Stock Market Clustering

Live Stock Market Data Analysis From Yahoo Finance (K-Means Clustering)

In this project, we will be extracting live Stock Market data from Yahoo Finance, find similarities amongst various companies using their stock market prices and then group them into different clusters using K-Means algorithm (Unsupervised Learning algorithm).

pandas-datareader will be used to create a DataFrame from the Yahoo Finance website.

yfinance will be also used to to an API problem currently with pandas-datereader

In [1]:
```
#! pip install pandas-datareader
#! pip install yfinance --upgrade --no-cache-dir
```

In [2]:
```
from pandas_datareader import data as pdr
import yfinance as yf
yf.pdr_override()
import pandas as pd
import numpy as np
import datetime
import matplotlib.pyplot as plt
pd.options.display.max_columns = 30
#import warnings
#warnings.simplefilter('ignore')
```

In [3]:
```
companies_dict = {'Amazon': 'AMZN',
                  'Apple': 'AAPL',
                  'Walgreens': 'WBA',
                  'Northrop Grumman': 'NOC',
                  'Boeing': 'BA',
                  'Lockheed Martin': 'LMT',
                  'McDonalds': 'MCD',
                  'Intel': 'INTC',
                  'IBM': 'IBM',
                  'Texas Instruments': 'TXN',
                  'MasterCard': 'MA',
                  'Visa': 'V',
                  'Microsoft': 'MSFT',
                  'General Electrics': 'GE',
                  'American Express': 'AXP',
                  'Pepsi': 'PEP',
                  'Coca-Cola': 'KO',
                  'Johnson & Johnson': 'JNJ',
                  'Toyoya': 'TM', 'Honda': 'HMC',
                  'Sony': 'SONY',
                  'Exxon': 'XOM',
                  'Chevron': 'CVX',
                  'Ford': 'F',
```

```
                     'Bank of America': 'BAC',
                     'JP Morgan Chase': 'JPM',
                     'HP': 'HPQ',
                     'Facebook': 'FB',
                     'Twitter': 'TWTR',
                     'Alphabet': 'GOOGL'}
names = list(companies_dict.keys())
codes = list(companies_dict.values())
```

In [4]:

```
data_source = 'yahoo'
start_date = str(datetime.date.today() - datetime.timedelta(days=365*5))
end_date = str(datetime.date.today())
print(f"Start Date: {start_date}")
print(f"End Date: {end_date}")
```

```
Start Date: 2016-07-27
End Date: 2021-07-26
```

In [5]:

```
df = pdr.get_data_yahoo(list(companies_dict.values()), start=start_date, end=e
df
```

```
[*********************100%***********************]  30 of 30 completed
```

Out[5]:

	AAPL	AMZN	AXP	BA	BAC	CVX	F	F
Date								
2016-07-27	24.043583	736.669983	59.814663	124.065628	13.233034	82.196663	11.135532	123.33999
2016-07-28	24.368214	752.609985	59.999882	121.373680	13.278264	81.818871	10.226344	125.00000
2016-07-29	24.337851	758.809998	59.694275	121.966843	13.106404	82.373528	10.186113	123.94000
2016-08-01	24.767578	767.739990	59.388668	121.556206	12.961682	79.664703	10.041287	124.30999
2016-08-02	24.400913	760.580017	58.777466	120.087067	12.780781	80.050514	9.606809	123.08999
...	...	...	...	...	...	...	...	.
2021-07-19	142.449997	3549.590088	162.809998	206.990005	36.930000	95.959999	13.280000	336.95001
2021-07-20	146.149994	3573.189941	168.869995	217.149994	37.689999	96.529999	13.910000	341.66000
2021-07-21	145.399994	3585.199951	172.509995	222.539993	38.459999	99.820000	14.190000	346.23001
2021-07-22	146.800003	3638.030029	170.899994	220.869995	37.959999	98.820000	13.910000	351.19000
2021-07-23	148.559998	3656.639893	173.179993	221.520004	37.700001	98.860001	13.820000	369.79000

1257 rows × 180 columns

In [6]:

```
df.isna().sum().sort_values(ascending=False)[:15]
```

Out[6]:
```
Adj Close  AAPL    0
Low        SONY    0
           TWTR    0
           TXN     0
           V       0
           WBA     0
           XOM     0
Open       AAPL    0
           AMZN    0
           AXP     0
           BA      0
           BAC     0
           CVX     0
           F       0
           FB      0
dtype: int64
```

movements is the difference of closing and opening prices of a particular day. Positive movement implies buying the stock while negative movement implies selling.

In [7]:
```
stock_open = df['Open'].T
stock_close = df['Close'].T
movements = stock_close - stock_open
```

In [8]:
```
print(f"{'='*65}\n{'*'*15} stock_open {'*'*15}\n{stock_open.iloc[:3,:5]}\n{'='
print(f"{'='*65}\n{'*'*15} stock_close {'*'*15}\n{stock_close.iloc[:3,:5]}\n{'
print(f"{'='*65}\n{'*'*15} movements {'*'*15}\n{movements.iloc[:3,:5]}\n{'='*6
```

```
=================================================================
*************** stock_open ***************
Date  2016-07-27  2016-07-28  2016-07-29  2016-08-01  2016-08-02
AAPL   26.067499   25.707500   26.047501   26.102501   26.512501
AMZN  737.969971  745.979980  765.000000  759.869995  763.809998
AXP    64.269997   64.360001   64.699997   64.489998   63.959999
=================================================================
=================================================================
*************** stock_close ***************
Date  2016-07-27  2016-07-28  2016-07-29  2016-08-01  2016-08-02
AAPL   25.737499   26.084999   26.052500   26.512501   26.120001
AMZN  736.669983  752.609985  758.809998  767.739990  760.580017
AXP    64.589996   64.790001   64.459999   64.129997   63.470001
=================================================================
=================================================================
*************** movements ***************
Date  2016-07-27  2016-07-28  2016-07-29  2016-08-01  2016-08-02
AAPL   -0.330000    0.377499    0.004999    0.410000   -0.392500
AMZN   -1.299988    6.630005   -6.190002    7.869995   -3.229980
AXP     0.320000    0.430000   -0.239998   -0.360001   -0.489998
=================================================================
```

sum_of_movements is the sum of differences of closing and opening prices for all days.

In [9]:
```
sum_of_movements = np.sum(movements, axis=1)
```

In [10]:
```
for i in range(len(companies_dict)):
    print(f"{'='*30}\nCompany: {df['High'].columns[i]}\nSum of Movements(5 Yea
```

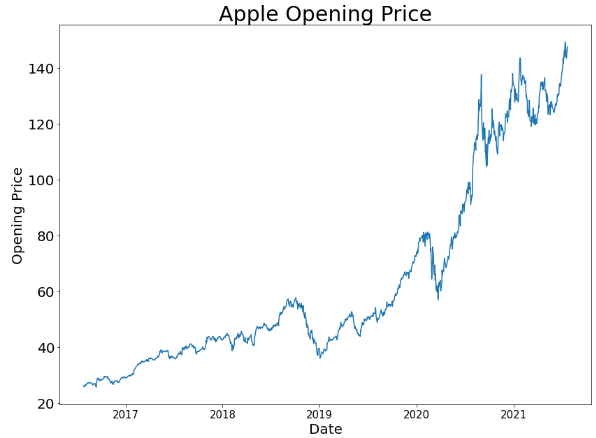
```
==============================
Company: AAPL
Sum of Movements(5 Years):
55.34481239318848
==============================
==============================
Company: AMZN
Sum of Movements(5 Years):
-843.2141723632812
==============================
==============================
Company: AXP
Sum of Movements(5 Years):
16.770206451416016
==============================
==============================
Company: BA
Sum of Movements(5 Years):
-189.9903793334961
==============================
==============================
Company: BAC
Sum of Movements(5 Years):
5.869987487792969
==============================
==============================
Company: CVX
Sum of Movements(5 Years):
-65.4798355102539
==============================
==============================
Company: F
Sum of Movements(5 Years):
-12.000017166137695
==============================
==============================
Company: FB
Sum of Movements(5 Years):
126.07007598876953
==============================
==============================
Company: GE
Sum of Movements(5 Years):
-24.255778789520264
==============================
==============================
Company: GOOGL
Sum of Movements(5 Years):
948.3408203125
==============================
==============================
Company: HMC
Sum of Movements(5 Years):
-10.510007858276367
==============================
==============================
Company: HPQ
Sum of Movements(5 Years):
13.349958419799805
==============================
```

```
===============================
Company: IBM
Sum of Movements(5 Years):
-39.17976379394531
===============================
===============================
Company: INTC
Sum of Movements(5 Years):
19.779972076416016
===============================
===============================
Company: JNJ
Sum of Movements(5 Years):
-2.5099868774414062
===============================
===============================
Company: JPM
Sum of Movements(5 Years):
19.959938049316406
===============================
===============================
Company: KO
Sum of Movements(5 Years):
-10.010047912597656
===============================
===============================
Company: LMT
Sum of Movements(5 Years):
-151.5192108154297
===============================
===============================
Company: MA
Sum of Movements(5 Years):
-17.349441528320312
===============================
===============================
Company: MCD
Sum of Movements(5 Years):
14.61029052734375
===============================
===============================
Company: MSFT
Sum of Movements(5 Years):
76.04998779296875
===============================
===============================
Company: NOC
Sum of Movements(5 Years):
-78.18048095703125
===============================
===============================
Company: PEP
Sum of Movements(5 Years):
23.16010284423828
===============================
===============================
Company: SONY
Sum of Movements(5 Years):
-13.520116806030273
===============================
```

```
==============================
Company: TM
Sum of Movements(5 Years):
-35.57001495361328
==============================
==============================
Company: TWTR
Sum of Movements(5 Years):
19.449939727783203
==============================
==============================
Company: TXN
Sum of Movements(5 Years):
55.400001525878906
==============================
==============================
Company: V
Sum of Movements(5 Years):
-1.1900711059570312
==============================
==============================
Company: WBA
Sum of Movements(5 Years):
-34.16001892089844
==============================
==============================
Company: XOM
Sum of Movements(5 Years):
-53.20004653930664
==============================
```

