Assignment4

February 25, 2024

[4]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from scipy.stats import norm
from tabulate import tabulate
import time
import psutil

# Set up notebook to display multiple outputs in one cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"
```

1 Import Fund Data

[5]:
```
start_time = time.time()

data_active_1 = pd.read_csv('PCCOX.csv')
data_active_2 = pd.read_csv('PRILX.csv')
data_active_3 = pd.read_csv('RWMGX.csv')
data_passive = pd.read_csv('WFSPX.csv')
data_active_1.head()
data_active_1.info()
data_active_2.head()
data_active_2.info()
data_active_3.head()
data_active_3.info()
data_passive.head()
data_passive.info()

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
```

```python
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

[5]:

```
         Date       Open       High        Low      Close  Adj Close  Volume
0  11/29/2016  23.879999  23.879999  23.879999  23.879999  19.379829     0.0
1  11/30/2016  23.770000  23.770000  23.770000  23.770000  19.290562     0.0
2   12/1/2016  23.639999  23.639999  23.639999  23.639999  19.185059     0.0
3   12/2/2016  23.660000  23.660000  23.660000  23.660000  19.201288     0.0
4   12/5/2016        NaN        NaN        NaN        NaN        NaN     NaN

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1816 entries, 0 to 1815
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       1816 non-null   object
 1   Open       1815 non-null   float64
 2   High       1815 non-null   float64
 3   Low        1815 non-null   float64
 4   Close      1815 non-null   float64
 5   Adj Close  1815 non-null   float64
 6   Volume     1815 non-null   float64
dtypes: float64(6), object(1)
memory usage: 99.4+ KB
```

[5]:

```
        Date       Open       High        Low      Close  Adj Close  Volume
0  4/28/2006  25.590000  25.590000  25.590000  25.590000   9.497851       0
1   5/1/2006  25.510000  25.510000  25.510000  25.510000   9.468159       0
2   5/2/2006  25.580000  25.580000  25.580000  25.580000   9.494144       0
3   5/3/2006  25.610001  25.610001  25.610001  25.610001   9.505277       0
4   5/4/2006  25.670000  25.670000  25.670000  25.670000   9.527547       0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4482 entries, 0 to 4481
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       4482 non-null   object
 1   Open       4482 non-null   float64
 2   High       4482 non-null   float64
 3   Low        4482 non-null   float64
 4   Close      4482 non-null   float64
 5   Adj Close  4482 non-null   float64
 6   Volume     4482 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 245.2+ KB
```

[5]:

```
       Date       Open       High        Low      Close  Adj Close  Volume
0  5/1/2009  19.950001  19.950001  19.950001  19.950001   8.865779       0
1  5/4/2009  20.580000  20.580000  20.580000  20.580000   9.145751       0
2  5/5/2009  20.520000  20.520000  20.520000  20.520000   9.119089       0
3  5/6/2009  20.790001  20.790001  20.790001  20.790001   9.239075       0
4  5/7/2009  20.549999  20.549999  20.549999  20.549999   9.132417       0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3725 entries, 0 to 3724
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       3725 non-null   object
 1   Open       3725 non-null   float64
 2   High       3725 non-null   float64
 3   Low        3725 non-null   float64
 4   Close      3725 non-null   float64
 5   Adj Close  3725 non-null   float64
 6   Volume     3725 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 203.8+ KB
```

[5]:

```
       Date       Open       High        Low      Close  Adj Close  Volume
0  7/2/1993  80.000000  80.000000  80.000000  80.000000  19.307232       0
1  7/6/1993  79.199997  79.199997  79.199997  79.199997  19.114147       0
2  7/7/1993  79.440002  79.440002  79.440002  79.440002  19.172077       0
3  7/8/1993  80.480003  80.480003  80.480003  80.480003  19.423063       0
4  7/9/1993  80.400002  80.400002  80.400002  80.400002  19.403767       0

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7712 entries, 0 to 7711
Data columns (total 7 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Date       7712 non-null   object
 1   Open       7712 non-null   float64
 2   High       7712 non-null   float64
 3   Low        7712 non-null   float64
 4   Close      7712 non-null   float64
 5   Adj Close  7712 non-null   float64
 6   Volume     7712 non-null   int64
dtypes: float64(5), int64(1), object(1)
memory usage: 421.9+ KB
Execution Time: 0.055719852447509766 seconds
Memory Usage: 201.046875 MB
```

1.1 Fund Transformation

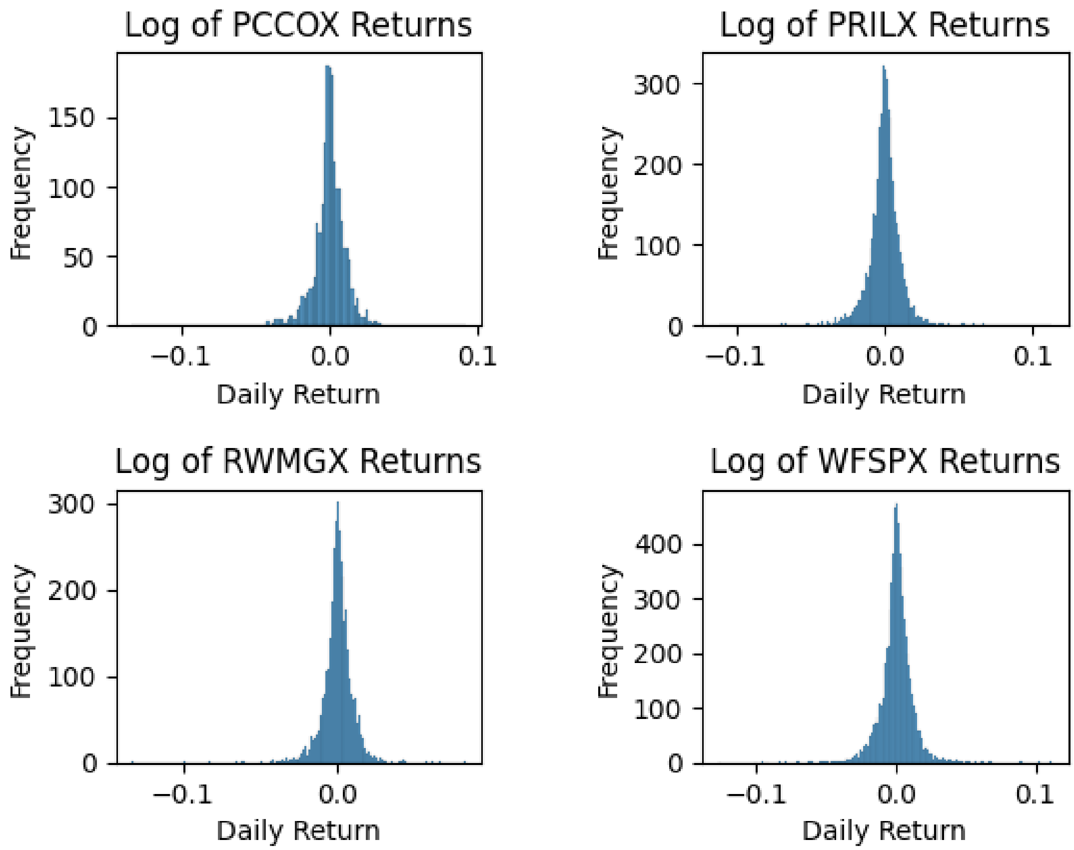
[6]:
```python
start_time = time.time()

# compute the logarithmic returns of each of the funds
log_return_active_1 = np.log(1 + data_active_1['Adj Close'].pct_change())
log_return_active_2 = np.log(1 + data_active_2['Adj Close'].pct_change())
log_return_active_3 = np.log(1 + data_active_3['Adj Close'].pct_change())
log_return_passive = np.log(1 + data_passive['Adj Close'].pct_change())

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
Execution Time: 0.005049705505371094 seconds
Memory Usage: 201.21875 MB
```

