In [224…
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima import auto_arima
from sklearn.metrics import mean_absolute_percentage_error
```

In [2]:
```
xl_file_1 = pd.ExcelFile(r"D:\Data Science\Data Science Career Readiness Program\Datas
```

In [3]:
```
DF1 = xl_file_1.parse('Sheet1')
DF1.head()
```

Out[3]:

	Symbol	Name	Market Cap	Last Sale	Net Change	Percentage Change
0	AAPL	Apple Inc.	2625740143000	$151.45	$2.00	0.0134
1	ABNB	Airbnb, Inc.	69569944167	$116.65	$0.26	0.0022
2	ADBE	Adobe Inc.	149144569000	$320.81	$4.59	0.0145
3	ADI	Analog Devices, Inc.	75484763090	$146.76	$2.23	0.0154
4	ADP	Automatic Data Processing, Inc.	98332762096	$236.78	$0.13	-0.0005

In [4]:
```
DF1.shape
```

Out[4]:
```
(102, 6)
```

In [5]:
```
xl_file_2 = pd.ExcelFile(r"D:\Data Science\Data Science Career Readiness Program\Datas
```

In [6]:
```
DF2 = xl_file_2.parse('Sheet 1')
DF2.head()
```

Out[6]:

	Symbol	company	sector	subsector	asset_turnover_2017	asset_turnover_2018	asset_tu
0	AAPL	Apple Inc.	Information Technology	Technology Hardware, Storage & Peripherals	0.66	0.72	
1	ABNB	Airbnb	Consumer Discretionary	Internet & Direct Marketing Retail	NaN	0.55	
2	ADBE	Adobe Inc.	Information Technology	Application Software	0.54	0.54	
3	ADI	Analog Devices	Information Technology	Semiconductors	0.36	0.30	
4	ADP	ADP	Information Technology	Data Processing & Outsourced Services	NaN	0.34	

5 rows × 283 columns

In [7]:
```
DF2.shape
```

Out[7]:
```
(102, 283)
```

In [8]:
```
# SMA - Share Market Analysis
SMA = pd.merge(DF1, DF2, on = "Symbol", how = "inner")
SMA.head()
```

Out[8]:

	Symbol	Name	Market Cap	Last Sale	Net Change	Percentage Change	company	sector	su
0	AAPL	Apple Inc.	2625740143000	$151.45	$2.00	0.0134	Apple Inc.	Information Technology	Tecl Ha St Per
1	ABNB	Airbnb, Inc.	69569944167	$116.65	$0.26	0.0022	Airbnb	Consumer Discretionary	Int Ma
2	ADBE	Adobe Inc.	149144569000	$320.81	$4.59	0.0145	Adobe Inc.	Information Technology	App S
3	ADI	Analog Devices, Inc.	75484763090	$146.76	$2.23	0.0154	Analog Devices	Information Technology	Semicon
4	ADP	Automatic Data Processing, Inc.	98332762096	$236.78	$0.13	-0.0005	ADP	Information Technology	Data Pro & Out

5 rows × 288 columns

In [9]:
```
SMA.shape
```

Out[9]:
```
(102, 288)
```

In [10]:
```
SMA.columns
```

Out[10]:
```
Index(['Symbol', 'Name', 'Market Cap', 'Last Sale', 'Net Change',
       'Percentage Change', 'company', 'sector', 'subsector',
       'asset_turnover_2017',
       ...
       'yoy_eps_growth_2021', 'yoy_eps_growth_2022', 'yoy_eps_growth_latest',
       'yoy_revenue_growth_2017', 'yoy_revenue_growth_2018',
       'yoy_revenue_growth_2019', 'yoy_revenue_growth_2020',
       'yoy_revenue_growth_2021', 'yoy_revenue_growth_2022',
       'yoy_revenue_growth_latest'],
      dtype='object', length=288)
```

In [11]:
```
SMA.isnull().sum()
```

Out[11]:
```
Symbol                      0
Name                        0
Market Cap                  0
Last Sale                   0
Net Change                  0
                           ..
yoy_revenue_growth_2019     1
yoy_revenue_growth_2020     0
yoy_revenue_growth_2021     2
yoy_revenue_growth_2022    76