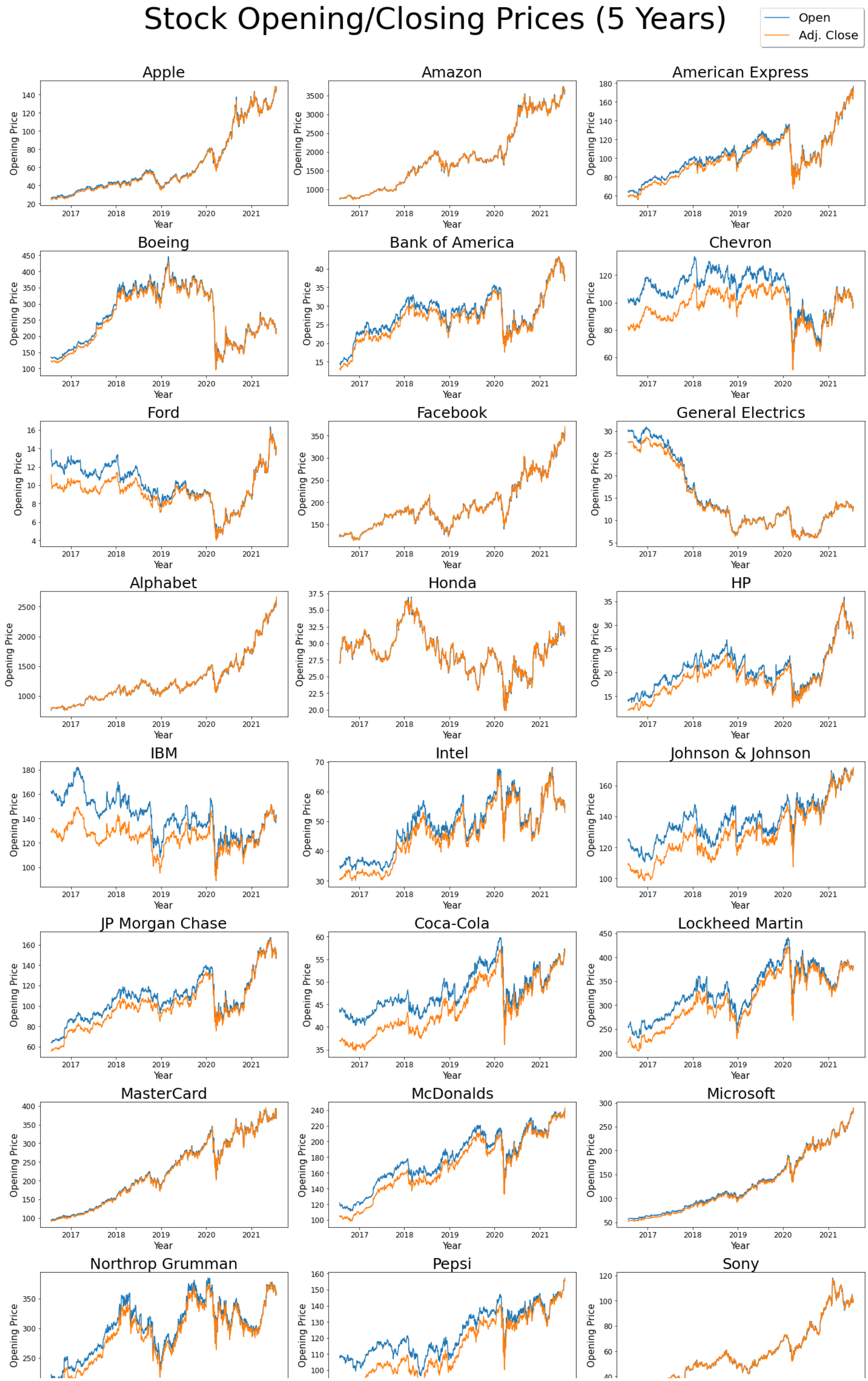
In [11]:

```
fig = plt.figure(figsize=(30,10))
fig.add_subplot(1,2,1)
plt.plot(df['Open']['AMZN'])
plt.title('Amazon Opening Price', fontsize=30)
plt.xticks(fontsize=15)
plt.yticks(fontsize=20)
plt.xlabel('Date', fontsize=20)
plt.ylabel('Opening Price', fontsize=20)

fig.add_subplot(1,2,2)
plt.plot(df['Open']['AAPL'])
plt.title('Apple Opening Price', fontsize=30)
plt.xticks(fontsize=15)
plt.yticks(fontsize=20)
plt.xlabel('Date', fontsize=20)
plt.ylabel('Opening Price', fontsize=20)

plt.show()
```



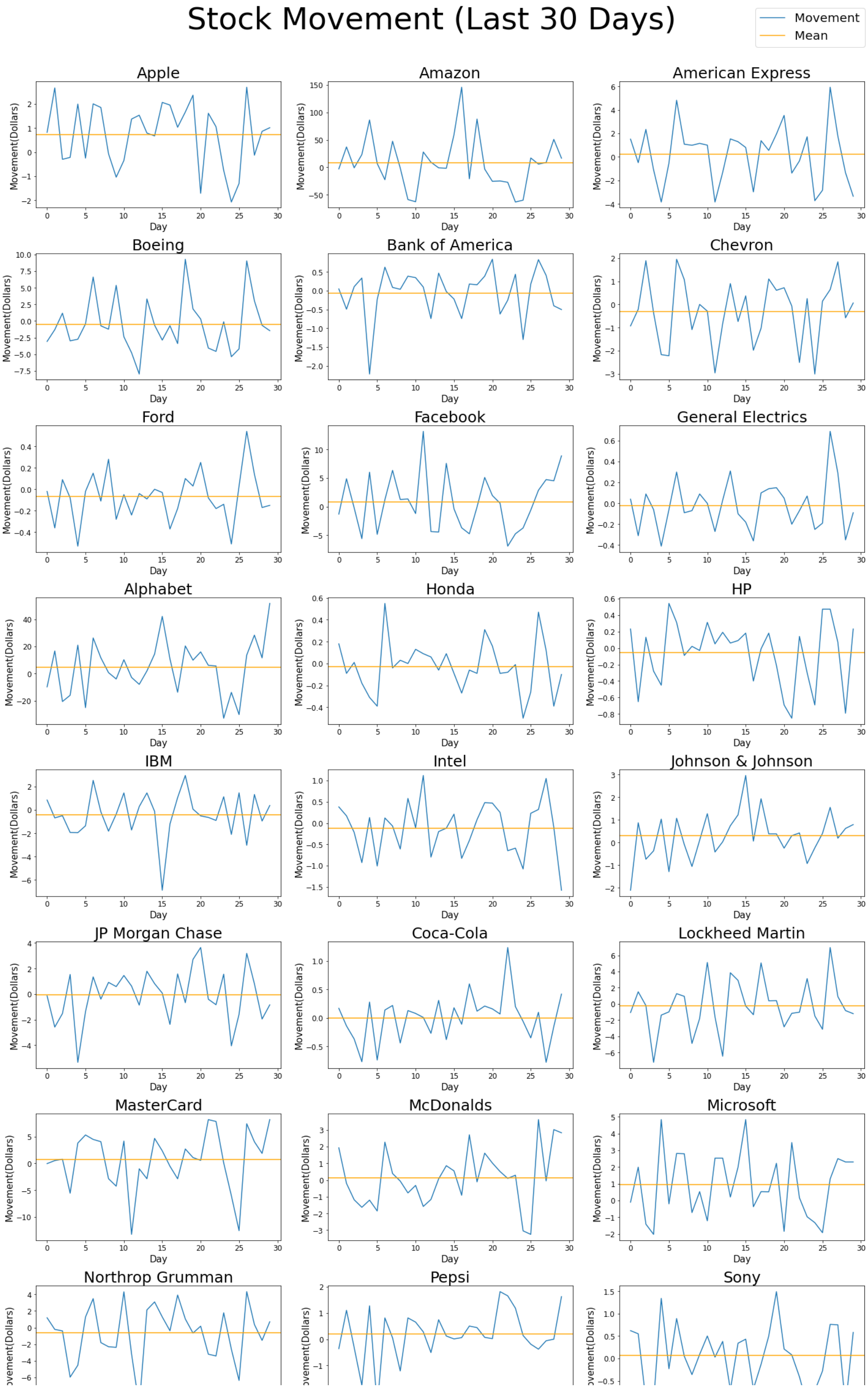
Plotting Opening/Closing Prices for Each Stock (5 Years)