[7]:
```python
start_time = time.time()

# plot the log returns

fig = plt.figure()
fig.subplots_adjust(hspace=0.6, wspace=0.6)

ax = fig.add_subplot(2, 2, 1)
sns.histplot(log_return_active_1.iloc[1:], ax=ax)
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.title('Log of PCCOX Returns')

ax = fig.add_subplot(2, 2, 2)
sns.histplot(log_return_active_2.iloc[1:], ax=ax)
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.title('Log of PRILX Returns')

ax = fig.add_subplot(2, 2, 3)
sns.histplot(log_return_active_3.iloc[1:], ax=ax)
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.title('Log of RWMGX Returns')

ax = fig.add_subplot(2, 2, 4)
```

```python
sns.histplot(log_return_passive.iloc[1:], ax=ax)
plt.xlabel('Daily Return')
plt.ylabel('Frequency')
plt.title('Log of WFSPX Returns')

plt.show()

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
[7]: <AxesSubplot: xlabel='Adj Close', ylabel='Count'>

[7]: Text(0.5, 0, 'Daily Return')

[7]: Text(0, 0.5, 'Frequency')

[7]: Text(0.5, 1.0, 'Log of PCCOX Returns')

[7]: <AxesSubplot: xlabel='Adj Close', ylabel='Count'>

[7]: Text(0.5, 0, 'Daily Return')

[7]: Text(0, 0.5, 'Frequency')

[7]: Text(0.5, 1.0, 'Log of PRILX Returns')

[7]: <AxesSubplot: xlabel='Adj Close', ylabel='Count'>

[7]: Text(0.5, 0, 'Daily Return')

[7]: Text(0, 0.5, 'Frequency')

[7]: Text(0.5, 1.0, 'Log of RWMGX Returns')

[7]: <AxesSubplot: xlabel='Adj Close', ylabel='Count'>

[7]: Text(0.5, 0, 'Daily Return')

[7]: Text(0, 0.5, 'Frequency')

[7]: Text(0.5, 1.0, 'Log of WFSPX Returns')
```




```
Execution Time: 0.7213199138641357 seconds
Memory Usage: 214.15625 MB
```

2 Simulation

2.1 Compute Drift & Variance

[8]:

```
start_time = time.time()

# compute the drift

mean_active_1 = log_return_active_1.mean()
var_active_1 = log_return_active_1.var()
drift_active_1 = mean_active_1 - (0.5*var_active_1)

mean_active_2 = log_return_active_2.mean()
var_active_2 = log_return_active_2.var()
drift_active_2 = mean_active_2 - (0.5*var_active_2)

mean_active_3 = log_return_active_3.mean()
```

```
var_active_3 = log_return_active_3.var()
drift_active_3 = mean_active_3 - (0.5*var_active_3)

mean_passive = log_return_passive.mean()
var_passive = log_return_passive.var()
drift_passive = mean_passive - (0.5*var_passive)

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
Execution Time: 0.002657175064086914 seconds
Memory Usage: 214.375 MB
```

[9]:
```
start_time = time.time()

# compute the variance and daily returns

days = 251
trials = 10000

stdev_active_1 = log_return_active_1.std()
Z_active_1 = norm.ppf(np.random.rand(days, trials))
daily_returns_active_1 = np.exp(drift_active_1 + stdev_active_1 * Z_active_1)

stdev_active_2 = log_return_active_2.std()
Z_active_2 = norm.ppf(np.random.rand(days, trials))
daily_returns_active_2 = np.exp(drift_active_2 + stdev_active_2 * Z_active_2)

stdev_active_3 = log_return_active_3.std()
Z_active_3 = norm.ppf(np.random.rand(days, trials))
daily_returns_active_3 = np.exp(drift_active_3 + stdev_active_3 * Z_active_3)

stdev_passive = log_return_passive.std()
Z_passive  = norm.ppf(np.random.rand(days, trials))
daily_returns_passive  = np.exp(drift_passive  + stdev_passive  * Z_passive)

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")
```

```
# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
Execution Time: 0.3926398754119873 seconds
Memory Usage: 499.3125 MB
```

[10]:
```
daily_returns_active_1.shape
```

[10]:
```
(251, 10000)
```

2.2 Simulate Price Path for Trials

[11]:
```
start_time = time.time()

# calculate the stock price for every trial

price_paths_active_1 = np.zeros_like(daily_returns_active_1)
price_paths_active_1[0] = data_active_1.iloc[-1,5]
for t in range(1, days):
       price_paths_active_1[t] =␣
 ↪price_paths_active_1[t-1]*daily_returns_active_1[t]

price_paths_active_2 = np.zeros_like(daily_returns_active_2)
price_paths_active_2[0] = data_active_2.iloc[-1,5]
for t in range(1, days):
       price_paths_active_2[t] =␣
 ↪price_paths_active_2[t-1]*daily_returns_active_2[t]

price_paths_active_3 = np.zeros_like(daily_returns_active_3)
price_paths_active_3[0] = data_active_3.iloc[-1,5]
for t in range(1, days):
       price_paths_active_3[t] =␣
 ↪price_paths_active_3[t-1]*daily_returns_active_3[t]

price_paths_passive = np.zeros_like(daily_returns_passive)
price_paths_passive[0] = data_passive.iloc[-1,5]
for t in range(1, days):
       price_paths_passive[t] = price_paths_passive[t-1]*daily_returns_passive␣
 ↪[t]

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
```

```
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
Execution Time: 0.026353836059570312 seconds
Memory Usage: 474.515625 MB
```