yoy_revenue_growth_latest   0
Length: 288, dtype: int64
```

In [12]:
```
SMA.describe()
```

Out[12]:

	Market Cap	Percentage Change	asset_turnover_2017	asset_turnover_2018	asset_turnover_2019	asset
count	1.020000e+02	102.000000	73.000000	100.000000	101.000000	
mean	1.559159e+11	0.020855	0.724932	0.746400	0.715842	
std	3.777520e+11	0.020055	0.529098	0.521082	0.507563	
min	8.980286e+09	-0.041700	0.140000	0.080000	0.030000	
25%	3.260737e+10	0.007200	0.400000	0.430000	0.410000	
50%	4.836661e+10	0.017150	0.600000	0.605000	0.580000	
75%	9.950980e+10	0.035150	0.850000	0.885000	0.810000	
max	2.625740e+12	0.079300	3.710000	3.670000	3.540000	

8 rows × 281 columns

In [13]:
```
missing_percentages = (SMA.isnull().sum() / len(SMA)) * 100
```

In [14]:
```
variables_to_delete = missing_percentages[missing_percentages >= 30].index
```

In [15]:
```
SMA = SMA.drop(variables_to_delete, axis=1)
```

In [16]:
```
SMA.shape
```

Out[16]:
```
(102, 204)
```

In [17]:
```
variables_to_impute = missing_percentages[missing_percentages < 30].index
```

In [18]:
```
grouped_data = SMA.groupby('sector')
```

In [19]:
```
for sector, sector_data in grouped_data:
    for variable in variables_to_impute:
        if sector_data[variable].isnull().sum() > 0:
            if sector_data[variable].dtype in ['int64', 'float64']:
                sector_data[variable].fillna(sector_data[variable].mean(), inplace=Tru
            else:
                sector_data[variable].fillna(sector_data[variable].mode()[0], inplace=
```

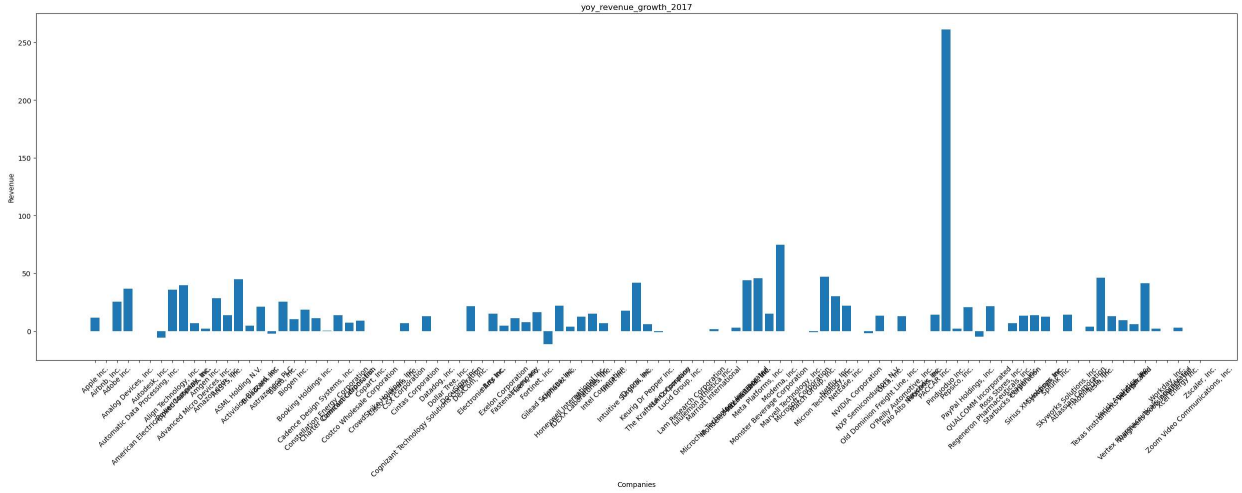
In [20]:
```
SMA.loc[SMA['sector'] == sector, variables_to_impute] = sector_data[variables_to_imput
```

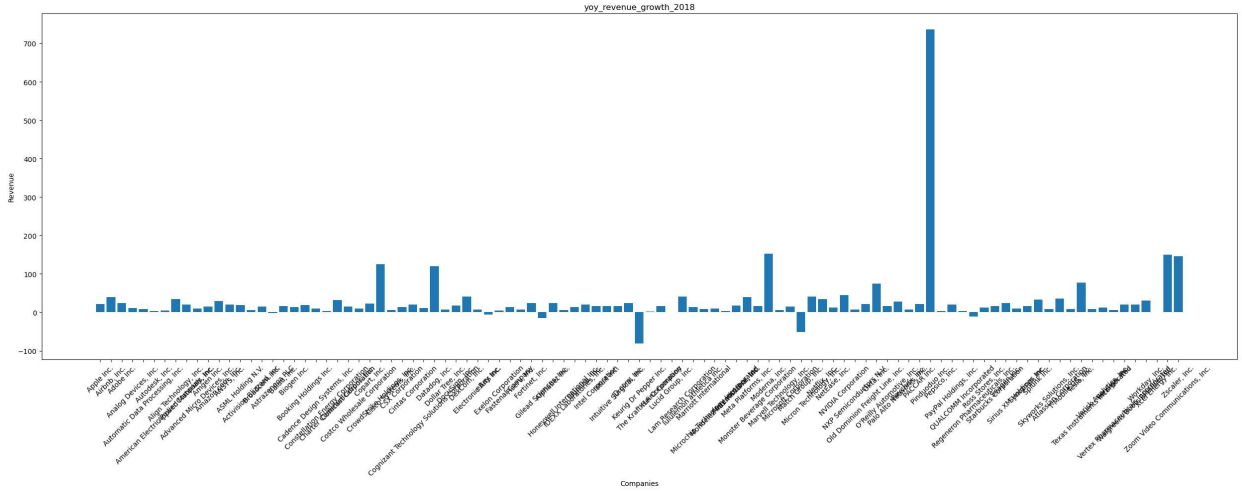
In [21]:
```
missing_percentages_after_imputation = (SMA.isnull().sum() / len(SMA)) * 100
print(missing_percentages_after_imputation)
```

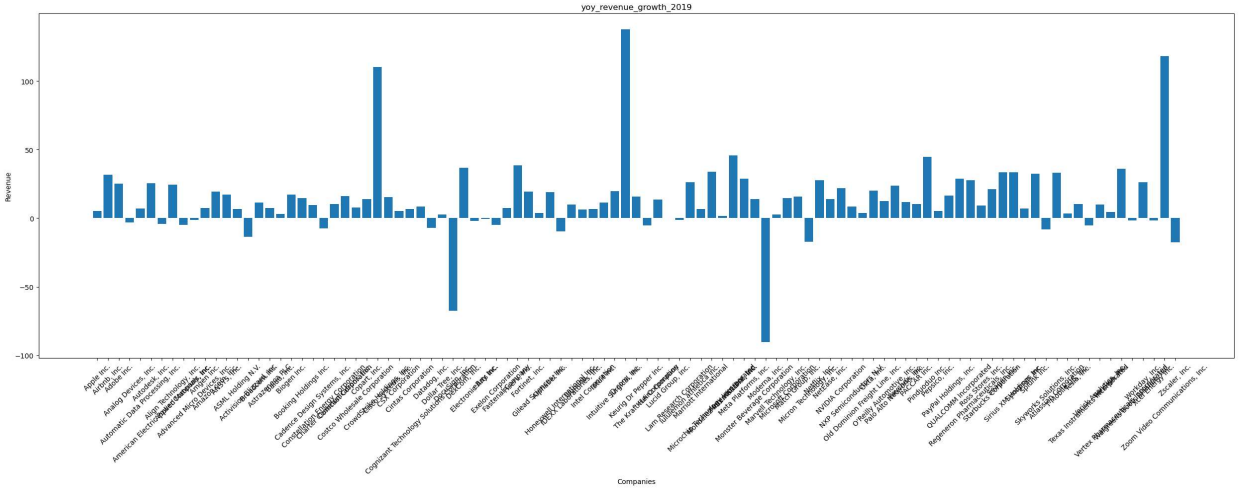
```
Symbol                        0.000000
Name                          0.000000
Market Cap                    0.000000
Last Sale                     0.000000
Net Change                    0.000000
                                 ...
yoy_revenue_growth_2018       0.980392
yoy_revenue_growth_2019       0.980392
yoy_revenue_growth_2020       0.000000
yoy_revenue_growth_2021       1.960784
yoy_revenue_growth_latest     0.000000
Length: 204, dtype: float64
```

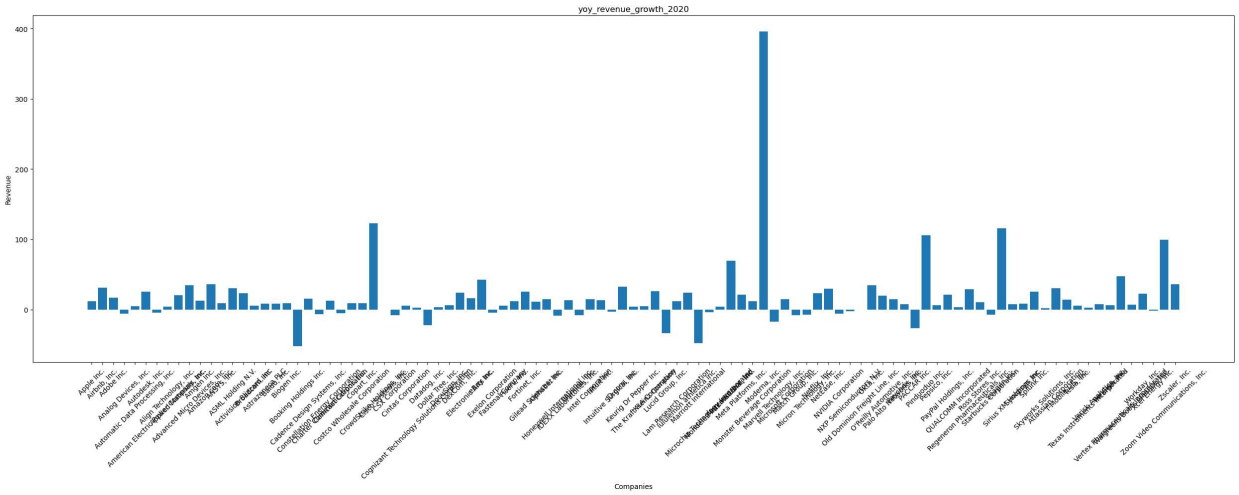
In [63]:
```
companies = SMA['Name']
revenue_columns = ['yoy_revenue_growth_2017', 'yoy_revenue_growth_2018', 'yoy_revenue_
```

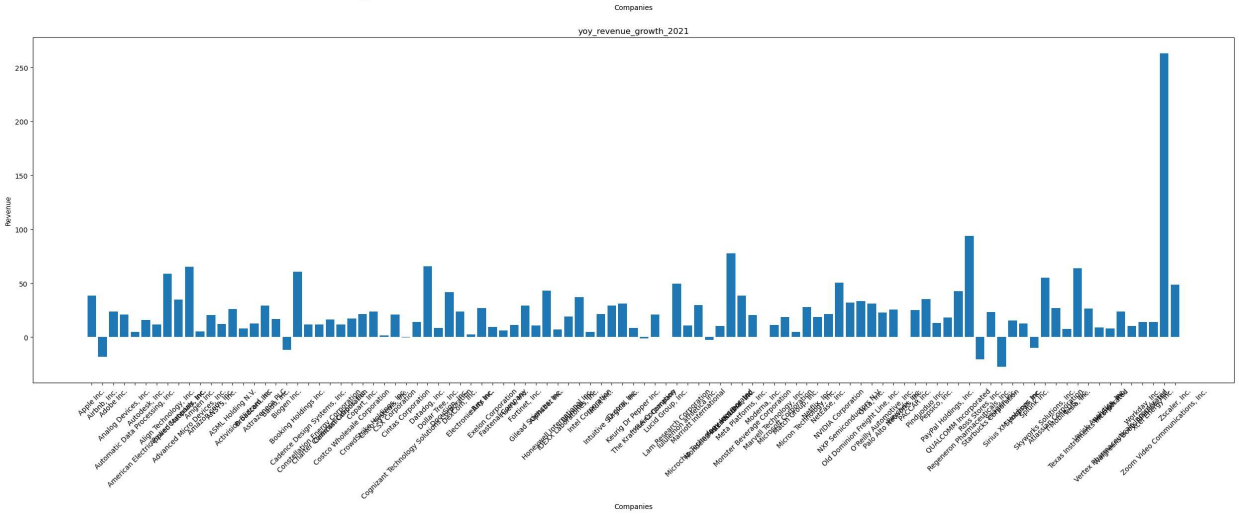
In [71]:
```
for year in revenue_columns:
    plt.figure(figsize=(25, 10))
    plt.bar(companies, SMA[year])
    plt.title(year)
    plt.xlabel('Companies')
    plt.ylabel('Revenue')
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.show()
```

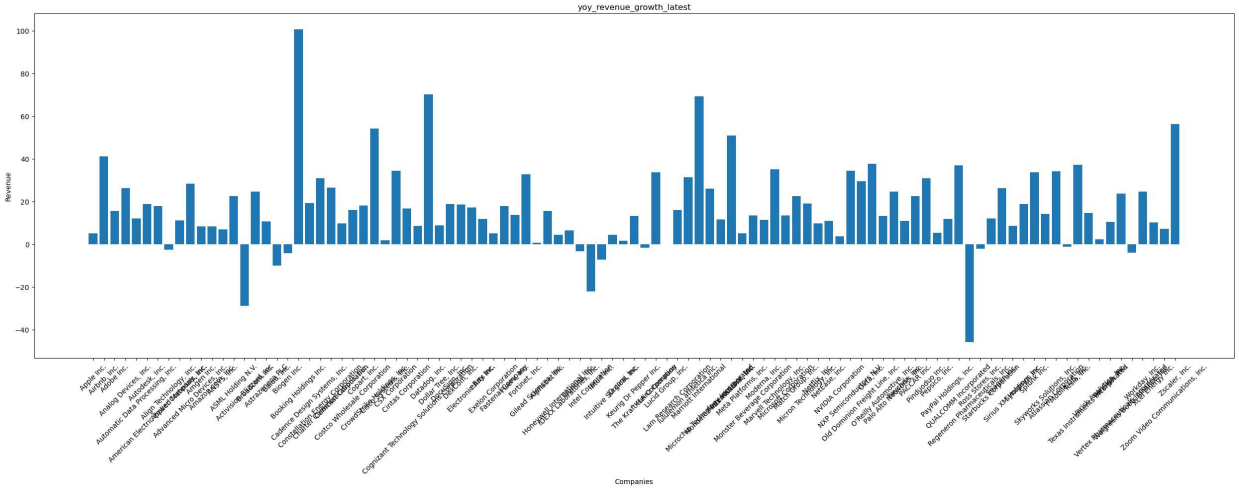





yoy_revenue_growth_2019
Revenue
Companies


yoy_revenue_growth_2020
Revenue
Companies


yoy_revenue_growth_2021
Revenue
Companies



In [22]:
```
numeric_data = SMA.drop(['Symbol', 'Name', 'company', 'sector', 'subsector'], axis = 1
numeric_data.head()
```

Out[22]:

	Market Cap	Last Sale	Net Change	Percentage Change	asset_turnover_2017	asset_turnover_2018	asset_turnc
0	2625740143000	$151.45	$2.00	0.0134	0.66	0.72	
1	69569944167	$116.65	$0.26	0.0022	NaN	0.55	
2	149144569000	$320.81	$4.59	0.0145	0.54	0.54	
3	75484763090	$146.76	$2.23	0.0154	0.36	0.30	
4	98332762096	$236.78	$0.13	-0.0005	NaN	0.34	

5 rows × 199 columns

In [23]:
```
def currency_to_int(currency):
    clean_value = currency.replace("$", "").replace(",", "").replace(".", "")
    return int(clean_value)
```

In [24]:
```
numeric_data['Last Sale'] = numeric_data['Last Sale'].apply(currency_to_int)
```

In [25]:
```
numeric_data['Net Change'] = numeric_data['Net Change'].apply(currency_to_int)
```

In [26]:
```
numeric_data.head()
```

Out[26]:

	Market Cap	Last Sale	Net Change	Percentage Change	asset_turnover_2017	asset_turnover_2018	asset_turnov
0	2625740143000	15145	200	0.0134	0.66	0.72	
1	69569944167	11665	26	0.0022	NaN	0.55	
2	149144569000	32081	459	0.0145	0.54	0.54	
3	75484763090	14676	223	0.0154	0.36	0.30	
4	98332762096	23678	13	-0.0005	NaN	0.34	

5 rows × 199 columns

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

Out[27]:
```
(102, 199)
```

In [28]:
```
numeric_data = numeric_data.dropna(axis = 1)
```

In [29]:
```
numeric_data.shape
```

Out[29]:
```
(102, 51)
```

In [30]:
```
# Step 1: Standardize the data
scaler = StandardScaler()
scaled_SMA = scaler.fit_transform(numeric_data)
```

In [31]:
```
# Step 2: Perform PCA
pca = PCA()
pca.fit(scaled_SMA)
```

Out[31]:
```
▾ PCA
PCA()
```

In [32]:
```
# Step 3: Determine the optimal number of components
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance = np.cumsum(explained_variance_ratio)
num_components = np.argmax(cumulative_variance >= 0.95) + 1
```

In [33]:
```
# Step 4: Select the principal components
pca = PCA(n_components=num_components)
reduced_data = pca.fit_transform(scaled_SMA)
```

In [34]:
```
# Step 5: Access the principal components and explained variance ratio
principal_components = pca.components_
explained_variance_ratio = pca.explained_variance_ratio_
```

In [35]:
```
# Step 6: Project the data onto the reduced-dimensional space
projected_data = pca.inverse_transform(reduced_data)
projected_data
```

Out[35]:
```
array([[ 6.02727722e+00,  5.31753610e-03, -8.37575739e-01, ...,
        -4.91639219e-01, -1.09083175e-01, -6.69079315e-01],
       [-3.12219952e-01, -1.76821768e-01, -6.99998832e-01, ...,
         2.08264985e+00,  1.09335404e+00,  1.33820061e+00],
       [ 3.50263574e-02, -3.98884396e-01, -4.11398047e-02, ...,
        -2.07959818e-01, -1.35427429e-01,  1.19971258e-02],
       ...,
       [-2.25487266e-01, -5.80702636e-01, -3.13369706e-01, ...,
         2.83234661e-02, -4.38009622e-01, -2.52063970e-01],
       [-3.58850595e-01, -5.66610099e-01, -2.35954338e-01, ...,
        -4.68682218e-01,  1.68998412e+00, -9.09940536e-01],
       [-4.09121587e-01, -6.05683492e-01,  9.52570843e-02, ...,
        -1.16121116e-01,  3.50044889e-01,  2.20564709e+00]])
```

In [36]:
```
# Perform cluster analysis using K-means
kmeans = KMeans(n_clusters=3)  # Specify the desired number of clusters
kmeans.fit(reduced_data)
```