In [12]:
```
fig, ax = plt.subplots(10,3,figsize=(20,40))
i = 0
for x in range(10):
    for y in range(3):
        ax[x,y].plot(df['Open'].iloc[:,i])
        ax[x,y].plot(df['Adj Close'].iloc[:,i])
        loc = codes.index(df['Open'].columns[i])
        ax[x,y].set_title(f'{names[loc]}', fontsize=25)
        ax[x,y].tick_params(axis='both', labelsize=12)
        ax[x,y].set_xlabel('Year', fontsize=15)
        ax[x,y].set_ylabel('Opening Price', fontsize=15)
        i +=1
fig.suptitle('Stock Opening/Closing Prices (5 Years)', fontsize=50, x=0.5, y=1
plt.tight_layout()
fig.legend(['Open', 'Adj. Close'], fontsize=20, shadow=True, loc='upper right'
plt.show()
```


Stock Opening/Closing Prices (5 Years)
Open
Adj. Close
Apple
Amazon
American Express
Boeing
Bank of America
Chevron
Ford
Facebook
General Electrics
Alphabet
Honda
HP
IBM
Intel
Johnson & Johnson
JP Morgan Chase
Coca-Cola
Lockheed Martin
MasterCard
McDonalds
Microsoft
Northrop Grumman
Pepsi
Sony
Opening Price
Year

Plotting Stock Movement For the Last 30 Days

In [13]:
```
fig, ax = plt.subplots(10,3,figsize=(20,40))
i = 0
for x in range(10):
    for y in range(3):
        ax[x,y].plot(movements.loc[movements.index[i]][-30:].reset_index(drop=
        ax[x,y].axhline(movements.loc[movements.index[i]][-30:].reset_index(dr
        loc = codes.index(df['Open'].columns[i])
        ax[x,y].set_title(f'{names[loc]}', fontsize=25)
        ax[x,y].tick_params(axis='both', labelsize=12)
        ax[x,y].set_xlabel('Day', fontsize=15)
        ax[x,y].set_ylabel('Movement(Dollars)', fontsize=15)
        i +=1
fig.legend(['Movement', 'Mean'],fontsize=20, loc='upper right')
fig.suptitle('Stock Movement (Last 30 Days)', fontsize=50, x=0.5, y=1.01)
plt.tight_layout()
plt.show()
```


Stock Movement (Last 30 Days)
Movement
Mean
Apple
Amazon
American Express
Boeing
Bank of America
Chevron
Ford
Facebook
General Electrics
Alphabet
Honda
HP
IBM
Intel
Johnson & Johnson
JP Morgan Chase
Coca-Cola
Lockheed Martin
MasterCard
McDonalds
Microsoft
Northrop Grumman
Pepsi
Sony
Movement(Dollars)
Day

A positive movement is desirable because it suggests the price has increased during the day.

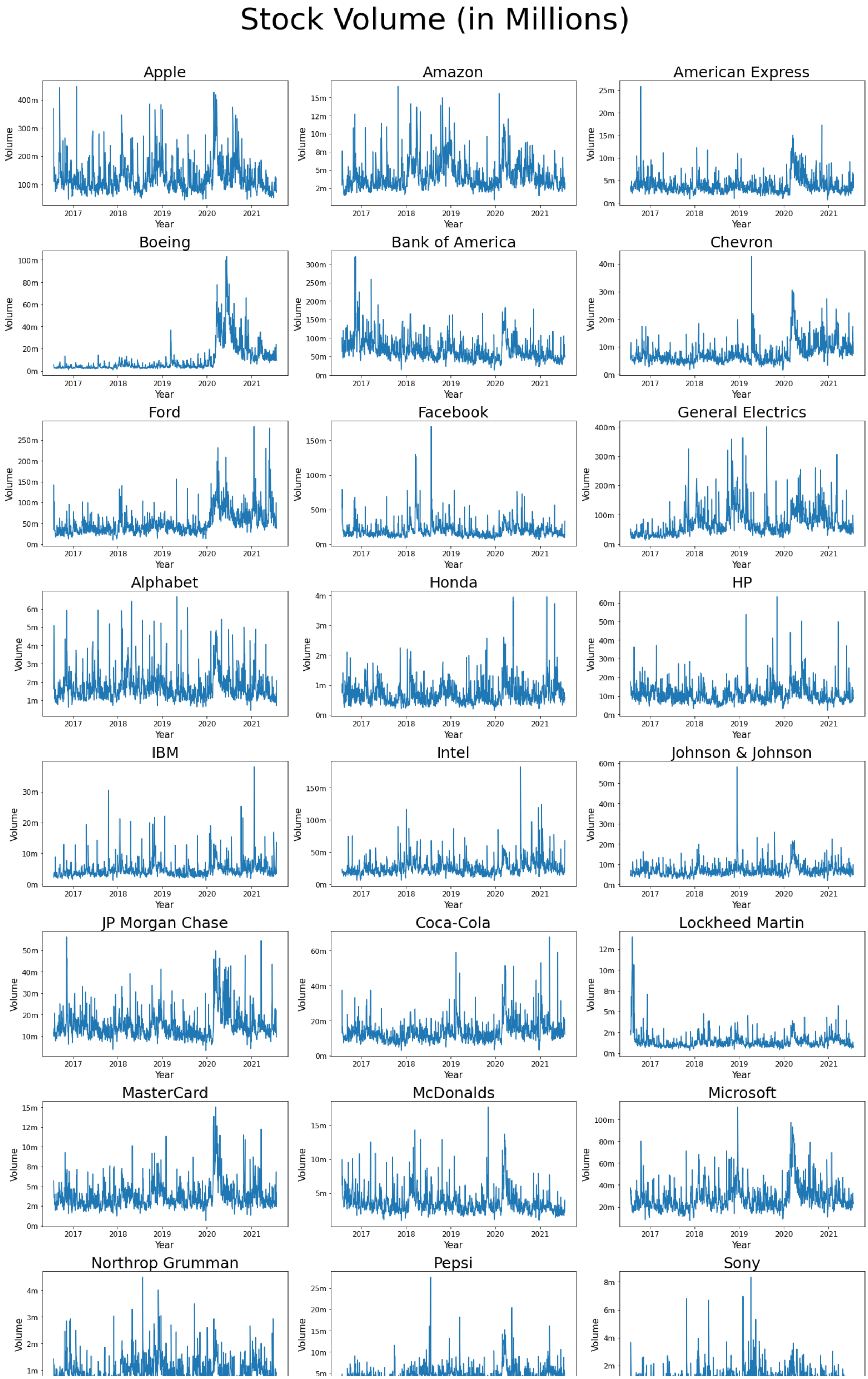
Plotting Each Stock's Volume Over 5 Years

Volume refers to the total number of shares that are actually traded (bought and sold) during the trading day or specified set period of time. If only five transactions occur in a day, the volume for that day is set at five.

While the same shares may be traded back and forth multiple times, the volume is counted on each transaction. Therefore if 500 shares of XYZ were bought, then sold, then re-bought and then re-sold again resulting in four tickets, then the volume would register as 2,000 shares, even though the same 500 shares may have been in play multiple times.