[12]:
```
#inspect price path array
price_paths_active_1
price_paths_active_1.shape
```

[12]:
```
array([[49.459999  , 49.459999  , 49.459999  , …, 49.459999  ,
        49.459999  , 49.459999  ],
       [50.14205521, 49.51983857, 49.17869469, …, 49.26825776,
        49.81380337, 49.63207521],
       [50.25564066, 49.9294514 , 48.63492719, …, 50.09936557,
        49.67344739, 49.03971022],
       …,
       [49.71855146, 54.57873545, 67.7532875 , …, 61.92270379,
        71.10400142, 59.21382922],
       [49.37451314, 55.82243494, 67.36904542, …, 61.70600876,
        72.16193496, 59.83542652],
       [49.43625487, 55.23601294, 66.40528452, …, 61.99495001,
        71.8752812 , 61.12013617]])
```

[12]:
```
(251, 10000)
```

2.2.1 Price Path Arrays to Dataframes

[13]:
```
start_time = time.time()

#array to dataframe
num_columns = 251

#price path 1
df1 = pd.DataFrame(price_paths_active_1)
df1 = df1.T
df1.columns = [f'Day {i}' for i in range(1, num_columns + 1)]

#price path 2
df2 = pd.DataFrame(price_paths_active_2)
df2 = df2.T
df2.columns = [f'Day {i}' for i in range(1, num_columns + 1)]

#price path 3
df3 = pd.DataFrame(price_paths_active_3)
df3 = df3.T
df3.columns = [f'Day {i}' for i in range(1, num_columns + 1)]
```

```
#price path passive
df_passive = pd.DataFrame(price_paths_passive)
df_passive = df_passive.T
df_passive.columns = [f'Day {i}' for i in range(1, num_columns + 1)]

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")

#inspect dataframes
df1.head()
df1.info()
df2.head()
df2.info()
df3.head()
df3.info()
df_passive.head()
df_passive.info()
```

```
Execution Time: 0.009378910064697266 seconds
Memory Usage: 459.8125 MB
```

[13]:

```
       Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  49.459999  50.142055  50.255641  49.954029  50.244520  51.324730
1  49.459999  49.519839  49.929451  50.447951  51.108579  51.986112
2  49.459999  49.178695  48.634927  49.926565  51.252371  51.740776
3  49.459999  49.589906  50.480854  49.878121  50.637257  51.489881
4  49.459999  49.252438  49.813462  49.225565  48.172159  47.365233

       Day 7      Day 8      Day 9     Day 10  …    Day 242    Day 243  \
0  51.512793  51.898652  51.851680  51.925461  …  49.877431  50.023224
1  51.678181  52.377834  51.105925  50.639021  …  55.368464  54.748442
2  51.841784  51.740166  51.489128  52.777658  …  65.737701  66.101872
3  52.164876  52.179596  52.335114  52.021144  …  46.672065  46.878675
4  47.141077  47.270846  47.745082  48.444651  …  53.256937  53.599854

     Day 244    Day 245    Day 246    Day 247    Day 248    Day 249  \
0  49.919597  50.287670  49.961039  49.428941  49.228788  49.718551
1  55.088559  54.136698  54.512995  54.432817  54.668094  54.578735
2  66.382560  66.226224  65.743686  66.037811  67.738443  67.753287
3  46.774629  47.031453  47.528797  47.495387  47.426392  47.333364
4  53.723650  54.369297  53.830896  54.169037  52.974778  53.632303
```

```
     Day 250    Day 251
0  49.374513  49.436255
1  55.822435  55.236013
2  67.369045  66.405285
3  48.049830  48.333099
4  53.497597  53.100190

[5 rows x 251 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 251 entries, Day 1 to Day 251
dtypes: float64(251)
memory usage: 19.1 MB
```

[13]:

```
       Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  57.369999  56.736916  57.463402  57.427945  56.809069  56.592897
1  57.369999  57.153898  56.936941  56.998576  57.726618  58.264930
2  57.369999  57.332079  58.515451  58.062084  57.392513  57.842025
3  57.369999  57.860861  58.336171  58.506123  58.352762  58.547720
4  57.369999  56.487252  56.710821  57.545210  57.387592  56.448676

       Day 7      Day 8      Day 9     Day 10  …    Day 242    Day 243  \
0  57.642147  57.975870  58.144153  58.322589  …  67.878725  67.516306
1  59.572960  60.144333  60.996670  61.079078  …  66.575242  67.351102
2  58.118601  57.736659  58.006618  56.984477  …  64.903184  64.794017
3  59.042679  58.409335  58.964540  59.516963  …  74.305787  74.255289
4  57.018717  56.470174  56.630518  57.531361  …  76.211214  77.619138

     Day 244    Day 245    Day 246    Day 247    Day 248    Day 249  \
0  68.223211  68.079959  67.696580  66.909891  67.174185  66.424769
1  67.067401  67.533645  66.755985  66.815412  65.420927  65.451817
2  64.418688  63.946906  63.393734  64.729022  64.639764  65.854754
3  73.148148  73.483927  74.724817  74.894142  75.719815  76.686841
4  77.902740  77.242045  78.046764  79.003897  78.804506  79.453323

     Day 250    Day 251
0  65.879670  66.342836
1  66.576924  66.285637
2  66.174727  66.159152
3  78.546436  79.021308
4  79.915620  80.337336

[5 rows x 251 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
```

```
Columns: 251 entries, Day 1 to Day 251
dtypes: float64(251)
memory usage: 19.1 MB
```