```
C:\Users\naray\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1332: UserWarni
ng: KMeans is known to have a memory leak on Windows with MKL, when there are less ch
unks than available threads. You can avoid it by setting the environment variable OMP
_NUM_THREADS=1.
  warnings.warn(
```

Out[36]:
```
KMeans
KMeans(n_clusters=3)
```

In [37]:
```
# Get the cluster labels
cluster_labels = kmeans.labels_
```

In [38]:
```
# Add the cluster labels to the original dataset
data_with_clusters = numeric_data.copy()
data_with_clusters['Cluster'] = cluster_labels
```

In [39]:
```
# Perform cluster profiling
cluster_profiles = data_with_clusters.groupby('Cluster').mean()
```

In [40]:
```
# Display the cluster profiles
print(cluster_profiles)
```

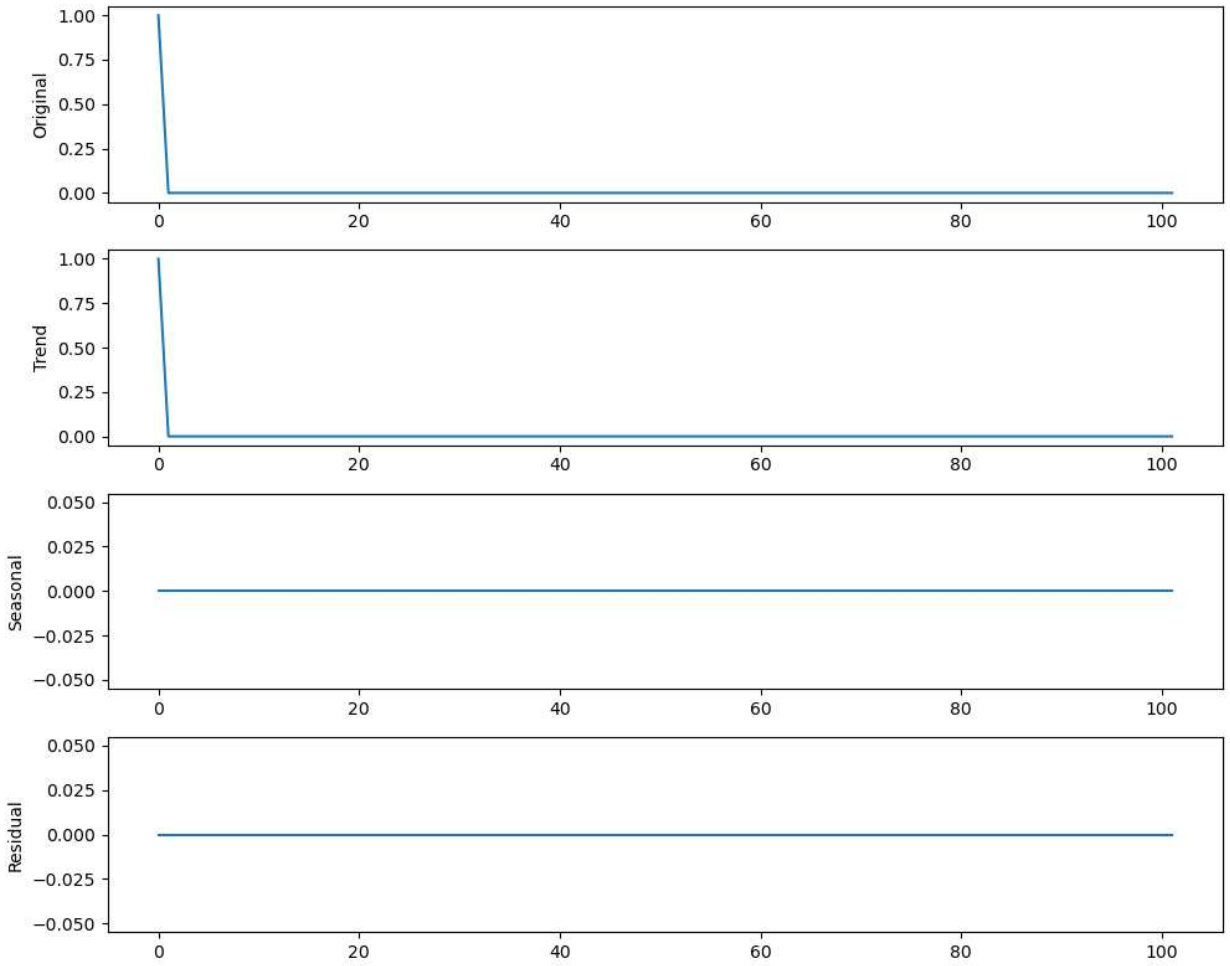
```
           Market Cap      Last Sale   Net Change  Percentage Change  \
Cluster
0        2.003056e+11  110988.307692  3280.025641           0.020631
1        1.301343e+11   70022.258065  1410.822581           0.020053
2        2.317920e+10   13815.000000  1015.000000           0.079300