In [14]:
```
import matplotlib.ticker as mticker
fig, ax = plt.subplots(10,3,figsize=(20,40))
i = 0
for x in range(10):
    for y in range(3):
        ax[x,y].plot(df['Volume'].iloc[:,i])
        loc = codes.index(df['Open'].columns[i])
        ax[x,y].set_title(f'{names[loc]}', fontsize=25)
        ax[x,y].tick_params(axis='both', labelsize=12)
        ax[x,y].ticklabel_format(axis='y',style='plain')
        ax[x,y].set_xlabel('Year', fontsize=15)
        ax[x,y].set_ylabel('Volume', fontsize=15)
        ticks_loc = ax[x,y].get_yticks()
        ax[x,y].yaxis.set_major_locator(mticker.FixedLocator(ticks_loc))
        ax[x,y].set_yticklabels(['{:,.0f}'.format(x) + 'm' for x in ticks_loc
        i +=1
fig.suptitle('Stock Volume (in Millions)', fontsize=50, x=0.5, y=1.01)
```

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

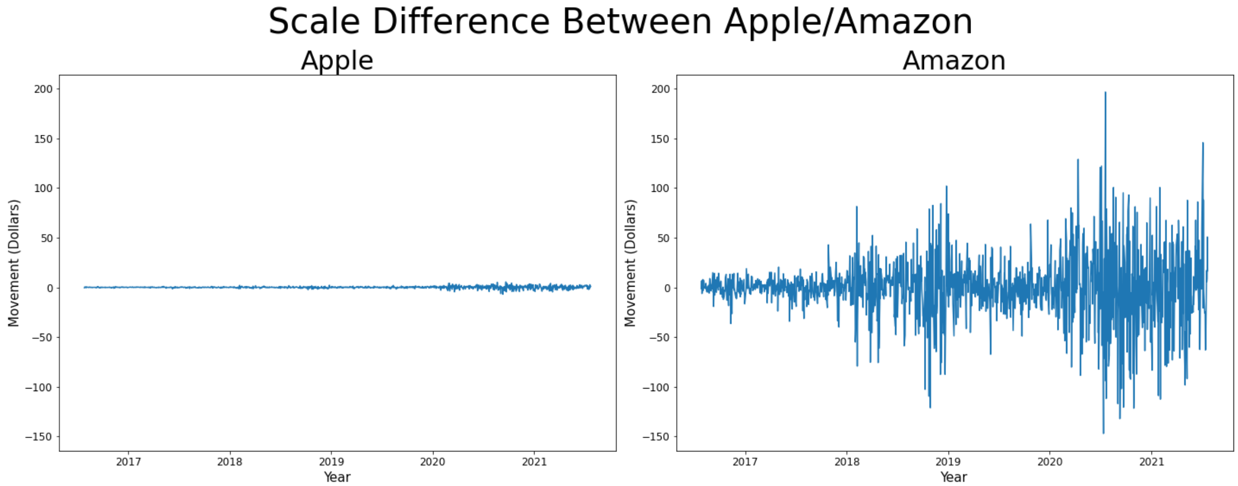

Stock Volume (in Millions)
Apple
Amazon
American Express
Boeing
Bank of America
Chevron
Ford
Facebook
General Electrics
Alphabet
Honda
HP
IBM
Intel
Johnson & Johnson
JP Morgan Chase
Coca-Cola
Lockheed Martin
MasterCard
McDonalds
Microsoft
Northrop Grumman
Pepsi
Sony
Volume
Year

The peaks suggest there are high volumes of stocks traded on certain days. This may be due to several situations which might affect the image of the company positively or negatively.

Normalization

Stock prices of Amazon and Apple have different scales. There is a significant difference in the rate of change of units in stock price, and in that case we need to normalize our data in order to have consistency for further analysis.

In [15]:
```
fig = plt.figure(figsize=(20,8))
ax = fig.add_subplot(121)
ax.plot(movements.loc['AAPL'])
ax.set_title('Apple', fontsize=30)
ax.tick_params(labelsize=12)
ax.set_xlabel('Year', fontsize=15)
ax.set_ylabel('Movement (Dollars)', fontsize=15)
ax2 = fig.add_subplot(122, sharey=ax)
ax2.plot(movements.loc['AMZN'])
ax2.set_title('Amazon', fontsize=30)
ax2.tick_params(labelsize=12)
ax2.set_xlabel('Year', fontsize=15)
ax2.set_ylabel('Movement (Dollars)', fontsize=15)
fig.suptitle('Scale Difference Between Apple/Amazon', fontsize=40)
plt.tight_layout()
plt.show()
```



Normalizer(): Normalizes each sample (row) using that row's L2 norm by default.

$$\|\mathbf{x}\|_2 = \left(\sum_{i=1}^{N} |x_i|^2 \right)^{1/2} = \sqrt{x_1^2 + x_2^2 + \ldots + x_N^2}$$

example below:

In [16]:
```
x = np.array([[4, 1, 2, 2], [1, 3, 9, 3], [5, 7, 5, 1]])
print(f"Original Array:\n{x}")
print(f"Square Each Value:\n{x**2}")
print(f"Sum Each Vector(Row):\n{np.sum(x**2, axis=1)}")
print(f"Square Root of Each Sum(L2 Norm):\n{np.sqrt(np.sum(x**2, axis=1))}")
print(f"Normalize by Dividing Each Vector(Row) With Its L2 Norm:")
for i in range(3):
    print(x[i] / np.sqrt(np.sum(x**2, axis=1))[i])
```

```
Original Array:
[[4 1 2 2]
 [1 3 9 3]
 [5 7 5 1]]
Square Each Value:
[[16  1  4  4]
 [ 1  9 81  9]
 [25 49 25  1]]
Sum Each Vector(Row):
[ 25 100 100]
Square Root of Each Sum(L2 Norm):
[ 5. 10. 10.]
Normalize by Dividing Each Vector(Row) With Its L2 Norm:
[0.8 0.2 0.4 0.4]
[0.1 0.3 0.9 0.3]
[0.5 0.7 0.5 0.1]
```

Measures of Center/Spread for Normalizer

In [17]:
```
from sklearn.preprocessing import Normalizer
normalizer = Normalizer()
norm_movements = pd.DataFrame(normalizer.fit_transform(movements), columns=mov
norm_movements.T.describe()[1:]
```

Out[17]:

	AAPL	AMZN	AXP	BA	BAC	CVX	F	FB	(
mean	0.001123	-0.000591	0.000259	-0.000881	0.000326	-0.001077	-0.001717	0.000914	-0.0023
std	0.028194	0.028210	0.028215	0.028203	0.028215	0.028196	0.028164	0.028202	0.0281
min	-0.175265	-0.129463	-0.135908	-0.197650	-0.154892	-0.230826	-0.116905	-0.150878	-0.1433
25%	-0.007590	-0.010907	-0.011245	-0.012998	-0.013257	-0.015926	-0.016187	-0.011935	-0.0180
50%	0.001276	-0.000079	0.000969	-0.000583	0.000698	-0.001034	-0.001799	0.000456	-0.0023
75%	0.010587	0.010054	0.012408	0.010900	0.014652	0.014685	0.012590	0.013849	0.0127
max	0.141335	0.173110	0.204348	0.167516	0.137449	0.125754	0.151078	0.181400	0.1953

Measures of Center/Spread for StandardScaler (for comparison)

In [18]:
```
from sklearn.preprocessing import StandardScaler
pd.DataFrame(StandardScaler().fit_transform(movements.T), columns=movements.T.
```

Out[18]:

	AAPL	AMZN	AXP	BA	BAC	CVX	F	FB	(
mean	0.000000	-0.000000	-0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0000
std	1.000398	1.000398	1.000398	1.000398	1.000398	1.000398	1.000398	1.000398	1.0003
min	-6.258678	-4.570072	-4.827899	-6.979686	-5.503498	-8.151490	-4.091496	-5.384476	-5.0162
25%	-0.309158	-0.365854	-0.407869	-0.429805	-0.481585	-0.526831	-0.513973	-0.455795	-0.5589
50%	0.005404	0.018133	0.025201	0.010573	0.013191	0.001539	-0.002898	-0.016254	0.0002
75%	0.335809	0.377460	0.430770	0.417873	0.507963	0.559257	0.508176	0.458839	0.5348
max	4.975030	6.159778	7.236112	5.973285	4.861926	4.500005	5.427278	6.402343	7.0316

Measures of Center/Spread for MinMaxScaler (for comparison)

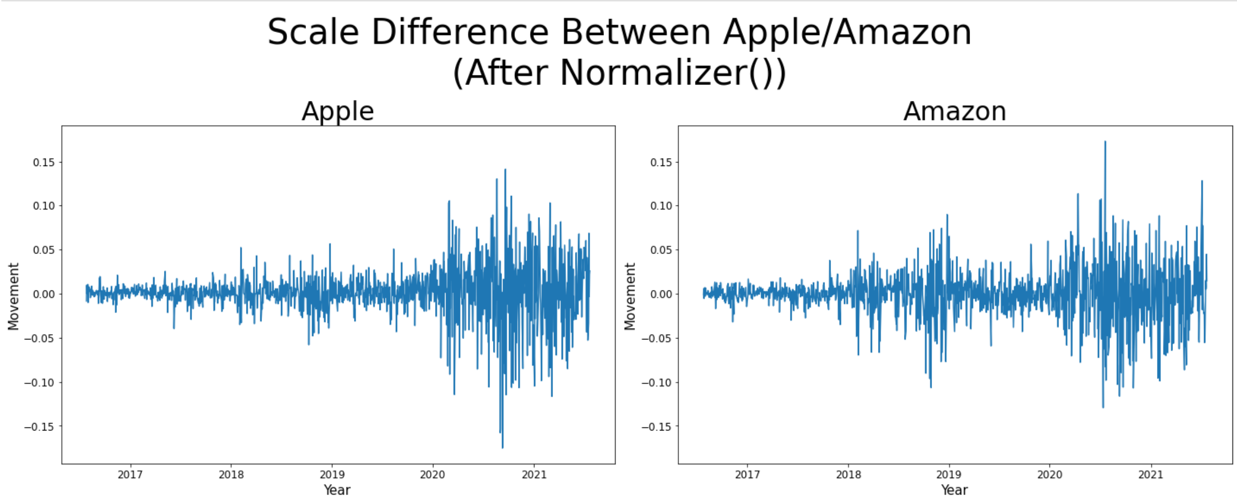
In [19]:
```
from sklearn.preprocessing import MinMaxScaler
pd.DataFrame(MinMaxScaler().fit_transform(movements.T), columns=movements.T.co
```

Out[19]:

	AAPL	AMZN	AXP	BA	BAC	CVX	F	FB	GE	C
mean	0.557134	0.425921	0.400190	0.538848	0.530948	0.644310	0.429834	0.456822	0.416359	0.
std	0.089053	0.093235	0.082924	0.077233	0.096513	0.079074	0.105097	0.084874	0.083035	0.
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.
25%	0.529613	0.391825	0.366381	0.505666	0.484487	0.602669	0.375839	0.418152	0.369965	0.
50%	0.557615	0.427611	0.402279	0.539665	0.532220	0.644432	0.429530	0.455443	0.416382	0.
75%	0.587027	0.461100	0.435897	0.571109	0.579953	0.688515	0.483221	0.495750	0.460751	0.
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.