[13]:

```
       Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  59.389999  60.707993  59.838719  59.785114  60.806134  61.193734
1  59.389999  60.166116  59.877287  60.130494  60.717470  59.314432
2  59.389999  60.220866  60.227533  60.834411  60.365568  61.330244
3  59.389999  59.000458  58.692505  58.248972  57.867721  57.318932
4  59.389999  60.061452  60.781550  59.982555  60.520402  60.875122

       Day 7      Day 8      Day 9     Day 10  …    Day 242    Day 243  \
0  61.442648  61.283836  61.876675  61.821764  …  70.486232  69.536005
1  59.338747  59.133606  58.700590  58.362560  …  65.486473  65.725207
2  61.977660  61.800100  60.958200  61.650128  …  70.300576  70.426631
3  56.769928  56.467460  56.820836  56.656224  …  54.330072  55.310252
4  60.118794  61.010521  60.276146  59.830078  …  73.592323  73.290657

     Day 244    Day 245    Day 246    Day 247    Day 248    Day 249  \
0  68.919543  69.640605  69.161296  68.633208  68.026000  68.926217
1  65.093343  65.942293  66.313873  66.143648  66.966856  67.566700
2  69.459533  68.463462  68.368954  68.107189  67.091223  66.875138
3  54.705369  54.863302  54.977028  54.623668  53.517801  53.602384
4  73.459376  73.332980  74.000925  74.497731  74.949035  74.442844

     Day 250    Day 251
0  68.914790  67.547719
1  68.073362  68.344322
2  66.912838  66.823846
3  52.565376  53.679401
4  75.666851  75.568713

[5 rows x 251 columns]
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 251 entries, Day 1 to Day 251
dtypes: float64(251)
memory usage: 19.1 MB
```

[13]:

```
        Day 1       Day 2       Day 3       Day 4       Day 5       Day 6  \
0  587.590027  589.490846  600.207762  599.753277  588.604102  584.376440
1  587.590027  591.834883  589.433102  589.273987  594.264130  593.161353
2  587.590027  581.393708  585.952983  585.042265  585.993372  576.807663
3  587.590027  587.561038  586.068964  576.291588  577.674929  570.339055
4  587.590027  595.269446  595.402728  600.838128  608.275495  612.568485

        Day 7       Day 8       Day 9      Day 10  …     Day 242  \
```

```
0  578.293380  586.972386  575.066475  586.793660  …  682.798269
1  593.101377  603.215052  605.532145  591.959675  …  690.593688
2  578.940671  588.673454  600.896339  595.495995  …  563.320508
3  559.871186  575.112003  577.431562  585.211297  …  733.725814
4  620.304767  619.019062  624.788094  630.919915  …  709.717050

      Day 243     Day 244     Day 245     Day 246     Day 247     Day 248  \
0  689.549165  686.281954  691.232239  684.092294  691.786492  718.169544
1  686.608866  688.113630  696.189327  691.828315  704.406626  704.640297
2  558.783147  552.472162  550.671652  556.681801  558.130579  565.676076
3  718.838284  740.699481  739.483087  741.929478  737.525436  748.017379
4  704.144017  707.872672  705.932440  706.499385  701.567848  716.892353

      Day 249     Day 250     Day 251
0  724.979082  717.545567  712.865037
1  704.480889  705.100904  698.333695
2  569.121609  570.626981  577.762869
3  742.978118  746.954925  743.852646
4  706.514307  688.701356  695.969641

[5 rows x 251 columns]

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Columns: 251 entries, Day 1 to Day 251
dtypes: float64(251)
memory usage: 19.1 MB
```

2.3 Calculate Return & Volatility

[14]:
```
start_time = time.time()

#PCCOX - data active 1
df1['Volatility'] = df1.std(axis = 1)
df1['Return'] = (df1['Day 251'] - df1['Day 1'])/df1['Day 1']

#PRILX - data active 2
df2['Volatility'] = df2.std(axis = 1)
df2['Return'] = (df2['Day 251'] - df2['Day 1'])/df2['Day 1']

#RWMGX - data active 3
df3['Volatility'] = df3.std(axis = 1)
df3['Return'] = (df3['Day 251'] - df3['Day 1'])/df3['Day 1']

#WFSPX - data passive
df_passive['Volatility'] = df_passive.std(axis = 1)
```

```
df_passive['Return'] = (df_passive['Day 251'] - df_passive['Day 1'])/
 ↪df_passive['Day 1']

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")

#inspect changes
df1.head()
df2.head()
df3.head()
df_passive.head()
```

```
Execution Time: 0.056699275970458984 seconds
Memory Usage: 410.640625 MB
```

[14]:

```
       Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  49.459999  50.142055  50.255641  49.954029  50.244520  51.324730
1  49.459999  49.519839  49.929451  50.447951  51.108579  51.986112
2  49.459999  49.178695  48.634927  49.926565  51.252371  51.740776
3  49.459999  49.589906  50.480854  49.878121  50.637257  51.489881
4  49.459999  49.252438  49.813462  49.225565  48.172159  47.365233

       Day 7      Day 8      Day 9     Day 10  …    Day 244    Day 245  \
0  51.512793  51.898652  51.851680  51.925461  …  49.919597  50.287670
1  51.678181  52.377834  51.105925  50.639021  …  55.088559  54.136698
2  51.841784  51.740166  51.489128  52.777658  …  66.382560  66.226224
3  52.164876  52.179596  52.335114  52.021144  …  46.774629  47.031453
4  47.141077  47.270846  47.745082  48.444651  …  53.723650  54.369297

     Day 246    Day 247    Day 248    Day 249    Day 250    Day 251  \
0  49.961039  49.428941  49.228788  49.718551  49.374513  49.436255
1  54.512995  54.432817  54.668094  54.578735  55.822435  55.236013
2  65.743686  66.037811  67.738443  67.753287  67.369045  66.405285
3  47.528797  47.495387  47.426392  47.333364  48.049830  48.333099
4  53.830896  54.169037  52.974778  53.632303  53.497597  53.100190

   Volatility    Return
0    3.947858 -0.000480
1    2.556718  0.116782
2    4.621787  0.342606
3    2.353785 -0.022784
```

```
4    3.221275  0.073599

[5 rows x 253 columns]
```

[14]:

```
      Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  57.369999  56.736916  57.463402  57.427945  56.809069  56.592897
1  57.369999  57.153898  56.936941  56.998576  57.726618  58.264930
2  57.369999  57.332079  58.515451  58.062084  57.392513  57.842025
3  57.369999  57.860861  58.336171  58.506123  58.352762  58.547720
4  57.369999  56.487252  56.710821  57.545210  57.387592  56.448676

      Day 7      Day 8      Day 9     Day 10  …    Day 244    Day 245  \
0  57.642147  57.975870  58.144153  58.322589  …  68.223211  68.079959
1  59.572960  60.144333  60.996670  61.079078  …  67.067401  67.533645
2  58.118601  57.736659  58.006618  56.984477  …  64.418688  63.946906
3  59.042679  58.409335  58.964540  59.516963  …  73.148148  73.483927
4  57.018717  56.470174  56.630518  57.531361  …  77.902740  77.242045

    Day 246    Day 247    Day 248    Day 249    Day 250    Day 251  \
0  67.696580  66.909891  67.174185  66.424769  65.879670  66.342836
1  66.755985  66.815412  65.420927  65.451817  66.576924  66.285637
2  63.393734  64.729022  64.639764  65.854754  66.174727  66.159152
3  74.724817  74.894142  75.719815  76.686841  78.546436  79.021308
4  78.046764  79.003897  78.804506  79.453323  79.915620  80.337336

   Volatility    Return
0    5.343291  0.156403
1    3.552965  0.155406
2    3.627888  0.153201
3    6.553574  0.377398
4    7.672104  0.400337

[5 rows x 253 columns]
```