         asset_turnover_2021  asset_turnover_latest  buyback_yield_latest  \
Cluster
0                   0.713590               0.169231              2.562821
1                   0.663065               0.175806              3.084516
2                   0.010000               0.010000              0.810000

         capex_to_revenue_latest  cash_ratio_2020  cash_ratio_2021  ...  \
Cluster                                                             ...
0                       0.079487         2.229231         1.869231  ...
1                       0.059677         0.786452         0.683226  ...
2                       3.180000         3.320000        15.810000  ...

         liabilities_to_assets_2021  liabilities_to_assets_latest  \
Cluster
0                          0.353077                      0.344359
1                          0.744355                      0.768871
2                          0.500000                      0.480000

         longterm_debt_to_assets_latest  profitability  rate_of_return_latest  \
Cluster
0                              0.121026       8.384615              13.594615
1                              0.347419       7.500000               7.598226
2                              0.310000       0.000000               0.000000

         scaled_net_operating_assets_latest  yoy_ebitda_growth_latest  \
Cluster
0                                  0.535641               1095.141282
1                                  0.438226                 31.704516
2                                  0.220000                 35.030000

         yoy_eps_growth_latest  yoy_revenue_growth_2020  \
Cluster
0                    11.301795                25.221282
1                    64.809032                12.867419
2                    19.750000               -33.330000

         yoy_revenue_growth_latest
Cluster
0                        13.166667
1                        19.303548
2                         0.000000

[3 rows x 51 columns]
```

In [41]:
```
index = np.arange(len(SMA['Name'] == 'Apple Inc.'))
```

In [42]:
```
result = seasonal_decompose((SMA['Name'] == 'Apple Inc.'), model='additive', period=1)
```

In [43]:
```
fig, axes = plt.subplots(4, 1, figsize=(10, 8))

# Original time series
axes[0].plot(index, (SMA['Name'] == 'Apple Inc.'))
axes[0].set_ylabel('Original')
```

```
# Trend component
axes[1].plot(index, result.trend)
axes[1].set_ylabel('Trend')

# Seasonal component
axes[2].plot(index, result.seasonal)
axes[2].set_ylabel('Seasonal')

# Residual (irregular) component
axes[3].plot(index, result.resid)
axes[3].set_ylabel('Residual')

# Adjust the layout
plt.tight_layout()

# Show the plot
plt.show()
```

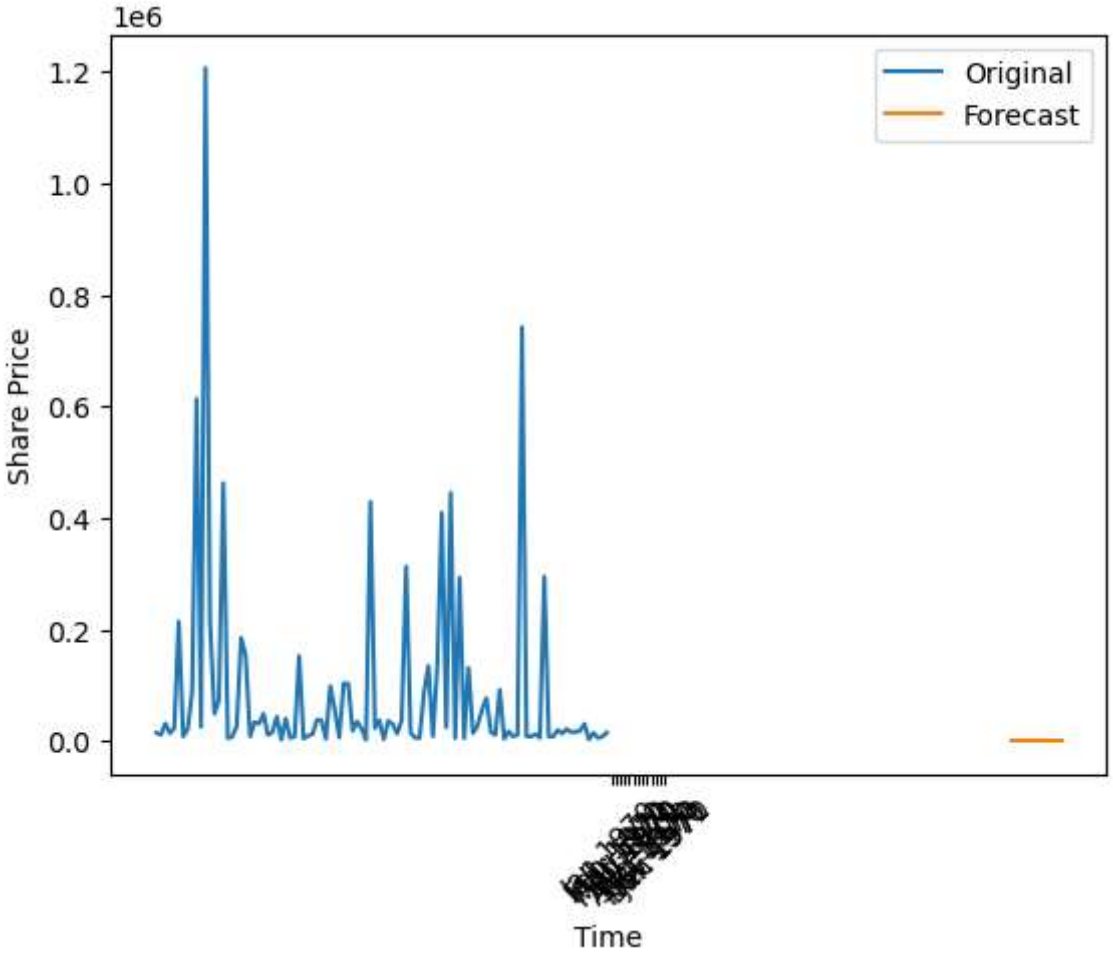


In [47]:
```
model = ARIMA(SMA['Name'] == 'Apple Inc.', order=(1, 1, 1))  # ARIMA(1, 1, 1) model
model_fit = model.fit()
```

In [48]:
```
forecast = model_fit.forecast(steps=12)
```

In [49]:
```
plt.plot(index, (numeric_data['Last Sale']), label='Original')
plt.plot(index[-12:] + len(numeric_data), forecast, label='Forecast')

months = pd.date_range(start=numeric_data.index[-1], periods=13, freq='M').strftime('%
```