In [20]:

```
fig = plt.figure(figsize=(20,8))
ax = fig.add_subplot(121)
ax.plot(norm_movements.loc['AAPL'])
ax.set_title('Apple', fontsize=30)
ax.tick_params(labelsize=12)
ax.set_xlabel('Year', fontsize=15)
ax.set_ylabel('Movement', fontsize=15)
ax2 = fig.add_subplot(122, sharey=ax)
ax2.plot(norm_movements.loc['AMZN'])
ax2.set_title('Amazon', fontsize=30)
ax2.tick_params(labelsize=12)
ax2.set_xlabel('Year', fontsize=15)
ax2.set_ylabel('Movement', fontsize=15)
fig.suptitle('Scale Difference Between Apple/Amazon\n(After Normalizer())', fo
plt.tight_layout()
plt.show()
```



Now we have normalized movements for Amazon and Apple.

KMeans Classifier: Finding the k value

In [21]:

```
from sklearn.cluster import KMeans
sum_of_squared_distances = []
for i in range(1,16):
    kmeans = KMeans(n_clusters=i, max_iter=1000, random_state=1)
```

```
    kmeans = kmeans.fit(norm_movements)
    sum_of_squared_distances.append(kmeans.inertia_)
```

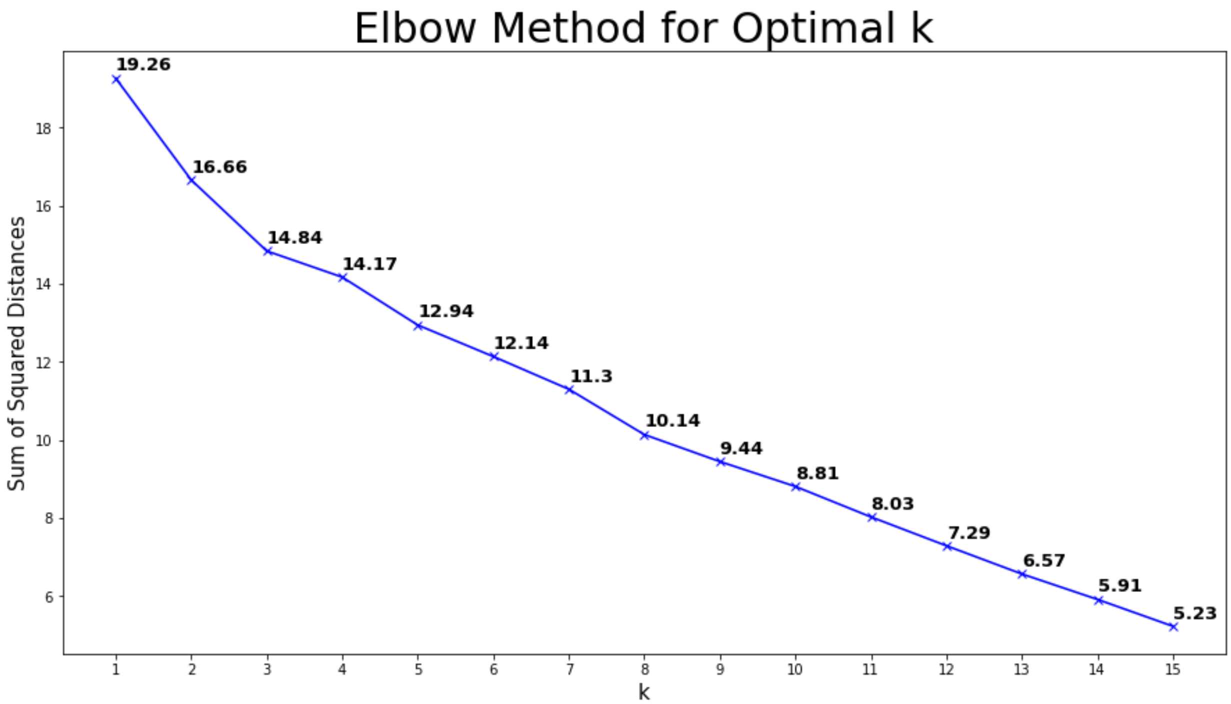
```
D:\anaconda3\envs\my-env\lib\site-packages\sklearn\cluster\_kmeans.py:881: UserWarning: KMeans is known to have a memory leak on Windows with MKL, when there are less chunks than available threads. You can avoid it by setting the environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```

In [22]:

```
# Sum of Squared Distances
```

In [23]:

```
plt.figure(figsize=(15,8))
plt.plot(np.arange(1,16), sum_of_squared_distances, 'bx-')
plt.xlabel('k', fontsize=15)
plt.ylabel('Sum of Squared Distances', fontsize=15)
plt.title('Elbow Method for Optimal k', fontsize=30)
plt.xticks(np.arange(1,16))
for i in range(15):
    plt.text(i+1, sum_of_squared_distances[i]+0.2, np.round(sum_of_squared_dis
plt.show()
```



Creating a Pipeline

In [24]:

```
from sklearn.pipeline import make_pipeline
from sklearn.cluster import KMeans

# Initialize a normalizer
normalizer = Normalizer()

# Create KMeans model
kmeans = KMeans(n_clusters=5, max_iter=1000, random_state=1)

# Make a pipeline combining our normalizer and KMeans model
pipeline = make_pipeline(normalizer, kmeans)
```

```
# Fit pipeline to daily stock movements created earlier
pipeline.fit(movements)
predictions = pipeline.predict(movements)
predictions
```

Out[24]:
```
array([0, 0, 3, 3, 3, 3, 3, 0, 2, 0, 2, 2, 2, 0, 4, 3, 4, 1, 0, 0, 0, 1,
       4, 0, 2, 0, 0, 0, 2, 3])
```

Creating a DataFrame of the Resultant Clusters

In [25]:
```
clusters = pd.DataFrame({'Code': movements.index, 'Cluster': predictions})
clusters = clusters.merge(pd.DataFrame({'Company': companies_dict.keys(), 'Cod
clusters = clusters[['Company', 'Cluster']].sort_values(by='Cluster')
clusters
```

Out[25]:

Code	Company	Cluster
AAPL	Apple	0
MSFT	Microsoft	0
INTC	Intel	0
MA	MasterCard	0
FB	Facebook	0
SONY	Sony	0
GOOGL	Alphabet	0
TWTR	Twitter	0
TXN	Texas Instruments	0
V	Visa	0
AMZN	Amazon	0
MCD	McDonalds	0
NOC	Northrop Grumman	1
LMT	Lockheed Martin	1
GE	General Electrics	2
HMC	Honda	2
HPQ	HP	2
IBM	IBM	2
WBA	Walgreens	2
TM	Toyoya	2
XOM	Exxon	3
JPM	JP Morgan Chase	3
F	Ford	3
CVX	Chevron	3
BAC	Bank of America	3
BA	Boeing	3
AXP	American Express	3
PEP	Pepsi	4
KO	Coca-Cola	4
JNJ	Johnson & Johnson	4

Dimensionality Reduction: Principal Component Analysis(PCA)

In [26]:
```
from sklearn.decomposition import PCA

normalizer = Normalizer()
pca = PCA(n_components=2)
kmeans = KMeans(n_clusters=5, max_iter=1000, random_state=1)