[14]:

```
      Day 1      Day 2      Day 3      Day 4      Day 5      Day 6  \
0  59.389999  60.707993  59.838719  59.785114  60.806134  61.193734
1  59.389999  60.166116  59.877287  60.130494  60.717470  59.314432
2  59.389999  60.220866  60.227533  60.834411  60.365568  61.330244
3  59.389999  59.000458  58.692505  58.248972  57.867721  57.318932
4  59.389999  60.061452  60.781550  59.982555  60.520402  60.875122

      Day 7      Day 8      Day 9     Day 10  …    Day 244    Day 245  \
0  61.442648  61.283836  61.876675  61.821764  …  68.919543  69.640605
1  59.338747  59.133606  58.700590  58.362560  …  65.093343  65.942293
2  61.977660  61.800100  60.958200  61.650128  …  69.459533  68.463462
3  56.769928  56.467460  56.820836  56.656224  …  54.705369  54.863302
4  60.118794  61.010521  60.276146  59.830078  …  73.459376  73.332980
```

```
    Day 246    Day 247    Day 248    Day 249    Day 250    Day 251  \
0  69.161296  68.633208  68.026000  68.926217  68.914790  67.547719
1  66.313873  66.143648  66.966856  67.566700  68.073362  68.344322
2  68.368954  68.107189  67.091223  66.875138  66.912838  66.823846
3  54.977028  54.623668  53.517801  53.602384  52.565376  53.679401
4  74.000925  74.497731  74.949035  74.442844  75.666851  75.568713

   Volatility    Return
0    4.535004  0.137358
1    5.405600  0.150772
2    4.020033  0.125170
3    2.401950 -0.096154
4    5.621738  0.272415

[5 rows x 253 columns]
```

[14]:

```
        Day 1       Day 2       Day 3       Day 4       Day 5       Day 6  \
0  587.590027  589.490846  600.207762  599.753277  588.604102  584.376440
1  587.590027  591.834883  589.433102  589.273987  594.264130  593.161353
2  587.590027  581.393708  585.952983  585.042265  585.993372  576.807663
3  587.590027  587.561038  586.068964  576.291588  577.674929  570.339055
4  587.590027  595.269446  595.402728  600.838128  608.275495  612.568485

        Day 7       Day 8       Day 9      Day 10  …     Day 244  \
0  578.293380  586.972386  575.066475  586.793660  …  686.281954
1  593.101377  603.215052  605.532145  591.959675  …  688.113630
2  578.940671  588.673454  600.896339  595.495995  …  552.472162
3  559.871186  575.112003  577.431562  585.211297  …  740.699481
4  620.304767  619.019062  624.788094  630.919915  …  707.872672

      Day 245     Day 246     Day 247     Day 248     Day 249     Day 250  \
0  691.232239  684.092294  691.786492  718.169544  724.979082  717.545567
1  696.189327  691.828315  704.406626  704.640297  704.480889  705.100904
2  550.671652  556.681801  558.130579  565.676076  569.121609  570.626981
3  739.483087  741.929478  737.525436  748.017379  742.978118  746.954925
4  705.932440  706.499385  701.567848  716.892353  706.514307  688.701356

      Day 251  Volatility    Return
0  712.865037   52.022360  0.213201
1  698.333695   42.111909  0.188471
2  577.762869   17.353598 -0.016725
3  743.852646   50.990507  0.265938
4  695.969641   29.529557  0.184448

[5 rows x 253 columns]
```

3 Calculate Average Return

[15]:
```
start_time = time.time()

PCCOX_returns = df1['Return'].mean()
PRILX_returns = df2['Return'].mean()
RWMGX_returns = df3['Return'].mean()
WFSPX_returns = df_passive['Return'].mean()

#create return table
table = [['Fund', 'Avg. Annual Return'],
         ['PCCOX', PCCOX_returns],
         ['PRILX', PRILX_returns],
         ['RWMGX', RWMGX_returns],
         ['WFSPX', WFSPX_returns]]

print(tabulate(table, headers='firstrow', tablefmt='grid'))

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
+--------+----------------------+
| Fund   |   Avg. Annual Return |
+========+======================+
| PCCOX  |             0.132947 |
+--------+----------------------+
| PRILX  |             0.105485 |
+--------+----------------------+
| RWMGX  |             0.135522 |
+--------+----------------------+
| WFSPX  |             0.116739 |
+--------+----------------------+
Execution Time: 0.005060195922851562 5 seconds
Memory Usage: 411.3125 MB
```

[16]:
```
#verify return values
PCCOX_returns
PRILX_returns
RWMGX_returns
WFSPX_returns
```

[16]: 0.13294700306556498

[16]: 0.10548515645859631

[16]: 0.1355216750564024

[16]: 0.11673880819466294

4 Visualizations

[17]:
```
#plot return distributions

fig = plt.figure()
fig.subplots_adjust(hspace=0.8, wspace=0.8)

ax = fig.add_subplot(2, 2, 1)
sns.histplot(pd.DataFrame(price_paths_active_1).iloc[-1], ax=ax)
plt.xlabel("Price after 251 days")
plt.title('PCCOX Returns')

ax = fig.add_subplot(2, 2, 2)
sns.histplot(pd.DataFrame(price_paths_active_2).iloc[-1], ax=ax)
plt.xlabel("Price after 251 days")
plt.title('PRILX Returns')

ax = fig.add_subplot(2, 2, 3)
sns.histplot(pd.DataFrame(price_paths_active_3).iloc[-1], ax=ax)
plt.xlabel("Price after 251 days")
plt.title('RWMGX Returns')

ax = fig.add_subplot(2, 2, 4)
sns.histplot(pd.DataFrame(price_paths_passive).iloc[-1], ax=ax)
plt.xlabel("Price after 251 days")
plt.title('WFSPX Returns')

plt.show()
```

[17]: <AxesSubplot: xlabel='250', ylabel='Count'>

[17]: Text(0.5, 0, 'Price after 251 days')

[17]: Text(0.5, 1.0, 'PCCOX Returns')