```
plt.xticks(np.arange(len(index), len(index) + 13), months, rotation=45)
plt.xlabel('Time')
plt.ylabel('Share Price')
plt.legend()

plt.show()
```



Dickey - Fuller Test

In [50]:
```
apple_stock = (SMA['Name'] == 'Apple Inc.')
```

In [51]:
```
result = adfuller(apple_stock)
```

```
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\regression\linear_model.py:92
4: RuntimeWarning: divide by zero encountered in log
  llf = -nobs2*np.log(2*np.pi) - nobs2*np.log(ssr / nobs) - nobs2
```

In [52]:
```
adf_statistic = result[0]
p_value = result[1]

print(f'ADF Statistic: {adf_statistic}')
print(f'p-value: {p_value}')

if p_value < 0.05:
    print('The time series is stationary.')
else:
    print('The time series is not stationary.')
```

```
ADF Statistic: -8.5116450262455e+16
p-value: 0.0
The time series is stationary.
```

In [53]:
```
SMA_2 = SMA.dropna(axis = 1)
```

In [54]:
```
SMA_2.shape
```

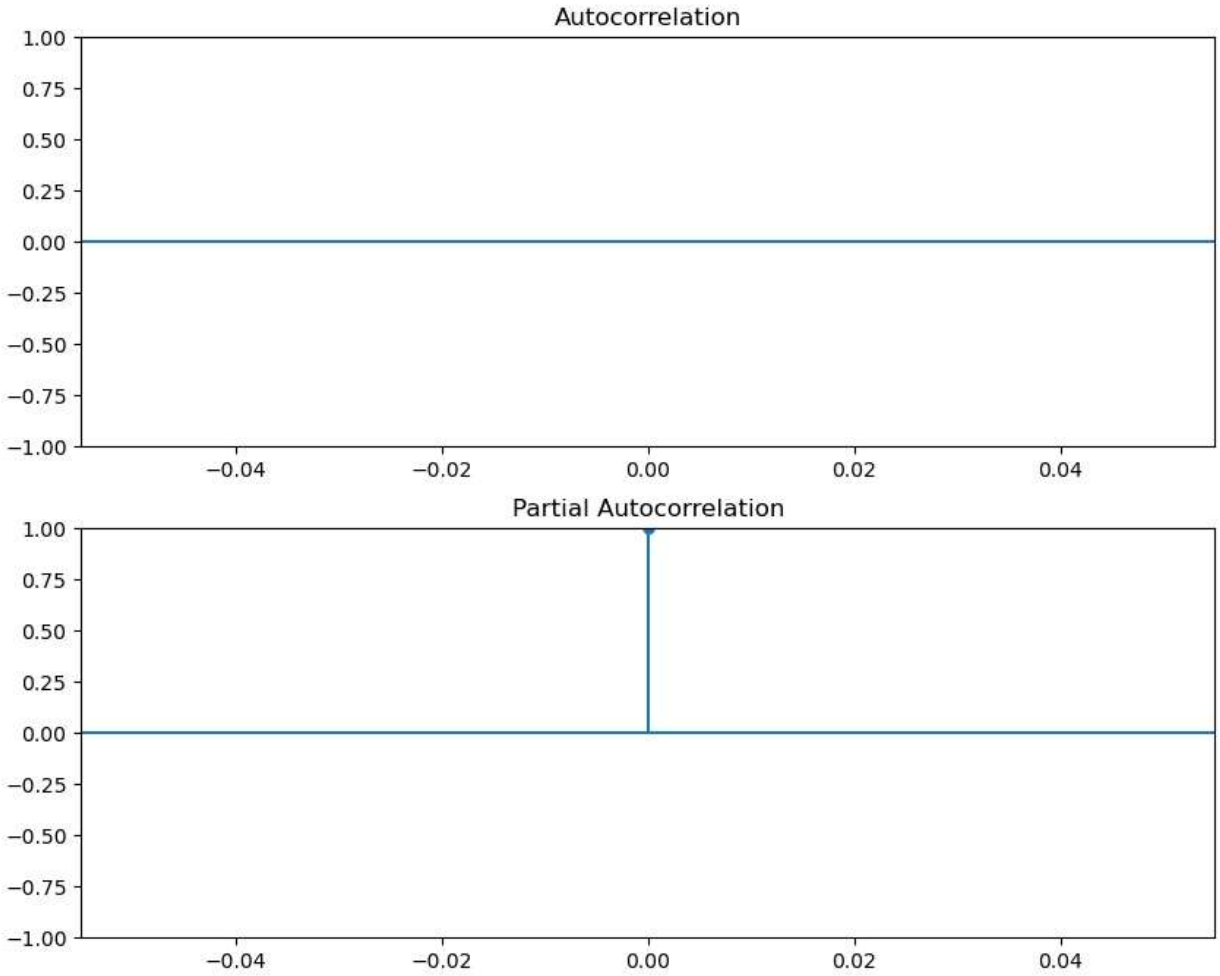
Out[54]:
```
(102, 56)
```

ACF and PACF

In [55]:
```
apple_stock = (SMA_2['Name'] == 'Apple Inc.')
apple_stock = pd.Series(apple_stock, index=pd.to_datetime(SMA_2.index))
```

In [56]:
```
fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
plot_acf(apple_stock, ax=ax1, lags=50)
plot_pacf(apple_stock, ax=ax2, lags=50)
plt.show()
```

```
C:\Users\naray\anaconda3\lib\site-packages\matplotlib\axes\_base.py:2480: UserWarnin
g: Warning: converting a masked element to nan.
  xys = np.asarray(xys)
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.py:348: Futu
reWarning: The default method 'yw' can produce PACF values outside of the [-1,1] inte
rval. After 0.13, the default will change tounadjusted Yule-Walker ('ywm'). You can u
se this method now by setting method='ywm'.
  warnings.warn(
```



In [57]:
```
p = 1
q = 0
d = 1
```

In [58]:
```
apple_stock = pd.to_numeric(apple_stock, errors='coerce')
```

In [59]:
```
model = ARIMA(apple_stock, order=(p, d, q))
model_fit = model.fit()
```