pipeline = make_pipeline(normalizer, pca, kmeans)
pipeline.fit(movements)
predictions = pipeline.predict(movements)
predictions
```

Out[26]:
```
array([4, 4, 3, 0, 3, 3, 3, 4, 3, 4, 0, 0, 0, 1, 2, 3, 2, 2, 1, 0, 4, 2,
       2, 1, 0, 4, 1, 1, 0, 3])
```

In [27]:
```
clusters_pca = pd.DataFrame({'Code': movements.index, 'Cluster': predictions})
clusters_pca = clusters_pca.merge(pd.DataFrame({'Company': companies_dict.keys
clusters_pca = clusters_pca[['Company', 'Cluster']].sort_values(by='Cluster')
clusters_pca
```

Out[27]:

Code	Company	Cluster
MCD	McDonalds	0
BA	Boeing	0
TM	Toyoya	0
WBA	Walgreens	0
IBM	IBM	0
HMC	Honda	0
HPQ	HP	0
V	Visa	1
TXN	Texas Instruments	1
SONY	Sony	1
INTC	Intel	1
MA	MasterCard	1
JNJ	Johnson & Johnson	2
NOC	Northrop Grumman	2
PEP	Pepsi	2
LMT	Lockheed Martin	2
KO	Coca-Cola	2
XOM	Exxon	3
GE	General Electrics	3
F	Ford	3
CVX	Chevron	3
BAC	Bank of America	3
AXP	American Express	3
JPM	JP Morgan Chase	3
GOOGL	Alphabet	4
MSFT	Microsoft	4
FB	Facebook	4
TWTR	Twitter	4
AMZN	Amazon	4
AAPL	Apple	4

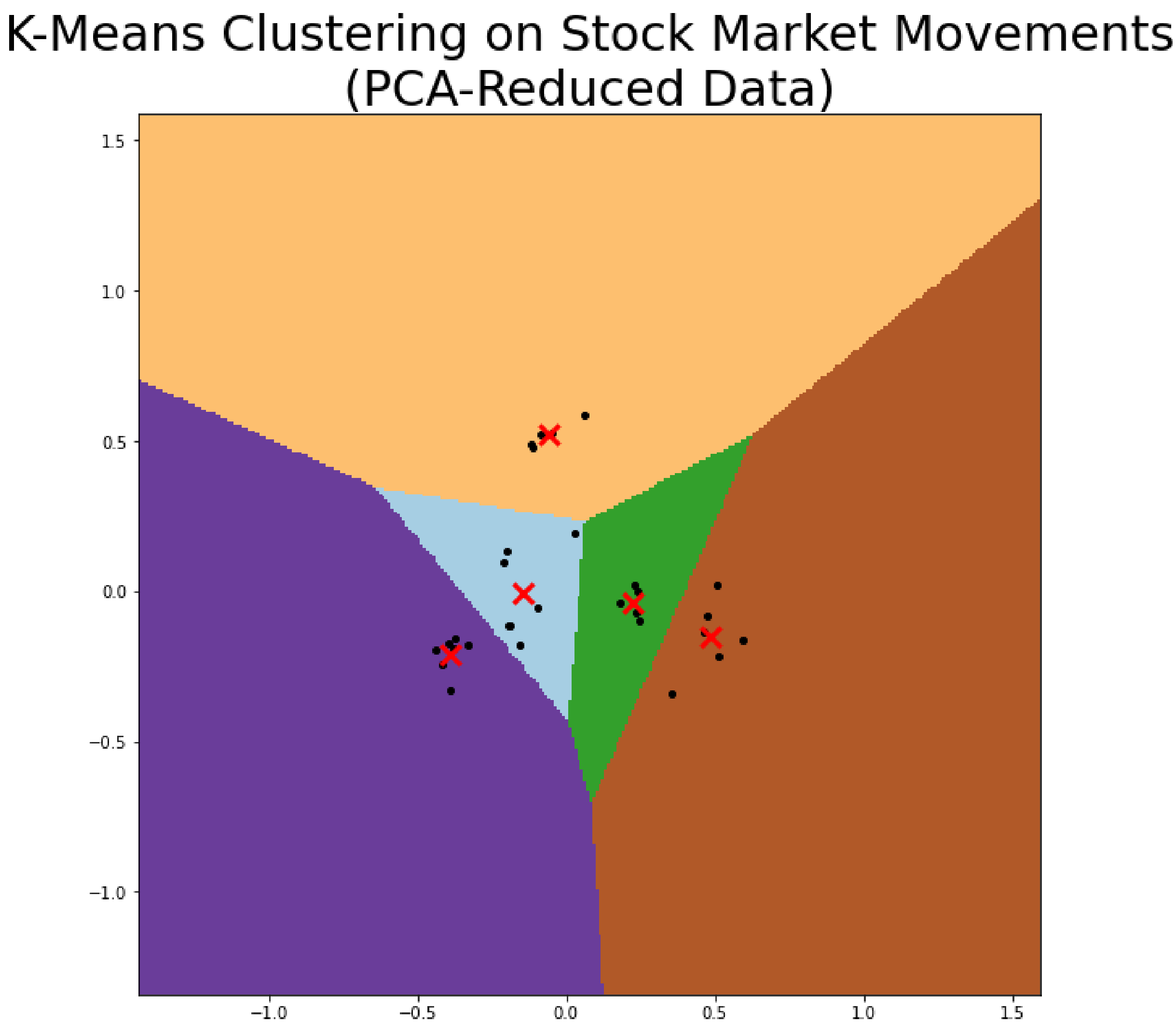
Plotting the Decision Boundary

In [28]:

```
pca = PCA(n_components=2)
pca_data = pca.fit_transform(norm_movements)
# Define step size of mesh
```

```
h = 0.01
# Plot the decision boundary
x_min, x_max = pca_data[:,0].min() - 1, pca_data[:,0].max() + 1
y_min, y_max = pca_data[:,1].min() - 1, pca_data[:,1].max() + 1
xx, yy, = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_min, y_max, h))
# Obtain labels for each point in the mesh using our trained model
kpredictions = kmeans.predict(np.c_[xx.ravel(), yy.ravel()])
# Put the result into a color plot
kpredictions = kpredictions.reshape(xx.shape)
# Define color plot
cmap = plt.cm.Paired
# Plotting figure
plt.clf()
plt.figure(figsize=(10,10))
plt.imshow(kpredictions, interpolation='nearest', extent=(xx.min(), xx.max(),
plt.plot(pca_data[:,0], pca_data[:,1], 'k.', markersize=8)
# Plot the centroid of each cluster as a white X
centroids = kmeans.cluster_centers_
plt.scatter(centroids[:,0], centroids[:,1], marker='x', s=150, linewidths=3, c
plt.title('K-Means Clustering on Stock Market Movements\n(PCA-Reduced Data)',
plt.xlim(x_min, x_max)
plt.ylim(y_min, y_max)
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



Further detail on the above visualization and the process behind it.

In [29]:
```
print(f"x_min (pca_data[:,0].min() - 1): {x_min}")
print(f"This will be the left-most limit for the x-axis.\n-1 or +1 is used to
print(f"x_max (pca_data[:,0].max() + 1): {x_max}")
print(f"This will be the right-most limit for the x-axis.\n{'-'*75}")
print(f"y_min (pca_data[:,1].min() - 1): {y_min}")
print(f"This will be the down-most limit for the y-axis.\n{'-'*75}")
print(f"y_max (pca_data[:,1].max() + 1): {y_max}")
print(f"This will be the up-most limit for the y-axis.\n{'-'*75}")
print(f"np.arange(x_min, x_max, h):\n{np.arange(x_min, x_max, h)}\nShape: {np.
print(f"These will be the x-coordinates with step size 0.01 for the mesh grid.
print(f"np.arange(y_min, y_max, h):\n{np.arange(y_min, y_max, h)}\nShape: {np.
print(f"These will be the y-coordinates with step size 0.01 for the mesh grid.
print(f"{'='*75}\n{'*'*25} Results of Mesh Grid: {'*'*25}\n{'='*75}")
print(f"xx:\n{xx}\nShape: {xx.shape}")
print(f"From left to right on our figure, these will be the x-coordinates that
print(f"yy:\n{yy}\nShape: {yy.shape}")
print(f"From left to right on our figure, these will be the y-coordinates that
print(f"The reason they repeat across each row is because the y value will not
print(f"xx.ravel():\n{xx.ravel()}\nShape: {xx.ravel().shape}")
print(f"This reshapes xx into a 1D array.\n{'-'*75}")
print(f"yy.ravel():\n{yy.ravel()}\nShape: {yy.ravel().shape}")
print(f"This reshapes yy into a 1D array.\nNow both xx and yy will have the sa
print(f"np.c_[xx.ravel(), yy.ravel()]:\n{np.c_[xx.ravel(), yy.ravel()]}\nShape
print(f"This 2D array is a combination of every point in our figure from the x
print(f"All of these x/y combinations will be passed into our KMeans algorithm
print(f"We will utilize this to create a figure where each one of these points
print(f"kmeans.predict(np.c_[xx.ravel(), yy.ravel()]):\n{kmeans.predict(np.c_[
print(f"These are the predictions for every combination of x/y coordinates.\n{
```

```
x_min (pca_data[:,0].min() - 1): -1.4388524605985176
This will be the left-most limit for the x-axis.
-1 or +1 is used to provide more space in the figure in every case.
--------------------------------------------------------------------------
x_max (pca_data[:,0].max() + 1): 1.5965709394990164
This will be the right-most limit for the x-axis.
--------------------------------------------------------------------------
y_min (pca_data[:,1].min() - 1): -1.3410702734121012
This will be the down-most limit for the y-axis.
--------------------------------------------------------------------------
y_max (pca_data[:,1].max() + 1): 1.5875295148997282
This will be the up-most limit for the y-axis.
--------------------------------------------------------------------------
np.arange(x_min, x_max, h):
[-1.43885246e+00 -1.42885246e+00 -1.41885246e+00 -1.40885246e+00
 -1.39885246e+00 -1.38885246e+00 -1.37885246e+00 -1.36885246e+00
 -1.35885246e+00 -1.34885246e+00 -1.33885246e+00 -1.32885246e+00
 -1.31885246e+00 -1.30885246e+00 -1.29885246e+00 -1.28885246e+00
 -1.27885246e+00 -1.26885246e+00 -1.25885246e+00 -1.24885246e+00
 -1.23885246e+00 -1.22885246e+00 -1.21885246e+00 -1.20885246e+00
 -1.19885246e+00 -1.18885246e+00 -1.17885246e+00 -1.16885246e+00
 -1.15885246e+00 -1.14885246e+00 -1.13885246e+00 -1.12885246e+00
 -1.11885246e+00 -1.10885246e+00 -1.09885246e+00 -1.08885246e+00
 -1.07885246e+00 -1.06885246e+00 -1.05885246e+00 -1.04885246e+00
 -1.03885246e+00 -1.02885246e+00 -1.01885246e+00 -1.00885246e+00
 -9.98852461e-01 -9.88852461e-01 -9.78852461e-01 -9.68852461e-01
 -9.58852461e-01 -9.48852461e-01 -9.38852461e-01 -9.28852461e-01
 -9.18852461e-01 -9.08852461e-01 -8.98852461e-01 -8.88852461e-01
 -8.78852461e-01 -8.68852461e-01 -8.58852461e-01 -8.48852461e-01
 -8.38852461e-01 -8.28852461e-01 -8.18852461e-01 -8.08852461e-01
 -7.98852461e-01 -7.88852461e-01 -7.78852461e-01 -7.68852461e-01
 -7.58852461e-01 -7.48852461e-01 -7.38852461e-01 -7.28852461e-01
 -7.18852461e-01 -7.08852461e-01 -6.98852461e-01 -6.88852461e-01
 -6.78852461e-01 -6.68852461e-01 -6.58852461e-01 -6.48852461e-01
 -6.38852461e-01 -6.28852461e-01 -6.18852461e-01 -6.08852461e-01
 -5.98852461e-01 -5.88852461e-01 -5.78852461e-01 -5.68852461e-01
 -5.58852461e-01 -5.48852461e-01 -5.38852461e-01 -5.28852461e-01
 -5.18852461e-01 -5.08852461e-01 -4.98852461e-01 -4.88852461e-01
 -4.78852461e-01 -4.68852461e-01 -4.58852461e-01 -4.48852461e-01
 -4.38852461e-01 -4.28852461e-01 -4.18852461e-01 -4.08852461e-01
 -3.98852461e-01 -3.88852461e-01 -3.78852461e-01 -3.68852461e-01
 -3.58852461e-01 -3.48852461e-01 -3.38852461e-01 -3.28852461e-01
 -3.18852461e-01 -3.08852461e-01 -2.98852461e-01 -2.88852461e-01
 -2.78852461e-01 -2.68852461e-01 -2.58852461e-01 -2.48852461e-01
 -2.38852461e-01 -2.28852461e-01 -2.18852461e-01 -2.08852461e-01
 -1.98852461e-01 -1.88852461e-01 -1.78852461e-01 -1.68852461e-01
 -1.58852461e-01 -1.48852461e-01 -1.38852461e-01 -1.28852461e-01
 -1.18852461e-01 -1.08852461e-01 -9.88524606e-02 -8.88524606e-02
 -7.88524606e-02 -6.88524606e-02 -5.88524606e-02 -4.88524606e-02
 -3.88524606e-02 -2.88524606e-02 -1.88524606e-02 -8.85246060e-03
  1.14753940e-03  1.11475394e-02  2.11475394e-02  3.11475394e-02
  4.11475394e-02  5.11475394e-02  6.11475394e-02  7.11475394e-02
  8.11475394e-02  9.11475394e-02  1.01147539e-01  1.11147539e-01
  1.21147539e-01  1.31147539e-01  1.41147539e-01  1.51147539e-01
  1.61147539e-01  1.71147539e-01  1.81147539e-01  1.91147539e-01
  2.01147539e-01  2.11147539e-01  2.21147539e-01  2.31147539e-01
  2.41147539e-01  2.51147539e-01  2.61147539e-01  2.71147539e-01
  2.81147539e-01  2.91147539e-01  3.01147539e-01  3.11147539e-01
  3.21147539e-01  3.31147539e-01  3.41147539e-01  3.51147539e-01
  3.61147539e-01  3.71147539e-01  3.81147539e-01  3.91147539e-01
```

```
  4.01147539e-01  4.11147539e-01  4.21147539e-01  4.31147539e-01
  4.41147539e-01  4.51147539e-01  4.61147539e-01  4.71147539e-01
  4.81147539e-01  4.91147539e-01  5.01147539e-01  5.11147539e-01
  5.21147539e-01  5.31147539e-01  5.41147539e-01  5.51147539e-01
  5.61147539e-01  5.71147539e-01  5.81147539e-01  5.91147539e-01
  6.01147539e-01  6.11147539e-01  6.21147539e-01  6.31147539e-01
  6.41147539e-01  6.51147539e-01  6.61147539e-01  6.71147539e-01
  6.81147539e-01  6.91147539e-01  7.01147539e-01  7.11147539e-01
  7.21147539e-01  7.31147539e-01  7.41147539e-01  7.51147539e-01
  7.61147539e-01  7.71147539e-01  7.81147539e-01  7.91147539e-01
  8.01147539e-01  8.11147539e-01  8.21147539e-01  8.31147539e-01
  8.41147539e-01  8.51147539e-01  8.61147539e-01  8.71147539e-01
  8.81147539e-01  8.91147539e-01  9.01147539e-01  9.11147539e-01
  9.21147539e-01  9.31147539e-01  9.41147539e-01  9.51147539e-01
  9.61147539e-01  9.71147539e-01  9.81147539e-01  9.91147539e-01
  1.00114754e+00  1.01114754e+00  1.02114754e+00  1.03114754e+00
  1.04114754e+00  1.05114754e+00  1.06114754e+00  1.07114754e+00
  1.08114754e+00  1.09114754e+00  1.10114754e+00  1.11114754e+00
  1.12114754e+00  1.13114754e+00  1.14114754e+00  1.15114754e+00
  1.16114754e+00  1.17114754e+00  1.18114754e+00  1.19114754e+00
  1.20114754e+00  1.21114754e+00  1.22114754e+00  1.23114754e+00
  1.24114754e+00  1.25114754e+00  1.26114754e+00  1.27114754e+00
  1.28114754e+00  1.29114754e+00  1.30114754e+00  1.31114754e+00
  1.32114754e+00  1.33114754e+00  1.34114754e+00  1.35114754e+00
  1.36114754e+00  1.37114754e+00  1.38114754e+00  1.39114754e+00
  1.40114754e+00  1.41114754e+00  1.42114754e+00  1.43114754e+00
  1.44114754e+00  1.45114754e+00  1.46114754e+00  1.47114754e+00
  1.48114754e+00  1.49114754e+00  1.50114754e+00  1.51114754e+00
  1.52114754e+00  1.53114754e+00  1.54114754e+00  1.55114754e+00
  1.56114754e+00  1.57114754e+00  1.58114754e+00  1.59114754e+00]
Shape: (304,)
These will be the x-coordinates with step size 0.01 for the mesh grid.
----------------------------------------------------------------------------
np.arange(y_min, y_max, h):
[-1.34107027e+00 -1.33107027e+00 -1.32107027e+00 -1.31107027e+00
 -1.30107027e+00 -1.29107027e+00 -1.28107027e+00 -1.27107027e+00
 -1.26107027e+00 -1.25107027e+00 -1.24107027e+00 -1.23107027e+00
