - loss:
0.1796 - val_loss: 0.6717
Epoch 34/45
```

```
1/1              0s 82ms/step - loss:
0.1696 - val_loss: 0.5944
Epoch 35/45
1/1              0s 84ms/step - loss:
0.1603 - val_loss: 0.5217
Epoch 36/45
1/1              0s 93ms/step - loss:
0.1522 - val_loss: 0.4566
Epoch 37/45
1/1              0s 87ms/step - loss:
0.1444 - val_loss: 0.3994
Epoch 38/45
1/1              0s 85ms/step - loss:
0.1363 - val_loss: 0.3492
Epoch 39/45
1/1              0s 86ms/step - loss:
0.1280 - val_loss: 0.3052
Epoch 40/45
1/1              0s 84ms/step - loss:
0.1198 - val_loss: 0.2669
Epoch 41/45
1/1              0s 83ms/step - loss:
0.1124 - val_loss: 0.2359
Epoch 42/45
1/1              0s 85ms/step - loss:
0.1056 - val_loss: 0.2082
Epoch 43/45
1/1              0s 84ms/step - loss:
0.0989 - val_loss: 0.1821
Epoch 44/45
1/1              0s 85ms/step - loss:
0.0923 - val_loss: 0.1563
Epoch 45/45
1/1              0s 84ms/step - loss:
0.0854 - val_loss: 0.1317
1/1              0s 36ms/step - loss:
0.1317
Model Loss: 0.1316661387681961
1/1              0s 80ms/step
```

0.7 Initial Predictions & R squared

[48]:
```
target.inverse_transform(predictions)
```

[48]:
```
array([[5033.2017],
       [5232.2866],
       [5224.6724],
```

```
        [5247.2266],
        [5381.1904],
        [5413.9375],
        [5595.971 ]], dtype=float32)
```

[49]:
```
target.inverse_transform(y_test)
```

[49]:
```
array([[5035.689941],
       [5277.509766],
       [5460.47998 ],
       [5522.299805],
       [5648.399902],
       [5762.47998 ],
       [5705.450195]])
```

[228]:
```
r2 = r2_score(y_test, predictions)
print(f'R^2 Score: {r2}')
```

```
R^2 Score: 0.6312855463998615
```

0.8 Model Tuning

[50]:
```
import keras_tuner as kt
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

[51]:
```
def build_model(hp):
    model = Sequential()


    for i in range(hp.Int('num_layers', 2, 5)):

        model.add(Dense(hp.Int(f'num_units_{i}', min_value=8, max_value=128,␣
 ↪step=8),
                        activation='relu'))


    model.add(Dense(1))


    model.compile(optimizer=Adam(learning_rate=hp.Float('learning_rate',␣
 ↪min_value=1e-5, max_value=1e-2, sampling='LOG')),
                  loss='mean_squared_error')

    epochs = hp.Int('epochs', min_value=20, max_value=100, step=5)

    return model
```

[52]:
```
tuner = kt.Hyperband(
    build_model,
    objective='val_loss',
    max_epochs=30,
    hyperband_iterations=2,
    directory='loss',
    project_name='stock_market_tuning'
)


tuner.search(X_train_scaled, y_train, validation_data=(X_test_scaled, y_test),␣
 ↪batch_size=32)
```

```
Reloading Tuner from loss\stock_market_tuning\tuner0.json
```

[225]:
```
best_model = tuner.get_best_models(num_models=15)[10]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]


print("Best hyperparameters:", best_hyperparameters.values)

loss = best_model.evaluate(X_test_scaled, y_test)
print(f'Model Loss: {loss}')


predictions = best_model.predict(X_test_scaled)
```

```
Best hyperparameters: {'num_layers': 5, 'num_units_0': 64, 'num_units_1': 72,
'learning_rate': 0.007592175016682904, 'epochs': 65, 'num_units_2': 112,
'num_units_3': 40, 'num_units_4': 8, 'tuner/epochs': 10, 'tuner/initial_epoch':
0, 'tuner/bracket': 1, 'tuner/round': 0}
WARNING:tensorflow:6 out of the last 135 calls to <function
TensorFlowTrainer._make_function.<locals>.multi_step_on_iterator at
0x00000271CAAFAB60> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1              0s 156ms/step - loss:
0.0425
1/1              0s 156ms/step - loss:
0.0425
Model Loss: 0.04245270416140556
WARNING:tensorflow:6 out of the last 13 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
```

```
0x00000271CEA09620> triggered tf.function retracing. Tracing is expensive and
the excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/1              0s 77ms/step
```