[17]: <AxesSubplot: xlabel='250', ylabel='Count'>

[17]: Text(0.5, 0, 'Price after 251 days')

[17]: Text(0.5, 1.0, 'PRILX Returns')

[17]: <AxesSubplot: xlabel='250', ylabel='Count'>

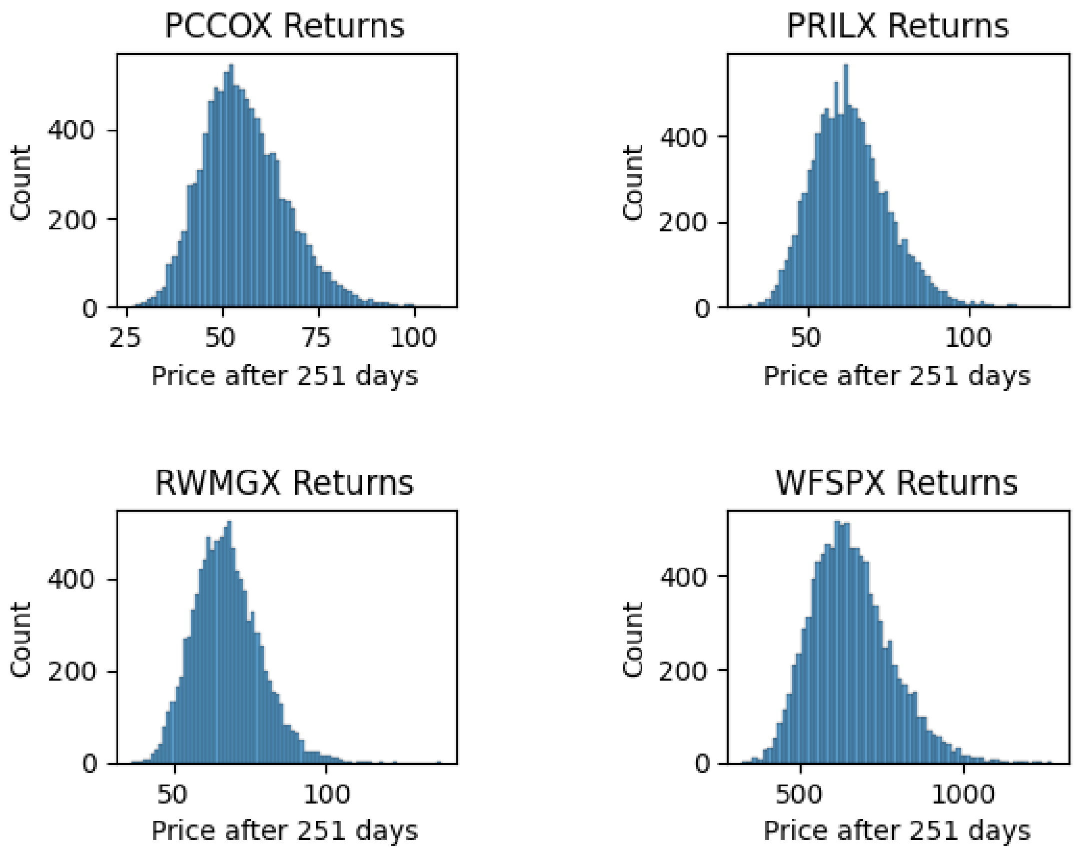
[17]: Text(0.5, 0, 'Price after 251 days')

[17]: Text(0.5, 1.0, 'RWMGX Returns')

[17]: <AxesSubplot: xlabel='250', ylabel='Count'>

[17]: Text(0.5, 0, 'Price after 251 days')

[17]: Text(0.5, 1.0, 'WFSPX Returns')



4.1 PCCOX

[18]:
```python
#plot 20 price paths
plt.figure(figsize=(15,6))
plt.plot(pd.DataFrame(price_paths_active_1).iloc[:,0:20])
plt.title("Simulated PCCOX Prices")
plt.xlabel("Days")
plt.ylabel("Price ($)")
```

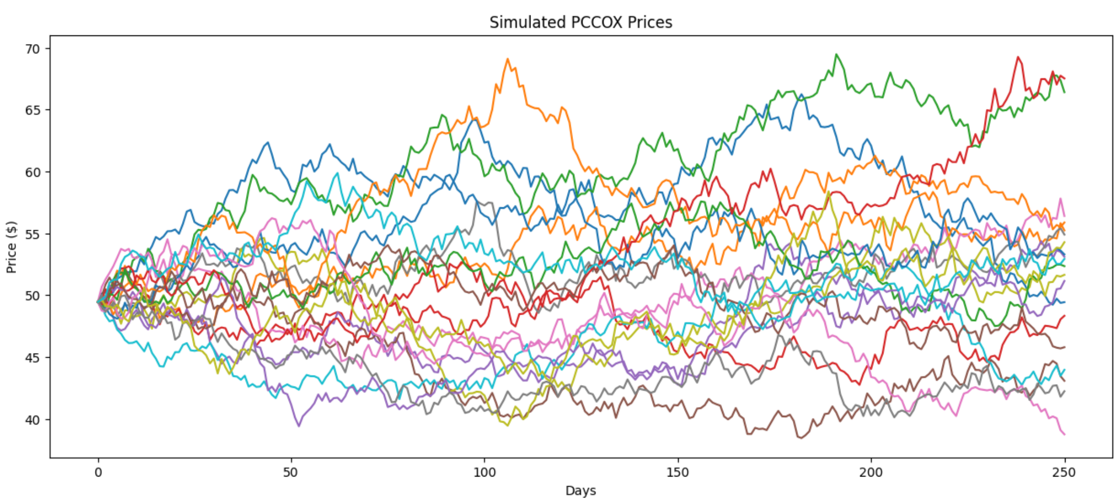
[18]: <Figure size 1500x600 with 0 Axes>

```
[18]: [<matplotlib.lines.Line2D at 0x28fd91990>,
 <matplotlib.lines.Line2D at 0x28fd919f0>,
 <matplotlib.lines.Line2D at 0x28fd91a20>,
 <matplotlib.lines.Line2D at 0x28fd91b10>,
 <matplotlib.lines.Line2D at 0x28fd91c00>,
 <matplotlib.lines.Line2D at 0x28fd91cf0>,
 <matplotlib.lines.Line2D at 0x28fd91de0>,
 <matplotlib.lines.Line2D at 0x28fd91ed0>,
 <matplotlib.lines.Line2D at 0x28fd91fc0>,
 <matplotlib.lines.Line2D at 0x28fd920b0>,
 <matplotlib.lines.Line2D at 0x28fd921a0>,
 <matplotlib.lines.Line2D at 0x28fd919c0>,
 <matplotlib.lines.Line2D at 0x28fd92290>,
 <matplotlib.lines.Line2D at 0x28fd92440>,
 <matplotlib.lines.Line2D at 0x28fd92530>,
 <matplotlib.lines.Line2D at 0x28fd92620>,
 <matplotlib.lines.Line2D at 0x28fd92710>,
 <matplotlib.lines.Line2D at 0x28fd92800>,
 <matplotlib.lines.Line2D at 0x28fd928f0>,
 <matplotlib.lines.Line2D at 0x28fd929e0>]
```

```
[18]: Text(0.5, 1.0, 'Simulated PCCOX Prices')
```

```
[18]: Text(0.5, 0, 'Days')
```

```
[18]: Text(0, 0.5, 'Price ($)')
```



4.2 PRILX

[19]:
```python
#plot 20 price paths
plt.figure(figsize=(15,6))
plt.plot(pd.DataFrame(price_paths_active_2).iloc[:,0:20])
plt.title("Simulated PRILX Prices")
plt.xlabel("Days")
plt.ylabel("Price ($)")
```

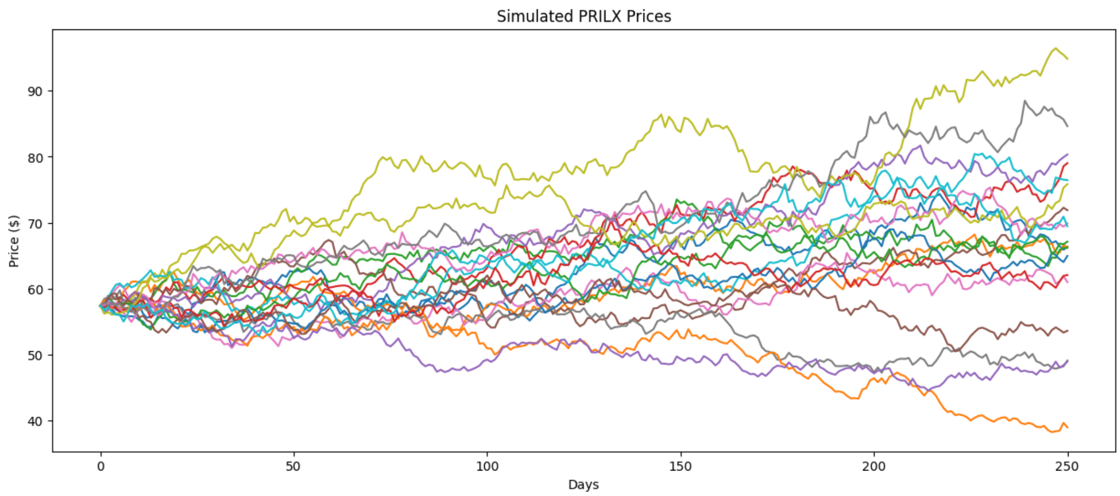
[19]:
```
<Figure size 1500x600 with 0 Axes>
```

[19]:
```
[<matplotlib.lines.Line2D at 0x28fe2fc70>,
 <matplotlib.lines.Line2D at 0x28fe2fcd0>,
 <matplotlib.lines.Line2D at 0x28fe2fd00>,
 <matplotlib.lines.Line2D at 0x28fe2fdf0>,
 <matplotlib.lines.Line2D at 0x28fe2fee0>,
 <matplotlib.lines.Line2D at 0x28fe2ffd0>,
 <matplotlib.lines.Line2D at 0x28fe58100>,
 <matplotlib.lines.Line2D at 0x28fe581f0>,
 <matplotlib.lines.Line2D at 0x28fe582e0>,
 <matplotlib.lines.Line2D at 0x28fe583d0>,
 <matplotlib.lines.Line2D at 0x28fe2fca0>,
 <matplotlib.lines.Line2D at 0x28fe584c0>,
 <matplotlib.lines.Line2D at 0x28fe585b0>,
 <matplotlib.lines.Line2D at 0x28fe58760>,
 <matplotlib.lines.Line2D at 0x28fe58850>,
 <matplotlib.lines.Line2D at 0x28fe58940>,
 <matplotlib.lines.Line2D at 0x28fe58a30>,
 <matplotlib.lines.Line2D at 0x28fe58b20>,
 <matplotlib.lines.Line2D at 0x28fe58c10>,
 <matplotlib.lines.Line2D at 0x28fe58d00>]
```

[19]:
```
Text(0.5, 1.0, 'Simulated PRILX Prices')
```

[19]:
```
Text(0.5, 0, 'Days')
```

[19]:
```
Text(0, 0.5, 'Price ($)')
```



4.3 RWMGX

[20]:
```
#plot 20 price paths
plt.figure(figsize=(15,6))
plt.plot(pd.DataFrame(price_paths_active_3).iloc[:,0:20])
plt.title("Simulated RWMGX Prices")
plt.xlabel("Days")
plt.ylabel("Price ($)")
```

[20]:
```
<Figure size 1500x600 with 0 Axes>
```

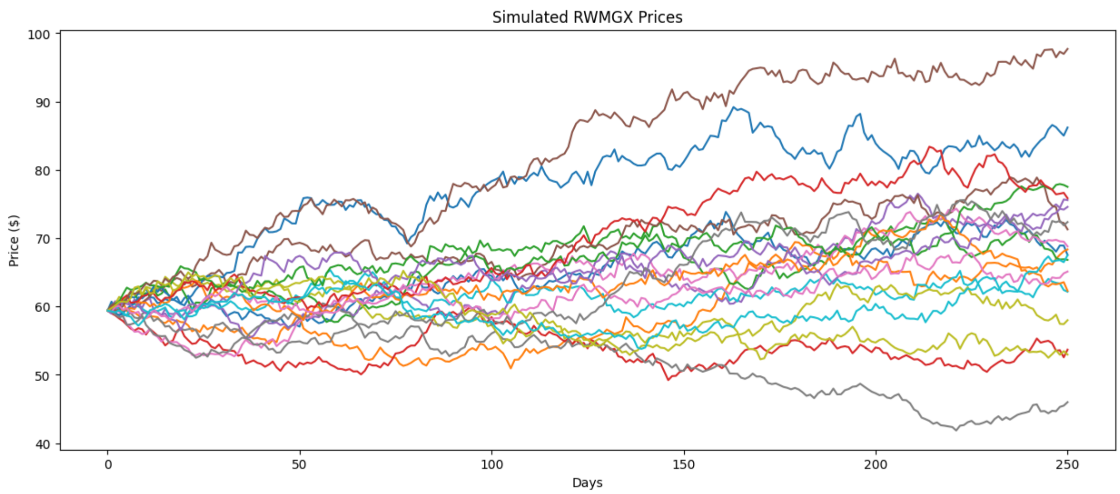
[20]:
```
[<matplotlib.lines.Line2D at 0x2a7634ac0>,
 <matplotlib.lines.Line2D at 0x2a7634b20>,
 <matplotlib.lines.Line2D at 0x2a7634b50>,
 <matplotlib.lines.Line2D at 0x2a7634c40>,
 <matplotlib.lines.Line2D at 0x2a7634d30>,
 <matplotlib.lines.Line2D at 0x2a7634e20>,
 <matplotlib.lines.Line2D at 0x2a7634f10>,
 <matplotlib.lines.Line2D at 0x2a7635000>,
 <matplotlib.lines.Line2D at 0x2a76350f0>,
 <matplotlib.lines.Line2D at 0x2a76351e0>,
 <matplotlib.lines.Line2D at 0x2a76352d0>,
 <matplotlib.lines.Line2D at 0x2a7634af0>,
 <matplotlib.lines.Line2D at 0x2a76353c0>,
 <matplotlib.lines.Line2D at 0x2a7635570>,
 <matplotlib.lines.Line2D at 0x2a7635660>,
 <matplotlib.lines.Line2D at 0x2a7605c60>,
 <matplotlib.lines.Line2D at 0x2a7606b60>,
 <matplotlib.lines.Line2D at 0x2a7606a10>,
 <matplotlib.lines.Line2D at 0x2a7635900>,
```

```
 <matplotlib.lines.Line2D at 0x2a76359f0>]
```

[20]: Text(0.5, 1.0, 'Simulated RWMGX Prices')

[20]: Text(0.5, 0, 'Days')

[20]: Text(0, 0.5, 'Price ($)')



4.4 WFSPX

[21]:
```
#plot 20 price paths
plt.figure(figsize=(15,6))
plt.plot(pd.DataFrame(price_paths_passive).iloc[:,0:20])
plt.title("Simulated WFSPX Prices")
plt.xlabel("Days")
plt.ylabel("Price ($)")
```

[21]: <Figure size 1500x600 with 0 Axes>

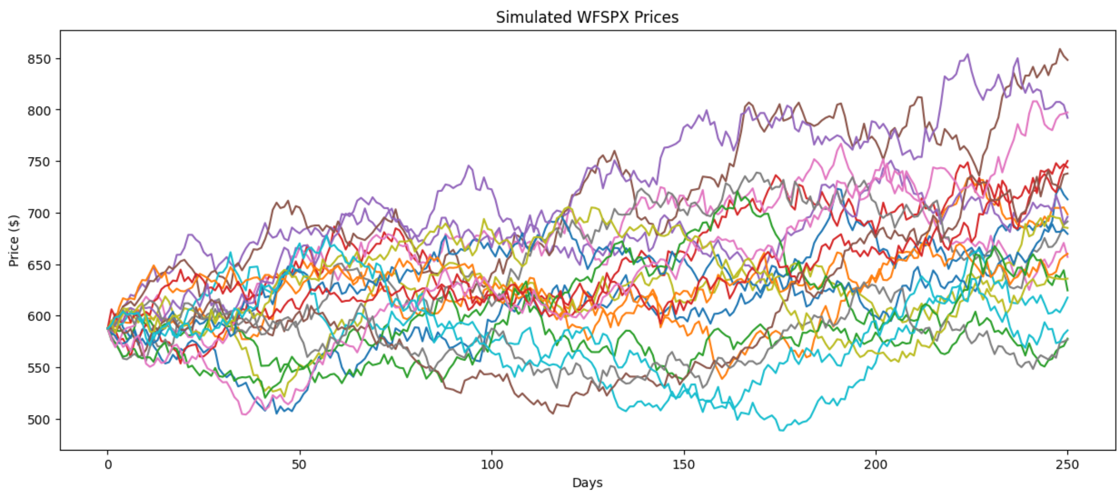
[21]:
```
[<matplotlib.lines.Line2D at 0x2a76c49d0>,
 <matplotlib.lines.Line2D at 0x2a76c5f60>,
 <matplotlib.lines.Line2D at 0x2a76c5f90>,
 <matplotlib.lines.Line2D at 0x2a76c6080>,
 <matplotlib.lines.Line2D at 0x2a76c6170>,
 <matplotlib.lines.Line2D at 0x2a76c6260>,
 <matplotlib.lines.Line2D at 0x2a76c6350>,
 <matplotlib.lines.Line2D at 0x2a76c6440>,
 <matplotlib.lines.Line2D at 0x2a76c6530>,
 <matplotlib.lines.Line2D at 0x2a76c6620>,
 <matplotlib.lines.Line2D at 0x2a76c6710>,
 <matplotlib.lines.Line2D at 0x2a76c5f30>,
```

```
 <matplotlib.lines.Line2D at 0x2a76c6800>,
 <matplotlib.lines.Line2D at 0x2a76c69b0>,
 <matplotlib.lines.Line2D at 0x2a76c6aa0>,
 <matplotlib.lines.Line2D at 0x2a76c6b90>,
 <matplotlib.lines.Line2D at 0x2a76c6c80>,
 <matplotlib.lines.Line2D at 0x2a76c6d70>,
 <matplotlib.lines.Line2D at 0x2a76c6e60>,
 <matplotlib.lines.Line2D at 0x2a76c6f50>]
```

[21]:
```
Text(0.5, 1.0, 'Simulated WFSPX Prices')
```

[21]:
```
Text(0.5, 0, 'Days')
```

[21]:
```
Text(0, 0.5, 'Price ($)')
```



4.4.1 Dataframes to CSV

[22]:
```python
start_time = time.time()

df1.to_csv('PCCOX_returns.csv', index = False, header = True)
df2.to_csv('PRILX_returns.csv', index = False, header = True)
df3.to_csv('RWMGX_returns.csv', index = False, header = True)
df_passive.to_csv('WFSPX_returns.csv', index = False, header = True)

# Get execution time
end_time = time.time()
execution_time = end_time - start_time
print(f"Execution Time: {execution_time} seconds")

# Get memory usage
```

```python
memory_info = psutil.Process().memory_info()
print(f"Memory Usage: {memory_info.rss / 1024 / 1024} MB")
```

```
Execution Time: 5.00885796546936 seconds
Memory Usage: 488.421875 MB
```