 -1.22107027e+00 -1.21107027e+00 -1.20107027e+00 -1.19107027e+00
 -1.18107027e+00 -1.17107027e+00 -1.16107027e+00 -1.15107027e+00
 -1.14107027e+00 -1.13107027e+00 -1.12107027e+00 -1.11107027e+00
 -1.10107027e+00 -1.09107027e+00 -1.08107027e+00 -1.07107027e+00
 -1.06107027e+00 -1.05107027e+00 -1.04107027e+00 -1.03107027e+00
 -1.02107027e+00 -1.01107027e+00 -1.00107027e+00 -9.91070273e-01
 -9.81070273e-01 -9.71070273e-01 -9.61070273e-01 -9.51070273e-01
 -9.41070273e-01 -9.31070273e-01 -9.21070273e-01 -9.11070273e-01
 -9.01070273e-01 -8.91070273e-01 -8.81070273e-01 -8.71070273e-01
 -8.61070273e-01 -8.51070273e-01 -8.41070273e-01 -8.31070273e-01
 -8.21070273e-01 -8.11070273e-01 -8.01070273e-01 -7.91070273e-01
 -7.81070273e-01 -7.71070273e-01 -7.61070273e-01 -7.51070273e-01
 -7.41070273e-01 -7.31070273e-01 -7.21070273e-01 -7.11070273e-01
 -7.01070273e-01 -6.91070273e-01 -6.81070273e-01 -6.71070273e-01
 -6.61070273e-01 -6.51070273e-01 -6.41070273e-01 -6.31070273e-01
 -6.21070273e-01 -6.11070273e-01 -6.01070273e-01 -5.91070273e-01
 -5.81070273e-01 -5.71070273e-01 -5.61070273e-01 -5.51070273e-01
 -5.41070273e-01 -5.31070273e-01 -5.21070273e-01 -5.11070273e-01
 -5.01070273e-01 -4.91070273e-01 -4.81070273e-01 -4.71070273e-01
 -4.61070273e-01 -4.51070273e-01 -4.41070273e-01 -4.31070273e-01
 -4.21070273e-01 -4.11070273e-01 -4.01070273e-01 -3.91070273e-01
 -3.81070273e-01 -3.71070273e-01 -3.61070273e-01 -3.51070273e-01
 -3.41070273e-01 -3.31070273e-01 -3.21070273e-01 -3.11070273e-01
```

```
 -3.01070273e-01 -2.91070273e-01 -2.81070273e-01 -2.71070273e-01
 -2.61070273e-01 -2.51070273e-01 -2.41070273e-01 -2.31070273e-01
 -2.21070273e-01 -2.11070273e-01 -2.01070273e-01 -1.91070273e-01
 -1.81070273e-01 -1.71070273e-01 -1.61070273e-01 -1.51070273e-01
 -1.41070273e-01 -1.31070273e-01 -1.21070273e-01 -1.11070273e-01
 -1.01070273e-01 -9.10702734e-02 -8.10702734e-02 -7.10702734e-02
 -6.10702734e-02 -5.10702734e-02 -4.10702734e-02 -3.10702734e-02
 -2.10702734e-02 -1.10702734e-02 -1.07027341e-03  8.92972659e-03
  1.89297266e-02  2.89297266e-02  3.89297266e-02  4.89297266e-02
  5.89297266e-02  6.89297266e-02  7.89297266e-02  8.89297266e-02
  9.89297266e-02  1.08929727e-01  1.18929727e-01  1.28929727e-01
  1.38929727e-01  1.48929727e-01  1.58929727e-01  1.68929727e-01
  1.78929727e-01  1.88929727e-01  1.98929727e-01  2.08929727e-01
  2.18929727e-01  2.28929727e-01  2.38929727e-01  2.48929727e-01
  2.58929727e-01  2.68929727e-01  2.78929727e-01  2.88929727e-01
  2.98929727e-01  3.08929727e-01  3.18929727e-01  3.28929727e-01
  3.38929727e-01  3.48929727e-01  3.58929727e-01  3.68929727e-01
  3.78929727e-01  3.88929727e-01  3.98929727e-01  4.08929727e-01
  4.18929727e-01  4.28929727e-01  4.38929727e-01  4.48929727e-01
  4.58929727e-01  4.68929727e-01  4.78929727e-01  4.88929727e-01
  4.98929727e-01  5.08929727e-01  5.18929727e-01  5.28929727e-01
  5.38929727e-01  5.48929727e-01  5.58929727e-01  5.68929727e-01
  5.78929727e-01  5.88929727e-01  5.98929727e-01  6.08929727e-01
  6.18929727e-01  6.28929727e-01  6.38929727e-01  6.48929727e-01
  6.58929727e-01  6.68929727e-01  6.78929727e-01  6.88929727e-01
  6.98929727e-01  7.08929727e-01  7.18929727e-01  7.28929727e-01
  7.38929727e-01  7.48929727e-01  7.58929727e-01  7.68929727e-01
  7.78929727e-01  7.88929727e-01  7.98929727e-01  8.08929727e-01
  8.18929727e-01  8.28929727e-01  8.38929727e-01  8.48929727e-01
  8.58929727e-01  8.68929727e-01  8.78929727e-01  8.88929727e-01
  8.98929727e-01  9.08929727e-01  9.18929727e-01  9.28929727e-01
  9.38929727e-01  9.48929727e-01  9.58929727e-01  9.68929727e-01
  9.78929727e-01  9.88929727e-01  9.98929727e-01  1.00892973e+00
  1.01892973e+00  1.02892973e+00  1.03892973e+00  1.04892973e+00
  1.05892973e+00  1.06892973e+00  1.07892973e+00  1.08892973e+00
  1.09892973e+00  1.10892973e+00  1.11892973e+00  1.12892973e+00
  1.13892973e+00  1.14892973e+00  1.15892973e+00  1.16892973e+00
  1.17892973e+00  1.18892973e+00  1.19892973e+00  1.20892973e+00
  1.21892973e+00  1.22892973e+00  1.23892973e+00  1.24892973e+00
  1.25892973e+00  1.26892973e+00  1.27892973e+00  1.28892973e+00
  1.29892973e+00  1.30892973e+00  1.31892973e+00  1.32892973e+00
  1.33892973e+00  1.34892973e+00  1.35892973e+00  1.36892973e+00
  1.37892973e+00  1.38892973e+00  1.39892973e+00  1.40892973e+00
  1.41892973e+00  1.42892973e+00  1.43892973e+00  1.44892973e+00
  1.45892973e+00  1.46892973e+00  1.47892973e+00  1.48892973e+00
  1.49892973e+00  1.50892973e+00  1.51892973e+00  1.52892973e+00
  1.53892973e+00  1.54892973e+00  1.55892973e+00  1.56892973e+00
  1.57892973e+00]
Shape: (293,)
These will be the y-coordinates with step size 0.01 for the mesh grid.
---------------------------------------------------------------------------
===========================================================================
************************ Results of Mesh Grid: ************************
===========================================================================
xx:
[[-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
   1.59114754]
 [-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
   1.59114754]
 [-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
```

```
  1.59114754]
 ...
 [-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
   1.59114754]
 [-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
   1.59114754]
 [-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
   1.59114754]]
Shape: (293, 304)
From left to right on our figure, these will be the x-coordinates that will be
used to create a color-map.
---------------------------------------------------------------------------
yy:
[[-1.34107027 -1.34107027 -1.34107027 ... -1.34107027 -1.34107027
  -1.34107027]
 [-1.33107027 -1.33107027 -1.33107027 ... -1.33107027 -1.33107027
  -1.33107027]
 [-1.32107027 -1.32107027 -1.32107027 ... -1.32107027 -1.32107027
  -1.32107027]
 ...
 [ 1.55892973  1.55892973  1.55892973 ...  1.55892973  1.55892973
   1.55892973]
 [ 1.56892973  1.56892973  1.56892973 ...  1.56892973  1.56892973
   1.56892973]
 [ 1.57892973  1.57892973  1.57892973 ...  1.57892973  1.57892973
   1.57892973]]
Shape: (293, 304)
From left to right on our figure, these will be the y-coordinates that will be
used to create a color-map.
The reason they repeat across each row is because the y value will not change
as we move horizontally in the figure.
---------------------------------------------------------------------------
xx.ravel():
[-1.43885246 -1.42885246 -1.41885246 ...  1.57114754  1.58114754
  1.59114754]
Shape: (89072,)
This reshapes xx into a 1D array.
---------------------------------------------------------------------------
yy.ravel():
[-1.34107027 -1.34107027 -1.34107027 ...  1.57892973  1.57892973
  1.57892973]
Shape: (89072,)
This reshapes yy into a 1D array.
Now both xx and yy will have the same shape, and will be concatenated.
---------------------------------------------------------------------------
np.c_[xx.ravel(), yy.ravel()]:
[[-1.43885246 -1.34107027]
 [-1.42885246 -1.34107027]
 [-1.41885246 -1.34107027]
 ...
 [ 1.57114754  1.57892973]
 [ 1.58114754  1.57892973]
 [ 1.59114754  1.57892973]]
Shape: (89072, 2)
This 2D array is a combination of every point in our figure from the x-min/x-m
ax to the y-min/y-max.
All of these x/y combinations will be passed into our KMeans algorithm to make
a prediction as to which cluster they belong to.
We will utilize this to create a figure where each one of these points is colo
r-coded with respect to the cluster they belong to.
```

```
-----------------------------------------------------------------------------
kmeans.predict(np.c_[xx.ravel(), yy.ravel()]):
[3 3 3 ... 2 2 2]
Shape: (89072,)
These are the predictions for every combination of x/y coordinates.
-----------------------------------------------------------------------------
```

In [30]:

```python
fig, ax = plt.subplots(1,2, figsize=(15,6))
ax[0].scatter(xx,yy, s=0.01)
ax[0].set_title('X/Y Coordinate Combinations', fontsize=25)
ax[1].imshow(kpredictions)
ax[1].set_title('KMeans Predictions\nfor Each X/Y Coordinate', fontsize=25)
plt.show()
```