0.9 Predictions & R Square

[55]:
```python
target.inverse_transform(predictions)
```

[55]:
```
array([[5089.971 ],
       [5236.091 ],
       [5501.4033],
       [5526.24  ],
       [5568.409 ],
       [5671.8945],
       [5787.5703]], dtype=float32)
```

[226]:
```python
from sklearn.metrics import r2_score

r2 = r2_score(y_test, predictions)
print(f'R^2 Score: {r2}')
```

```
R^2 Score: 0.8811165321700758
```

0.10 Errors

[58]:
```python
history = best_model.fit(
    X_train_scaled,
    y_train,
    validation_data=(X_test_scaled, y_test),
    epochs=best_hyperparameters['epochs'],
    batch_size=32
)


def plot_loss(history):
    plt.figure(figsize=(10, 4))
    plt.plot(history.history['loss'], label='Training Loss')
    if 'val_loss' in history.history:
        plt.plot(history.history['val_loss'], label='Validation Loss')
    plt.title('Model Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
```

```python
    plt.legend()
    plt.grid(True)
    plt.show()

plot_loss(history)
```

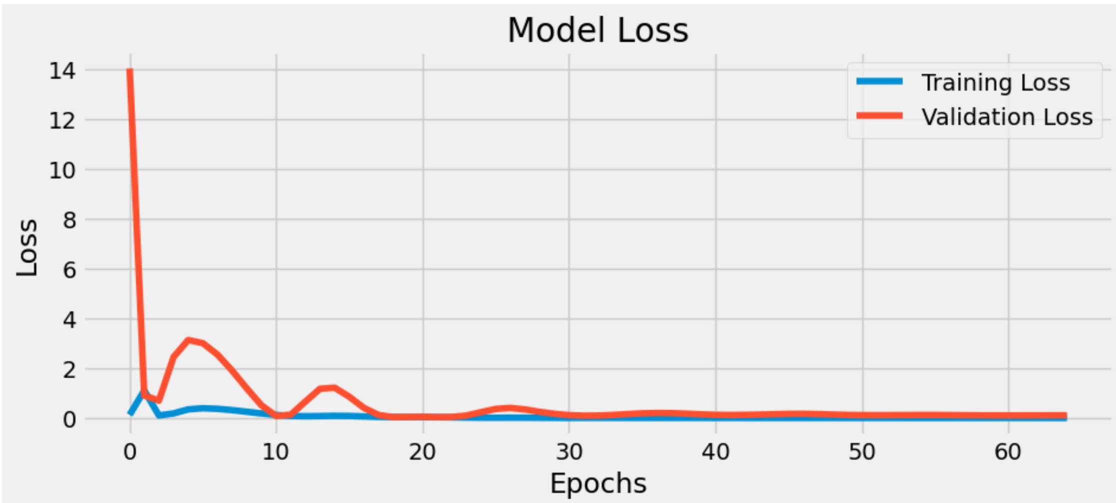
```
Epoch 1/65
1/1              2s 2s/step - loss:
0.1238 - val_loss: 14.0268
Epoch 2/65
1/1              0s 91ms/step - loss:
1.1043 - val_loss: 0.9003
Epoch 3/65
1/1              0s 103ms/step - loss:
0.0989 - val_loss: 0.6950
Epoch 4/65
1/1              0s 91ms/step - loss:
0.1826 - val_loss: 2.4468
Epoch 5/65
1/1              0s 96ms/step - loss:
0.3451 - val_loss: 3.1282
Epoch 6/65
1/1              0s 88ms/step - loss:
0.3903 - val_loss: 3.0005
Epoch 7/65
1/1              0s 89ms/step - loss:
0.3663 - val_loss: 2.5416
Epoch 8/65
1/1              0s 89ms/step - loss:
0.3143 - val_loss: 1.8979
Epoch 9/65
1/1              0s 86ms/step - loss:
0.2484 - val_loss: 1.1847
Epoch 10/65
1/1              0s 106ms/step - loss:
0.1817 - val_loss: 0.5069
Epoch 11/65
1/1              0s 89ms/step - loss:
0.1191 - val_loss: 0.0792
Epoch 12/65
1/1              0s 89ms/step - loss:
0.0800 - val_loss: 0.1292
Epoch 13/65
1/1              0s 85ms/step - loss:
0.0690 - val_loss: 0.6573
Epoch 14/65
1/1              0s 84ms/step - loss:
0.0760 - val_loss: 1.1715
```

```
Epoch 15/65
1/1            0s 91ms/step - loss:
0.0852 - val_loss: 1.2138
Epoch 16/65
1/1            0s 85ms/step - loss:
0.0798 - val_loss: 0.8496
Epoch 17/65
1/1            0s 93ms/step - loss:
0.0609 - val_loss: 0.4031
Epoch 18/65
1/1            0s 84ms/step - loss:
0.0435 - val_loss: 0.1190
Epoch 19/65
1/1            0s 86ms/step - loss:
0.0347 - val_loss: 0.0330
Epoch 20/65
1/1            0s 88ms/step - loss:
0.0334 - val_loss: 0.0408
Epoch 21/65
1/1            0s 98ms/step - loss:
0.0329 - val_loss: 0.0445
Epoch 22/65
1/1            0s 97ms/step - loss:
0.0293 - val_loss: 0.0339
Epoch 23/65
1/1            0s 94ms/step - loss:
0.0234 - val_loss: 0.0393
Epoch 24/65
1/1            0s 90ms/step - loss:
0.0158 - val_loss: 0.0961
Epoch 25/65
1/1            0s 84ms/step - loss:
0.0101 - val_loss: 0.2319
Epoch 26/65
1/1            0s 85ms/step - loss:
0.0094 - val_loss: 0.3635
Epoch 27/65
1/1            0s 92ms/step - loss:
0.0110 - val_loss: 0.4007
Epoch 28/65
1/1            0s 88ms/step - loss:
0.0102 - val_loss: 0.3412
Epoch 29/65
1/1            0s 88ms/step - loss:
0.0067 - val_loss: 0.2438
Epoch 30/65
1/1            0s 86ms/step - loss:
0.0035 - val_loss: 0.1663
```

```
Epoch 31/65
1/1            0s 86ms/step - loss:
0.0027 - val_loss: 0.1116
Epoch 32/65
1/1            0s 84ms/step - loss:
0.0032 - val_loss: 0.0906
Epoch 33/65
1/1            0s 95ms/step - loss:
0.0038 - val_loss: 0.0949
Epoch 34/65
1/1            0s 94ms/step - loss:
0.0035 - val_loss: 0.1195
Epoch 35/65
1/1            0s 88ms/step - loss:
0.0026 - val_loss: 0.1550
Epoch 36/65
1/1            0s 87ms/step - loss:
0.0020 - val_loss: 0.1855
Epoch 37/65
1/1            0s 89ms/step - loss:
0.0020 - val_loss: 0.1992
Epoch 38/65
1/1            0s 85ms/step - loss:
0.0020 - val_loss: 0.1940
Epoch 39/65
1/1            0s 82ms/step - loss:
0.0015 - val_loss: 0.1723
Epoch 40/65
1/1            0s 98ms/step - loss:
0.0012 - val_loss: 0.1446
Epoch 41/65
1/1            0s 85ms/step - loss:
9.4358e-04 - val_loss: 0.1243
Epoch 42/65
1/1            0s 85ms/step - loss:
9.4216e-04 - val_loss: 0.1187
Epoch 43/65
1/1            0s 87ms/step - loss:
8.6132e-04 - val_loss: 0.1240
Epoch 44/65
1/1            0s 82ms/step - loss:
6.8396e-04 - val_loss: 0.1352
Epoch 45/65
1/1            0s 86ms/step - loss:
4.5711e-04 - val_loss: 0.1498
Epoch 46/65
1/1            0s 90ms/step - loss:
4.1773e-04 - val_loss: 0.1633
```

```
Epoch 47/65
1/1            0s 92ms/step - loss:
4.7066e-04 - val_loss: 0.1664
Epoch 48/65
1/1            0s 84ms/step - loss:
5.0932e-04 - val_loss: 0.1549
Epoch 49/65
1/1            0s 84ms/step - loss:
4.3503e-04 - val_loss: 0.1365
Epoch 50/65
1/1            0s 86ms/step - loss:
3.1044e-04 - val_loss: 0.1211
Epoch 51/65
1/1            0s 84ms/step - loss:
2.3595e-04 - val_loss: 0.1127
Epoch 52/65
1/1            0s 87ms/step - loss:
2.1076e-04 - val_loss: 0.1105
Epoch 53/65
1/1            0s 87ms/step - loss:
2.0611e-04 - val_loss: 0.1123
Epoch 54/65
1/1            0s 86ms/step - loss:
1.7146e-04 - val_loss: 0.1158
Epoch 55/65
1/1            0s 91ms/step - loss:
1.2791e-04 - val_loss: 0.1186
Epoch 56/65
1/1            0s 98ms/step - loss:
9.7309e-05 - val_loss: 0.1181
Epoch 57/65
1/1            0s 91ms/step - loss:
1.0570e-04 - val_loss: 0.1136
Epoch 58/65
1/1            0s 87ms/step - loss:
1.0226e-04 - val_loss: 0.1077
Epoch 59/65
1/1            0s 88ms/step - loss:
8.5556e-05 - val_loss: 0.1034
Epoch 60/65
1/1            0s 87ms/step - loss:
5.9525e-05 - val_loss: 0.1014
Epoch 61/65
1/1            0s 90ms/step - loss:
5.6711e-05 - val_loss: 0.1009
Epoch 62/65
1/1            0s 105ms/step - loss:
6.9299e-05 - val_loss: 0.1013
```

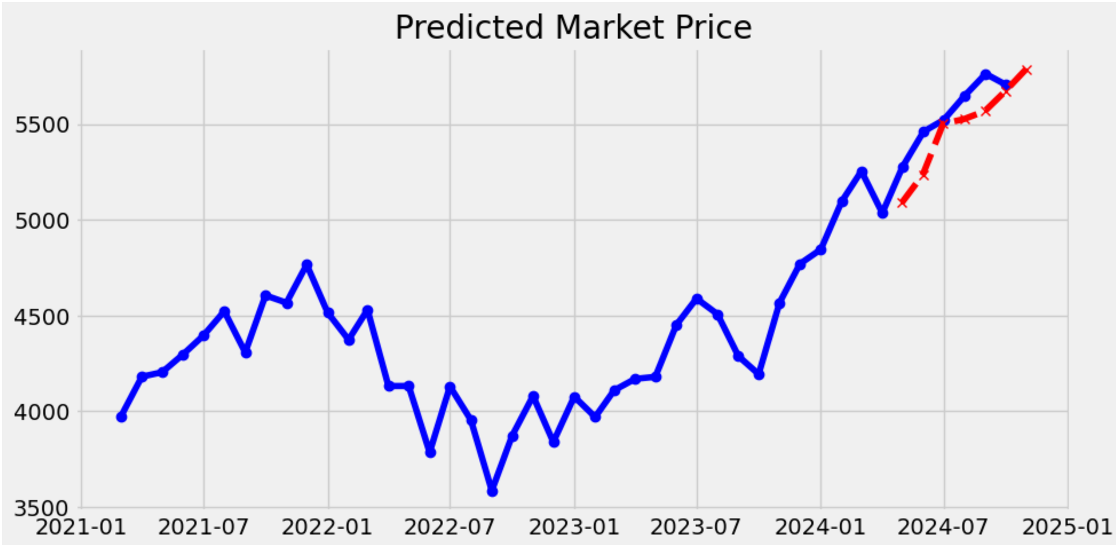
```
Epoch 63/65
1/1            0s 89ms/step - loss:
6.9684e-05 - val_loss: 0.1025
Epoch 64/65
1/1            0s 85ms/step - loss:
5.4022e-05 - val_loss: 0.1038
Epoch 65/65
1/1            0s 85ms/step - loss:
4.4995e-05 - val_loss: 0.1044
```



0.11 Final Model Predictions Visuals

[59]:
```
plt.figure(figsize=(10, 5))
plt.title('Predicted Market Price')
dates = pd.date_range(start='2024-04-01', periods=len(target.
    ↪inverse_transform(predictions)), freq='M')
plt.plot(phase_1['Date'],phase_1['SP500'], label='Actual Prices', color='blue',␣
    ↪linestyle='-', marker='o')
plt.plot(dates,target.inverse_transform(predictions), label='Predicted Prices',␣
    ↪color='red', linestyle='--', marker='x')
```

[59]:
```
[<matplotlib.lines.Line2D at 0x271cf4335d0>]
```



0.12 Visualising Hyper Parameters

[60]:
```python
best_trials = tuner.oracle.get_best_trials(num_trials=50)
```

[61]:
```python
hyperparams = []
performance = []


for trial in best_trials:

    trial_hyperparams = trial.hyperparameters.values
    hyperparams.append(trial_hyperparams)


    performance.append(trial.score)


hyperparams = np.array(hyperparams)
performance = np.array(performance)
```

[62]:
```python
hyperparams_list = [dict(trial) for trial in hyperparams]
df_hyperparams = pd.DataFrame(hyperparams_list)
```

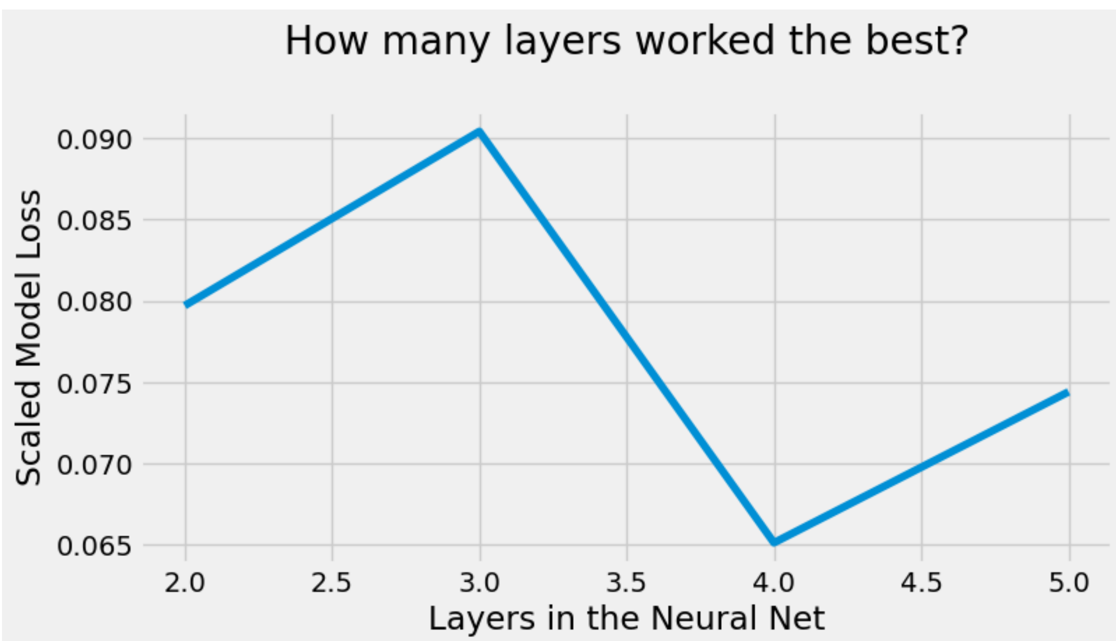
[63]:
```python
df_hyperparams['performance'] = performance
df_hyperparams.head()
```

[63]:
```
   num_layers  num_units_0  num_units_1  learning_rate  epochs  num_units_2  \
0           5           64           72       0.007592      65          112
1           5           96           56       0.007512      85          104
```

```
2         2       104       104   0.002454     60          64
3         4        48        32   0.003977     30          88
4         4        48        32   0.003977     30          88

   num_units_3  num_units_4  tuner/epochs  tuner/initial_epoch  tuner/bracket  \
0           40            8            10                    0              1
1           56           16             4                    2              3
2           56           64            30                   10              3
3           32           32            10                    4              3
4           32           32            30                   10              3

   tuner/round tuner/trial_id  performance
0            0            NaN     0.024780
1            1           0033     0.024858
2            3           0047     0.025135
3            2           0133     0.027891
4            3           0138     0.031119
```

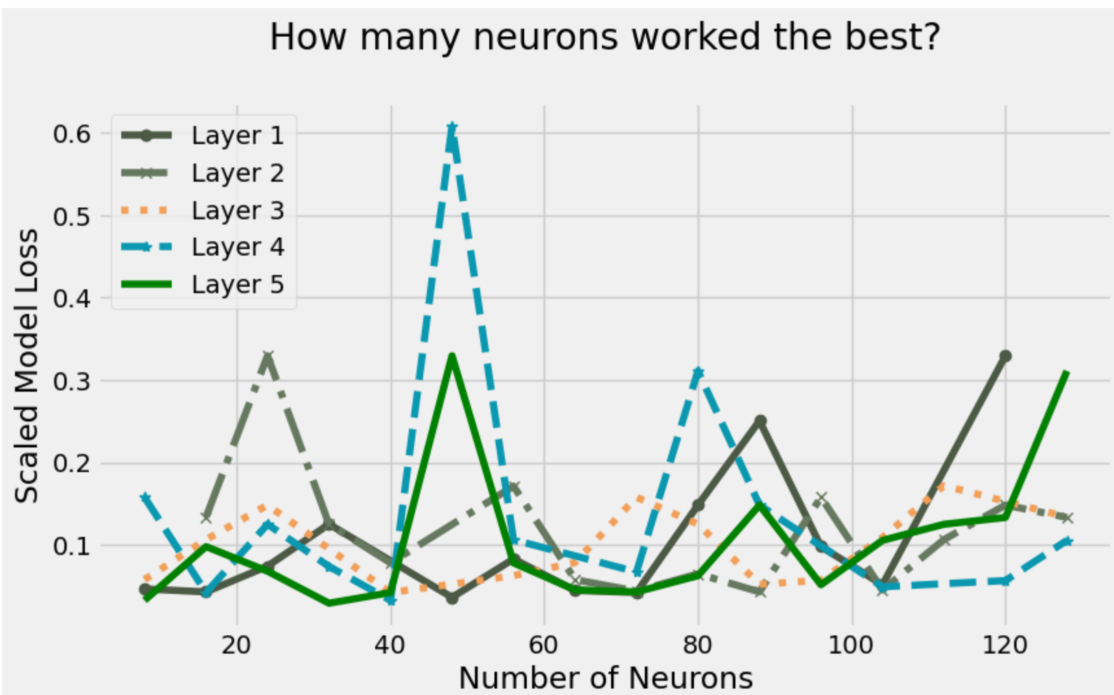
[81]:
```
plt.figure(figsize=(8,4))
plt.title('How many layers worked the best?\n')
df_hyperparams.groupby('num_layers')['performance'].median().plot()
plt.xlabel('Layers in the Neural Net')
plt.ylabel('Scaled Model Loss')
```

[81]:
```
Text(0, 0.5, 'Scaled Model Loss')
```



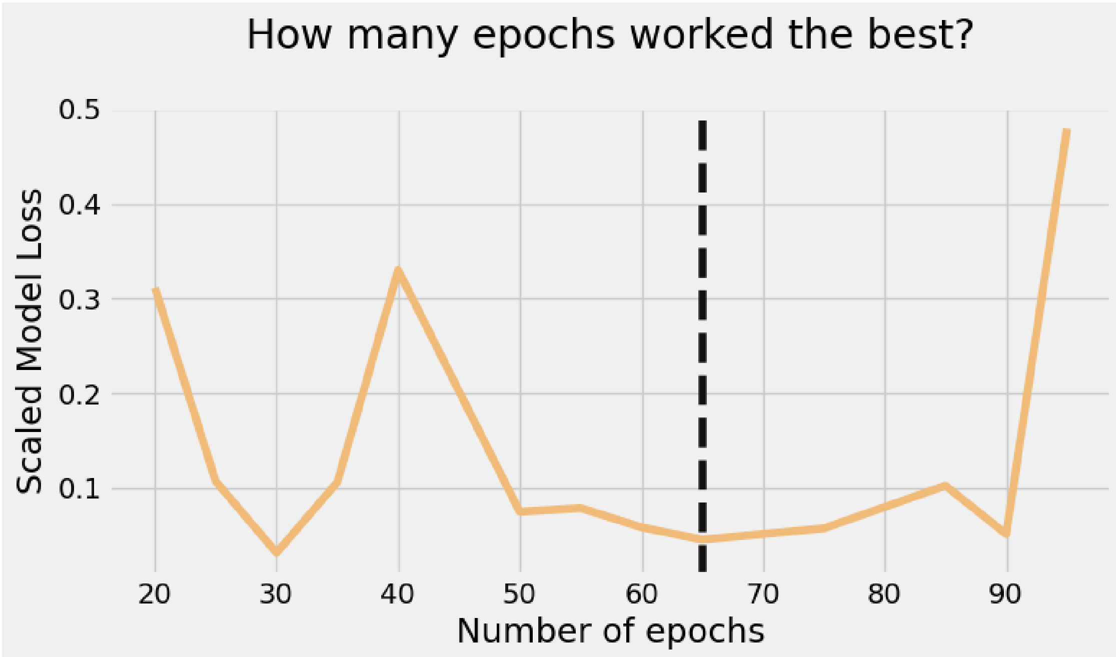
[107]:
```
plt.figure(figsize=(9,5))
plt.title('How many neurons worked the best?\n')
df_hyperparams.groupby('num_units_0')['performance'].median().
 ↪plot(color='#4B5945' ,linestyle='-', marker='o',label='Layer 1')
df_hyperparams.groupby('num_units_1')['performance'].median().
 ↪plot(color='#66785F' ,linestyle='dashdot', marker='x',label='Layer 2')
df_hyperparams.groupby('num_units_2')['performance'].median().
 ↪plot(color='#F29F58' ,linestyle=':', marker='',label='Layer 3')
df_hyperparams.groupby('num_units_3')['performance'].median().
 ↪plot(color="#0A97B0" ,linestyle='dashed', marker='*',label='Layer 4')
df_hyperparams.groupby('num_units_4')['performance'].median().
 ↪plot(color='green',label='Layer 5')
plt.legend(loc='upper left', bbox_to_anchor=(0, 1))
plt.xlabel('Number of Neurons')
plt.ylabel('Scaled Model Loss')
```

[107]:
```
Text(0, 0.5, 'Scaled Model Loss')
```



[116]:
```
plt.figure(figsize=(8,4))
plt.title('How many epochs worked the best?\n')
df_hyperparams.groupby('epochs')['performance'].median().plot(color='#F0BB78')
plt.axvline(x=65,linestyle='dashed',color='#131010')
plt.xlabel('Number of epochs')
plt.ylabel('Scaled Model Loss')
```

[116]: Text(0, 0.5, 'Scaled Model Loss')



0.13 Model Dump

[64]:
```
import pickle
#pickle.dump(model, open('Best_Model.sav', 'wb'))
```