```
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency N will be use
d.
  self._init_dates(dates, freq)
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency N will be use
d.
  self._init_dates(dates, freq)
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency N will be use
d.
  self._init_dates(dates, freq)
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\tsa\statespace\sarimax.py:866:
UserWarning: Too few observations to estimate starting parameters for ARMA and trend.
All parameters except for variances will be set to zeros.
  warn('Too few observations to estimate starting parameters%s.'
C:\Users\naray\anaconda3\lib\site-packages\numpy\core\fromnumeric.py:3723: RuntimeWar
ning: Degrees of freedom <= 0 for slice
  return _methods._var(a, axis=axis, dtype=dtype, out=out, ddof=ddof,
C:\Users\naray\anaconda3\lib\site-packages\numpy\core\_methods.py:222: RuntimeWarnin
g: invalid value encountered in true_divide
  arrmean = um.true_divide(arrmean, div, out=arrmean, casting='unsafe',
C:\Users\naray\anaconda3\lib\site-packages\numpy\core\_methods.py:254: RuntimeWarnin
g: invalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
C:\Users\naray\anaconda3\lib\site-packages\statsmodels\base\model.py:604: Convergence
Warning: Maximum Likelihood optimization failed to converge. Check mle_retvals
  warnings.warn("Maximum Likelihood optimization failed to "
```

In [60]:
```
forecast = model_fit.forecast(steps=12)
forecast
```

Out[60]:
```
1970-01-01 00:00:00.000000102    0.0
1970-01-01 00:00:00.000000103    0.0
1970-01-01 00:00:00.000000104    0.0
1970-01-01 00:00:00.000000105    0.0
1970-01-01 00:00:00.000000106    0.0
1970-01-01 00:00:00.000000107    0.0
1970-01-01 00:00:00.000000108    0.0
1970-01-01 00:00:00.000000109    0.0
1970-01-01 00:00:00.000000110    0.0
1970-01-01 00:00:00.000000111    0.0
1970-01-01 00:00:00.000000112    0.0
1970-01-01 00:00:00.000000113    0.0
Freq: N, Name: predicted_mean, dtype: float64
```

MAPE

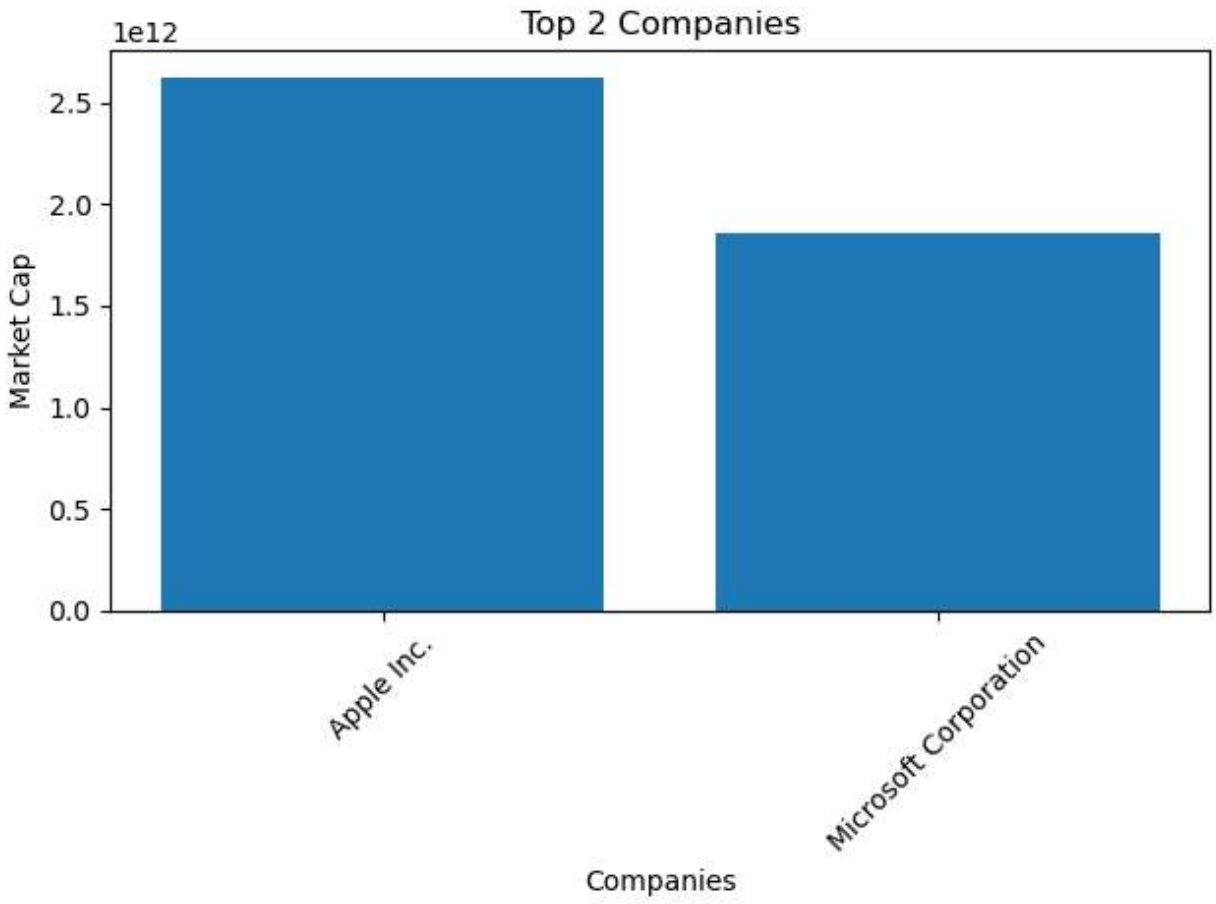
In [61]:
```
actual_values = apple_stock[-12:]
forecast_values = model_fit.forecast(steps=12)[0]
absolute_percentage_errors = abs((actual_values - forecast_values) / actual_values)
mape = absolute_percentage_errors.mean()

print("Mean Absolute Percentage Error (MAPE): {:.2%}".format(mape))
```

```
Mean Absolute Percentage Error (MAPE): nan%
```

In [72]:
```
sorted_SMA = SMA.sort_values(by='Market Cap', ascending=False)
```

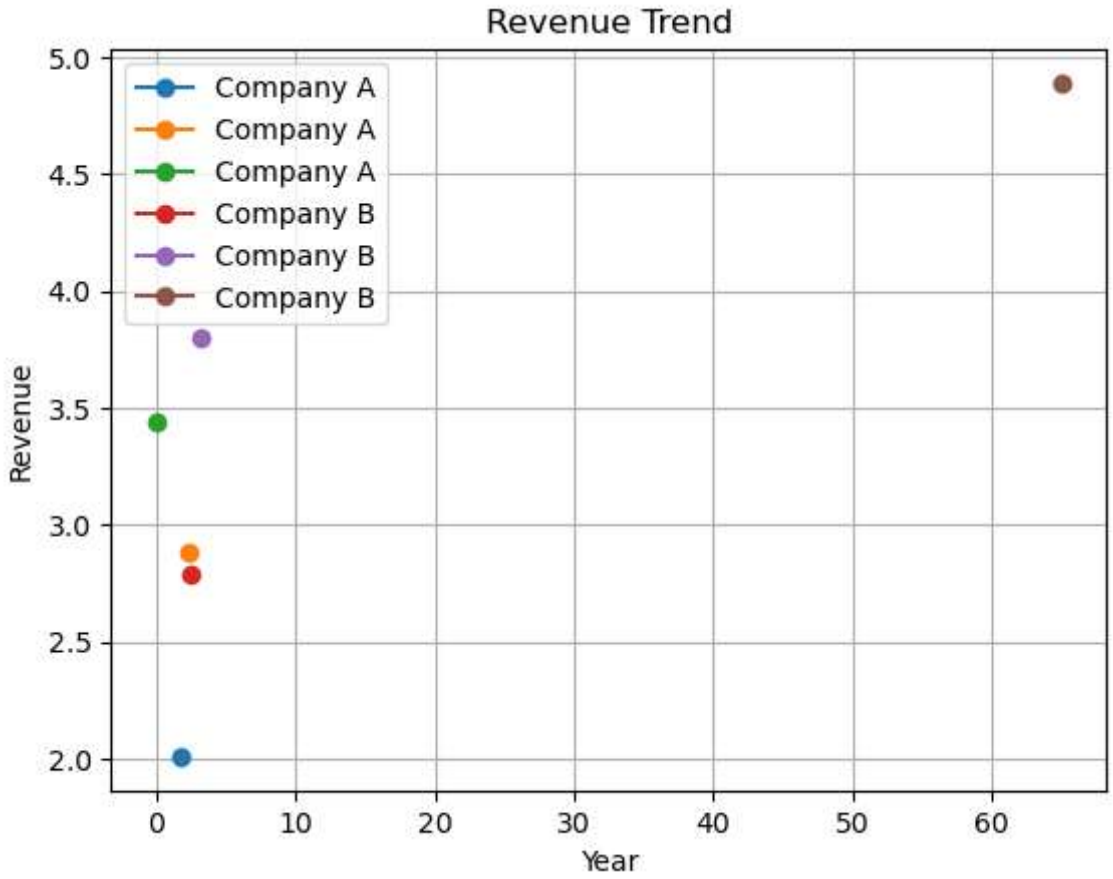
In [74]:
```
# Select the top 2 companies
top_2_companies = sorted_SMA.head(2)
```

In [75]:
```
# Create a bar graph to visualize the criterion values for the top 2 companies
plt.bar(top_2_companies['Name'], top_2_companies['Market Cap'])
plt.title('Top 2 Companies')
plt.xlabel('Companies')
plt.ylabel('Market Cap')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```



In [76]:
```
company_a_data = SMA[SMA['Name'] == 'Apple Inc.']
company_b_data = SMA[SMA['Name'] == 'Microsoft Corporation']
```

In [101…
```
plt.plot(company_a_data['e10_2018'], company_a_data['e10_2019'], company_a_data['e10_2
plt.plot(company_b_data['e10_2018'], company_b_data['e10_2019'], company_b_data['e10_2

# Customize the chart
plt.title('Revenue Trend')
plt.xlabel('Year')
plt.ylabel('Revenue')
plt.legend()
plt.grid(True)

# Show the chart
plt.show()
```



In [105…

```
# Perform ADF test for each company's revenue data
def perform_adf_test(SMA):
    result = adfuller(SMA)
    print('ADF Statistic:', result[0])
    print('p-value:', result[1])
    print('Critical Values:')
    for key, value in result[4].items():
        print('\t', key, ':', value)

print('ADF Test Results for Company A:')
perform_adf_test([company_a_data['e10_2018'], company_a_data['e10_2019'], company_a_da
print()

print('ADF Test Results for Company B:')
perform_adf_test([company_b_data['e10_2018'], company_b_data['e10_2019'], company_b_da
```

```
ADF Test Results for Company A:
ADF Statistic: 1.9750371594266611
p-value: 0.9986396233033543
Critical Values:
	 1% : -7.355440625
	 5% : -4.474365000000001
	 10% : -3.1269325

ADF Test Results for Company B:
ADF Statistic: 4.953922573780875
p-value: 1.0
Critical Values:
	 1% : -7.355440625
	 5% : -4.474365000000001
	 10% : -3.1269325
```

In [181…
```
# Group the data by sector and sort the companies based on market capitalization
grouped_data = SMA.groupby('sector')
top_2_companies = []
for sector, group in grouped_data:
    sorted_group = group.sort_values(by='Market Cap', ascending=False)
    top_2_companies.extend(sorted_group.head(2)['company'].tolist())
```

In [179…
```
numeric_data['Name'] = SMA['Name']
numeric_data.head()
```

Out[179]:

	Market Cap	Last Sale	Net Change	Percentage Change	asset_turnover_2021	asset_turnover_latest	buyback_yie
0	2625740143000	15145	200	0.0134	1.08	0.24	
1	69569944167	11665	26	0.0022	0.50	0.12	
2	149144569000	32081	459	0.0145	0.61	0.17	
3	75484763090	14676	223	0.0154	0.20	0.06	
4	98332762096	23678	13	-0.0005	0.33	0.06	

5 rows × 52 columns

In [190…
```
# Perform batch forecasting and calculate MAPE for each company
mape_results = {}
for company in top_2_companies:
    company_data = numeric_data[numeric_data['Name'] == 'Apple Inc.'].copy()
```

In [215…
```
# Separate the share price columns
share_price_columns = ['e10_2018', 'e10_2019', 'e10_2020', 'e10_2021', 'e10_latest']
company_data2 = SMA[share_price_columns]
```

In [216…
```
 # Split the data into training and testing sets
train_data = company_data2.iloc[:1]
test_data = company_data2.iloc[-1:]
```

In [217…
```
# Perform Auto ARIMA forecasting
model = auto_arima(train_data.values.flatten(), seasonal=False)
forecast = model.predict(n_periods=12)
```

In [218…
```
print(forecast)
```

```
[3.47149837 3.28649285 3.13769849 3.01802765 2.92178001 2.84437092
 2.78211312 2.73204104 2.69176958 2.65938046 2.63333086 2.61237995]
```

In [ ]: