Unlocking Insights: A Holistic Data Analysis Approach Using Advanced Statistical Techniques and Visualizations

In [ ]:
```
import pandas as pd  # For data manipulation and analysis
import numpy as np  # For numerical computations
import matplotlib.pyplot as plt  # For creating static, animated, and interactive visualizations
import seaborn as sns  # For statistical data visualization based on Matplotlib
import scipy  # For scientific and technical computing (including optimization, integration, and statistics)
from sklearn.preprocessing import StandardScaler, LabelEncoder  # For preprocessing data (scaling, encoding)
from sklearn.model_selection import train_test_split  # For splitting data into training and testing sets
from sklearn.linear_model import LinearRegression  # For linear regression models
from sklearn.metrics import mean_squared_error, r2_score  # For model evaluation metrics
import statsmodels.api as sm  # For statistical modeling and hypothesis testing
```

In [110…
```
ser = pd.Series(np.random.rand(34))
```

In [111…
```
type(ser)
```

Out[111… pandas.core.series.Series

In [112…
```
newdf = pd.DataFrame(np.random.rand(3343,20), index=np.arange(3343))
```

In [113…
```
newdf
```

Out[113…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	0.697938	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

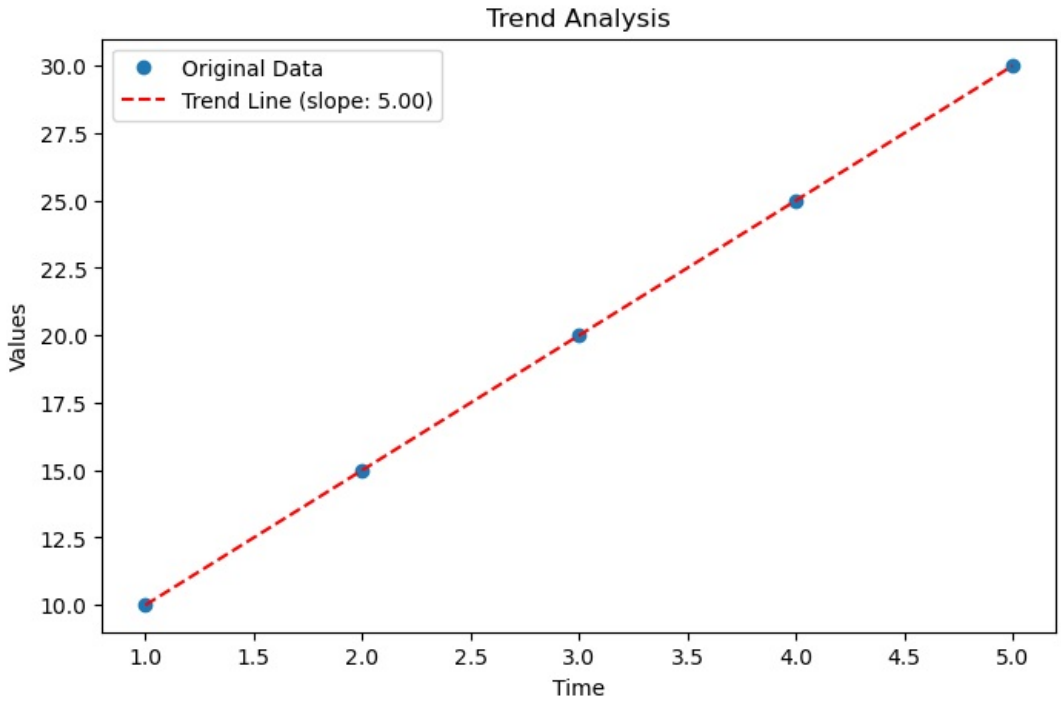
3343 rows × 20 columns

In [149…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Sample Data Generation (Replace with your actual data loading)
# For demonstration purposes, let's assume we have some sample data
data = {
    'category': ['A', 'B', 'C', 'D'],
    'values': [25, 37, 50, 23],
    'errors': [2, 3, 4, 5]
}

df = pd.DataFrame(data)
```

Sample Data Generation (Replace with actual data loading)

In [157…
```
# Sample data
time_series = np.array([1, 2, 3, 4, 5])
values = np.array([10, 15, 20, 25, 30])

# Trend Analysis using Polyfit (linear trend)
trend = np.polyfit(time_series, values, 1)
trend_line = np.polyval(trend, time_series)

# Plotting the original data and trend line
plt.figure(figsize=(8, 5))
plt.plot(time_series, values, 'o', label='Original Data')
```

```
plt.plot(time_series, trend_line, 'r--', label=f'Trend Line (slope: {trend[0]:.2f})')
plt.title('Trend Analysis')
plt.xlabel('Time')
plt.ylabel('Values')
plt.legend()
plt.show()
```

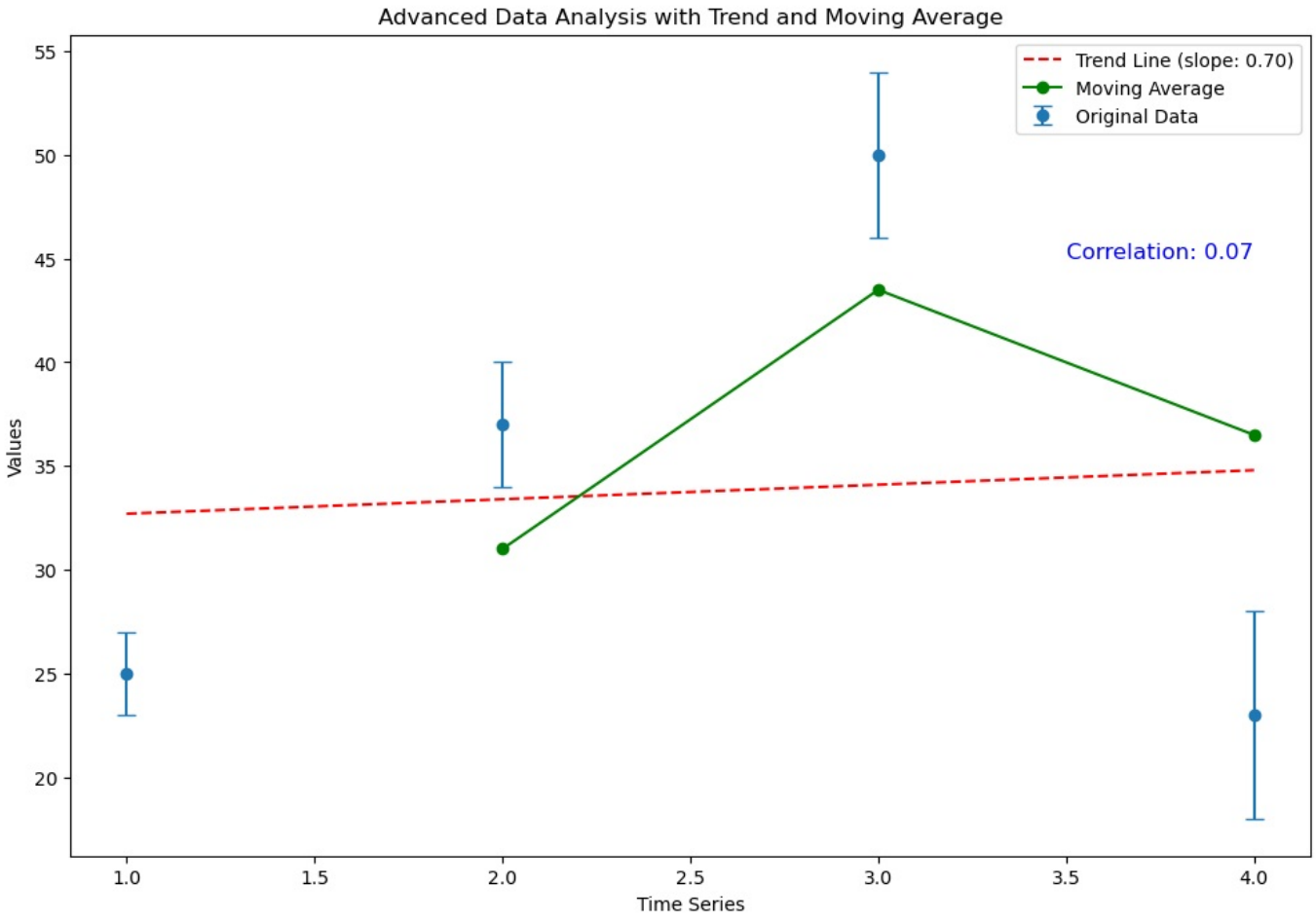


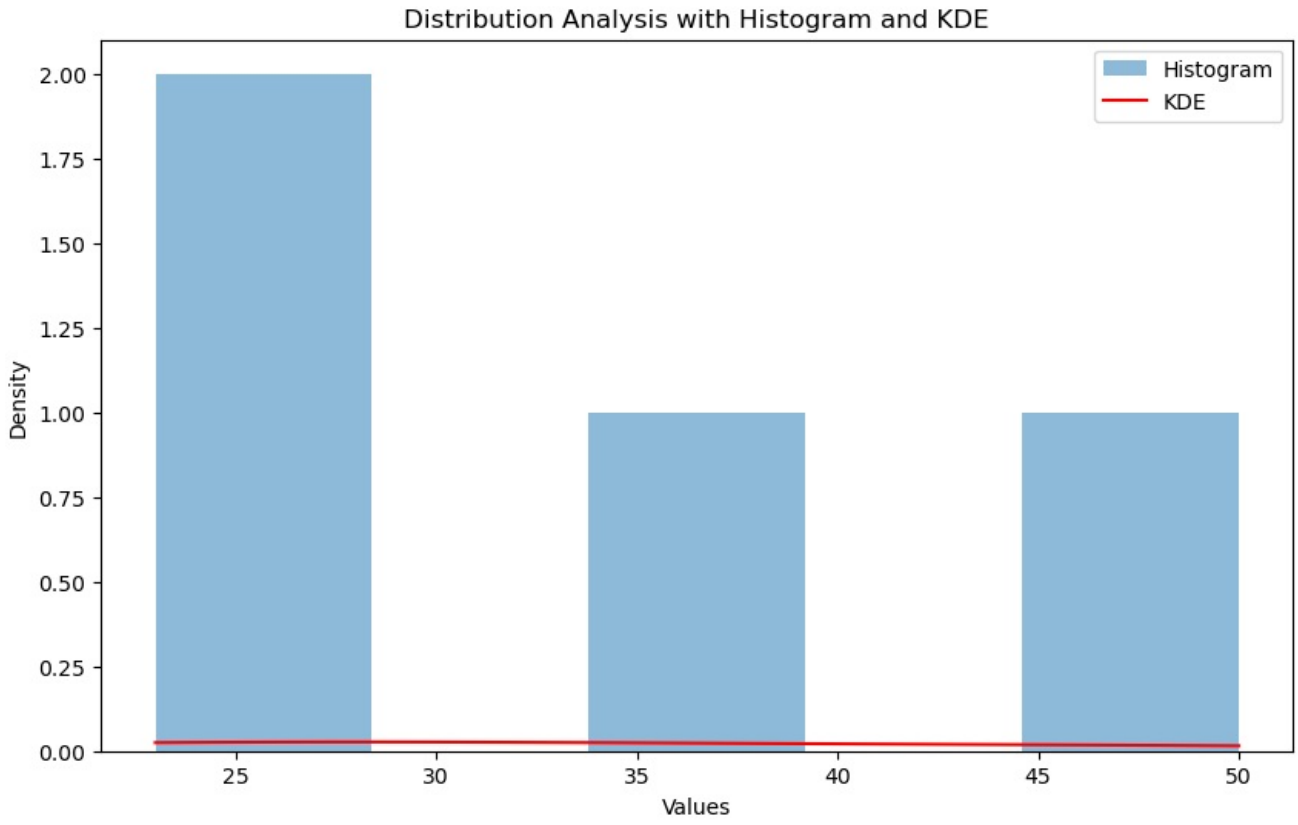
In [156…

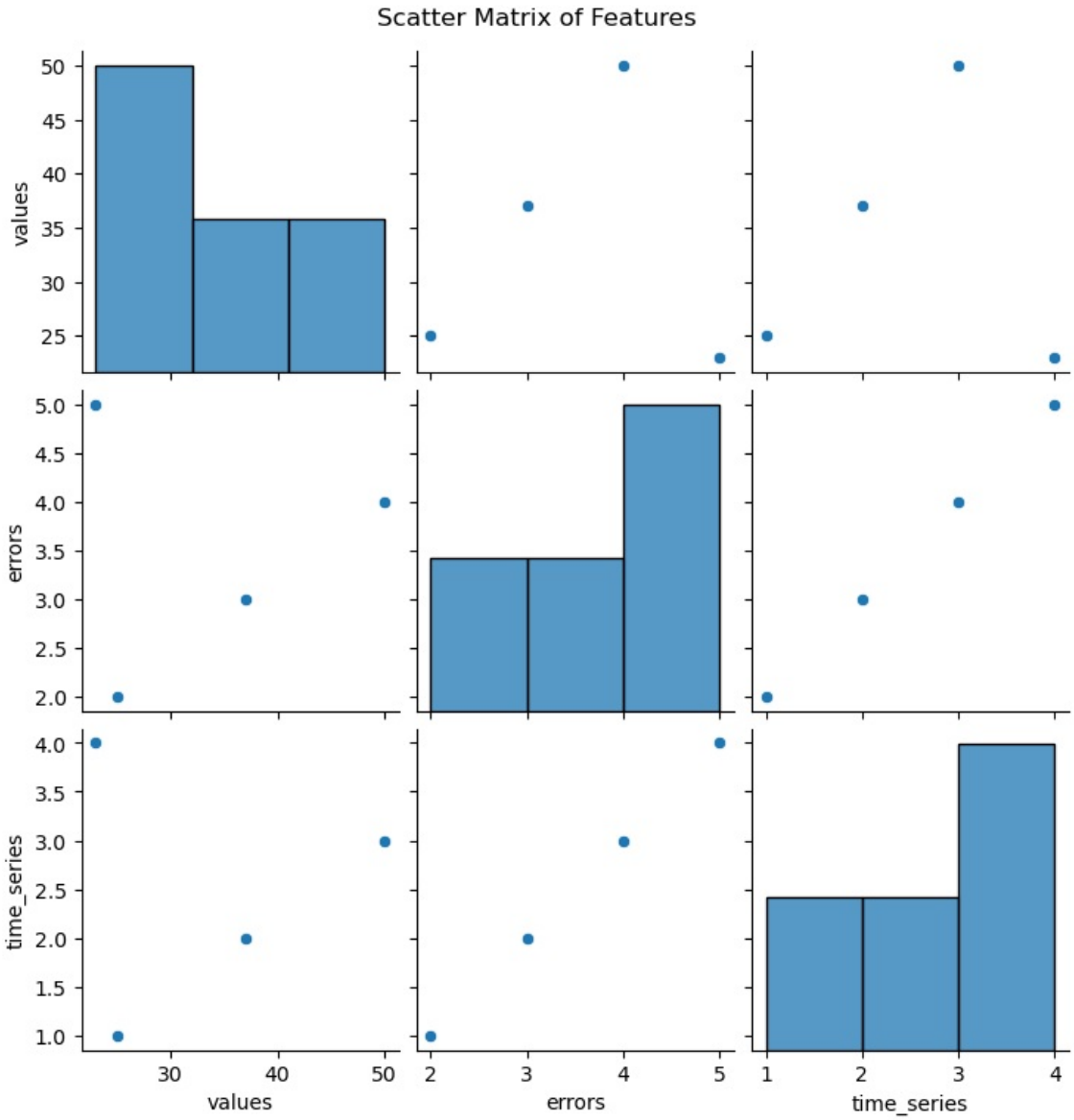
```
# This time, let's assume we have time-series data
data = {
    'category': ['A', 'B', 'C', 'D'],
    'values': [25, 37, 50, 23],
    'errors': [2, 3, 4, 5],
    'time_series': [1, 2, 3, 4]
}

df = pd.DataFrame(data)

# Convert data to NumPy arrays for detailed analysis
values = np.array(df['values'])
errors = np.array(df['errors'])
time_series = np.array(df['time_series'])

# 1. Trend Analysis using Polyfit (Polynomial Fitting)
trend = np.polyfit(time_series, values, 1)  # Linear trend
trend_line = np.polyval(trend, time_series)

# 2. Correlation Calculation using NumPy
correlation = np.corrcoef(time_series, values)[0, 1]
print(f"Correlation between Time and Values: {correlation:.2f}")

# 3. Moving Average Calculation
window_size = 2
moving_avg = np.convolve(values, np.ones(window_size)/window_size, mode='valid')

# Visualization using Matplotlib

plt.figure(figsize=(12, 8))

# Original Data with Error Bars
plt.errorbar(df['time_series'], values, yerr=errors, fmt='o', label='Original Data', capsize=5)

# Trend Line
plt.plot(time_series, trend_line, label=f'Trend Line (slope: {trend[0]:.2f})', color='r', linestyle='--')

# Moving Average
plt.plot(time_series[window_size-1:], moving_avg, label='Moving Average', color='g', marker='o')

# Annotations for Correlation
plt.text(3.5, np.max(values) - 5, f'Correlation: {correlation:.2f}', fontsize=12, color='b')

# Adding titles and labels
plt.title('Advanced Data Analysis with Trend and Moving Average')
plt.xlabel('Time Series')
plt.ylabel('Values')
plt.legend()
```

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

# 4. Histogram and KDE Plot for Distribution Analysis
plt.figure(figsize=(10, 6))

# Histogram
plt.hist(values, bins=5, alpha=0.5, label='Histogram')

# Kernel Density Estimation (KDE) using Matplotlib
from scipy.stats import gaussian_kde
kde = gaussian_kde(values)
kde_x = np.linspace(min(values), max(values), 100)
kde_y = kde(kde_x)

plt.plot(kde_x, kde_y, color='r', label='KDE')

# Adding titles and labels
plt.title('Distribution Analysis with Histogram and KDE')
plt.xlabel('Values')
plt.ylabel('Density')
plt.legend()

# Show plot
plt.show()

# 5. Scatter Matrix for Pairwise Relationships
import seaborn as sns

# Assuming the dataframe has multiple numerical columns
sns.pairplot(df)
plt.suptitle('Scatter Matrix of Features', y=1.02)

plt.show()
```

```
Correlation between Time and Values: 0.07
```




Distribution Analysis with Histogram and KDE
Histogram
KDE
Density
Values
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
25
30
35
40
45
50



In [148…
```
# Print calculated statistics
print(f"Mean: {mean}")
print(f"Standard Deviation: {std_dev}")
print(f"Variance: {variance}")
print(f"Median: {median}")
```

```
Mean: 33.75
Standard Deviation: 10.80219885023415
Variance: 116.6875
Median: 31.0
```

In [151…
```
# Convert data to NumPy arrays for detailed analysis
values = np.array(df['values'])
errors = np.array(df['errors'])

# Example of advanced analysis using NumPy
mean = np.mean(values)
std_dev = np.std(values)
variance = np.var(values)
median = np.median(values)
```

In [152…
```
print(newdf)
```

```
            0         1         2         3         4         5         6   \
0     satedner  0.561076  0.001192  0.450341  0.795850  0.930623  0.767605
1        Rahul  0.254787  0.427195  0.969418  0.213716  0.912283  0.974008
2     0.854784  0.793532  0.031116  0.471809  0.772358  0.447498  0.452217
3     0.562486  0.790980  0.744740  0.238140  0.171643  0.136453  0.936933
4     0.702867  0.270176  0.268086  0.076133  0.174052  0.712256  0.812254
...        ...       ...       ...       ...       ...       ...       ...
3338  0.242589  0.189695  0.527391  0.300127  0.902239  0.026674  0.125397
3339  0.887061  0.909481  0.001816  0.454096  0.124479  0.180765  0.284239
3340  0.531965  0.708004  0.179359  0.905226  0.234289  0.676948  0.990071
3341  0.060029  0.033379  0.814435  0.271410  0.507328  0.512107  0.554627
3342  0.930778  0.048853  0.275265  0.415583  0.517176  0.010622  0.092440

            7         8         9         10        11        12        13  \
0     0.894239  0.597057  0.125559  0.751090  0.278110  0.459299  0.171059
1     0.314478  0.691099  0.603735  0.979347  0.785053  0.310106  0.672351
2     0.644761  0.218107  0.125786  0.519473  0.229090  0.934016  0.107259
3     0.931844  0.380153  0.264501  0.673073  0.232032  0.122061  0.273927
4     0.648388  0.217212  0.763016  0.771966  0.055136  0.151104  0.818651
...        ...       ...       ...       ...       ...       ...       ...
3338  0.918755  0.432655  0.011054  0.326422  0.905210  0.211183  0.424995
3339  0.307691  0.909599  0.909509  0.890847  0.829765  0.066687  0.026478
3340  0.162226  0.992508  0.565066  0.343011  0.244501  0.411046  0.455853
3341  0.549610  0.743996  0.252561  0.118287  0.719574  0.237188  0.160270
3342  0.338265  0.599631  0.229081  0.001652  0.961243  0.287422  0.668404

            14        15        16        17        18        19
0     0.037895  0.065044  0.596998  0.741632  0.391194  0.382090
1     0.709704  0.082982  0.125456  0.143640  0.509255  0.451089
2     0.003470  0.085419  0.023010  0.987459  0.096069  0.430543
3     0.399441  0.192537  0.233651  0.729097  0.557619  0.680914
4     0.847048  0.466855  0.159128  0.219528  0.235784  0.922468
...        ...       ...       ...       ...       ...       ...
3338  0.089949  0.718238  0.329754  0.482533  0.567450  0.358958
3339  0.136172  0.921743  0.742154  0.520271  0.610764  0.880770
3340  0.369144  0.977681  0.170403  0.202506  0.942130  0.444679
3341  0.124291  0.142488  0.355018  0.010618  0.548974  0.066305
3342  0.187772  0.290403  0.631293  0.002956  0.535297  0.905234

[3343 rows x 20 columns]
```

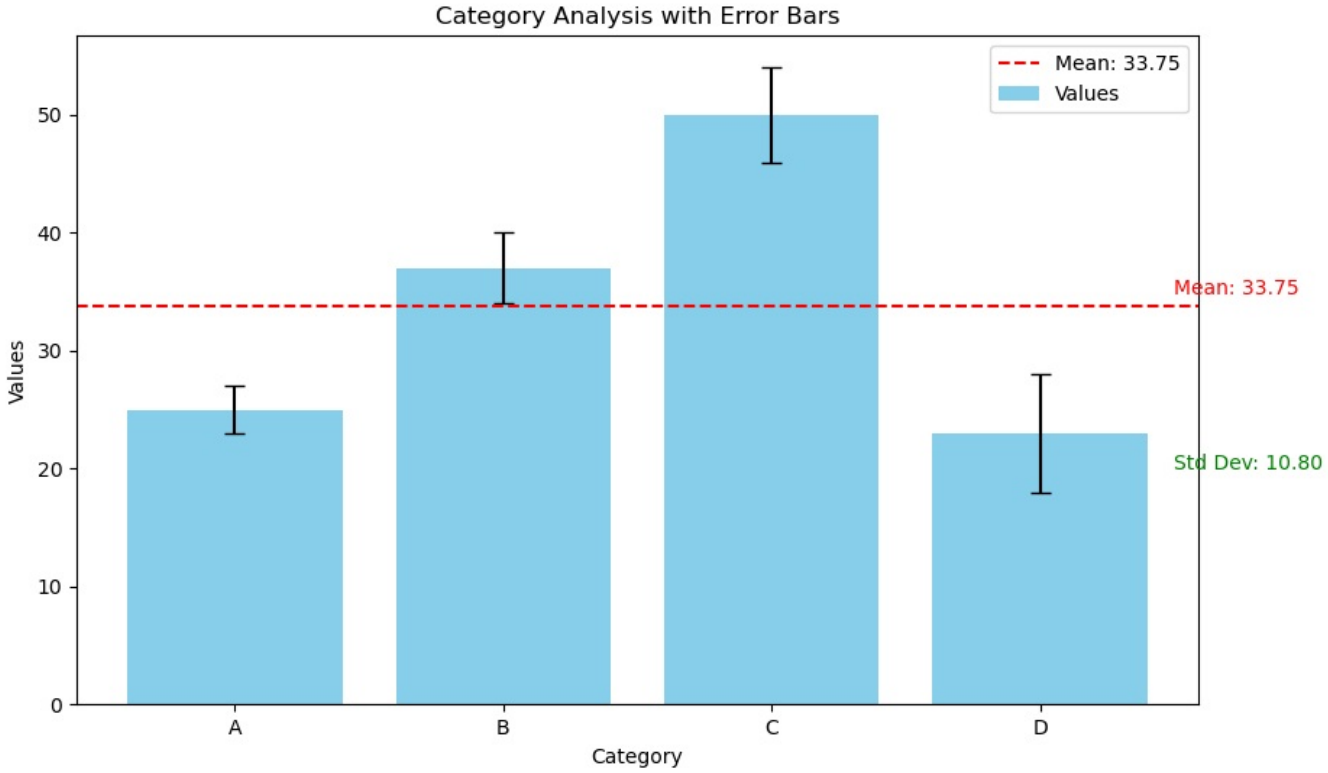
Print calculated statistics

In [154…

```
print(f"Mean: {mean}")
print(f"Standard Deviation: {std_dev}")
print(f"Variance: {variance}")
print(f"Median: {median}")

# Visualization using Matplotlib
plt.figure(figsize=(10, 6))

# Bar plot with error bars
plt.bar(df['category'], values, yerr=errors, capsize=5, color='skyblue', label='Values')

# Add mean line
plt.axhline(y=mean, color='r', linestyle='--', label=f'Mean: {mean:.2f}')

# Add text annotations for statistics
plt.text(3.5, mean + 1, f'Mean: {mean:.2f}', color='r')
plt.text(3.5, mean - std_dev - 3, f'Std Dev: {std_dev:.2f}', color='g')
plt.text(3.5, mean - variance - 6, f'Variance: {variance:.2f}', color='b')

# Adding titles and labels
plt.title('Category Analysis with Error Bars')
plt.xlabel('Category')
plt.ylabel('Values')
plt.legend()

# Show plot
plt.show()

# Advanced NumPy operations: Normalizing the data
normalized_values = (values - mean) / std_dev
print("Normalized Values:", normalized_values)

# Visualization of normalized data
plt.figure(figsize=(10, 6))
plt.bar(df['category'], normalized_values, color='orange', label='Normalized Values')

# Add titles and labels
plt.title('Normalized Values by Category')
plt.xlabel('Category')
plt.ylabel('Normalized Value')
plt.legend()
```

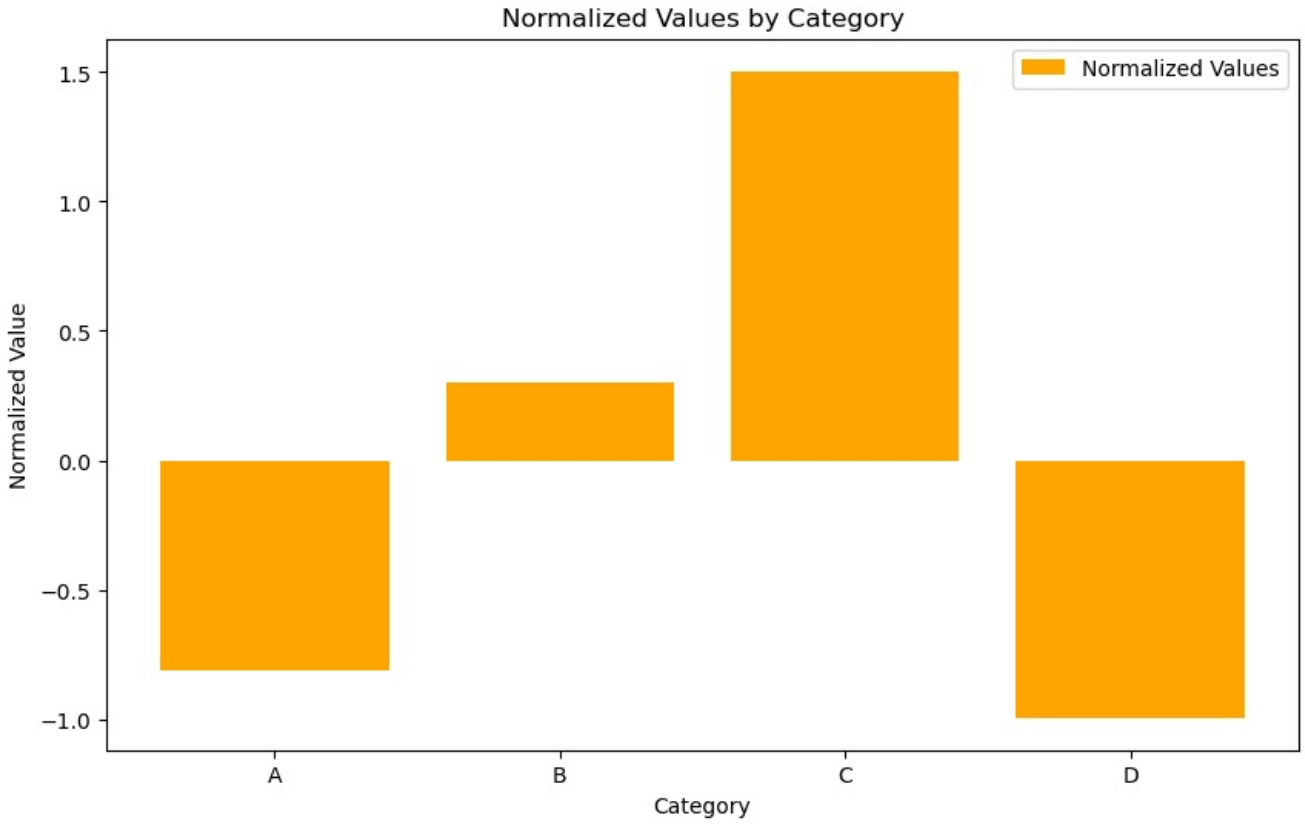
```
# Show plot
plt.show()
```

```
Mean: 33.75
Standard Deviation: 10.80219885023415
Variance: 116.6875
Median: 31.0
```



Variance: 116.69

```
Normalized Values: [-0.81002027  0.30086467  1.50432335 -0.99516776]
```



In [114…
```
newdf.head()
```

Out[114…

	0	1	2	3	4	5	6	7	8	9	10	11	1
0	0.697938	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45929
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.31010
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93401
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12206
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15110

In [115…
```
type(newdf)
```

Out[115… `pandas.core.frame.DataFrame`

In [116…
```
newdf.describe()
```

Out[116…

	0	1	2	3	4	5	6	7	8	
count	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	334
mean	0.503739	0.491004	0.491591	0.501394	0.493043	0.505059	0.505016	0.503800	0.505401	
std	0.289272	0.288965	0.287712	0.286441	0.289059	0.291096	0.290702	0.291508	0.283214	
min	0.000187	0.000288	0.000137	0.000232	0.000127	0.000114	0.000035	0.000412	0.000371	
25%	0.258086	0.232902	0.241878	0.255746	0.245966	0.249273	0.253529	0.252640	0.265151	
50%	0.509689	0.490917	0.487950	0.502484	0.485434	0.510141	0.511370	0.509279	0.505908	
75%	0.754876	0.742270	0.741858	0.748114	0.742720	0.754881	0.763551	0.760554	0.745003	
max	0.999349	0.999383	0.999779	0.999549	0.999999	0.999499	0.999996	0.999889	0.999710	

In [117…
```
newdf.dtypes
```

Out[117…
```
0     float64
1     float64
2     float64
3     float64
4     float64
5     float64
6     float64
7     float64
8     float64
9     float64
10    float64
11    float64
12    float64
13    float64
14    float64
15    float64
16    float64
17    float64
18    float64
19    float64
dtype: object
```

In [120…
```
# string put and change
```

In [121…
```
newdf.head()
```

Out[121…

	0	1	2	3	4	5	6	7	8	9	10	11	1
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45929
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3101(
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.9340
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.1220(
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.1511(

In [122…
```
newdf.index
```

Out[122…
```
Index([   0,    1,    2,    3,    4,    5,    6,    7,    8,    9,
       ...
       3333, 3334, 3335, 3336, 3337, 3338, 3339, 3340, 3341, 3342],
      dtype='int32', length=3343)
```

In [123…
```
newdf.columns
```

Out[123…
```
RangeIndex(start=0, stop=20, step=1)
```

In [124…
```
newdf.to_numpy()
```

Out[124…
```
array([['satedner', 0.5610761589842846, 0.001192114587430293, ...,
        0.7416324294616016, 0.3911937244361793, 0.3820896402352686],
       [0.046055937342585174, 0.254787362097771, 0.42719452008559466,
        ..., 0.14364003953672966, 0.5092546822021528, 0.4510891827054363],
       [0.8547835023621424, 0.7935318872325342, 0.03111625504524096, ...,
        0.987459426960006, 0.09606897531240688, 0.43054322731525496],
       ...,
       [0.5319647386673982, 0.7080038836479031, 0.17935934394912378, ...,
        0.2025064014616803, 0.9421295704371534, 0.4446792476788721],
       [0.060028950655676594, 0.03337944486345812, 0.814435289632806,
        ..., 0.010618131787275442, 0.5489742252985179,
        0.06630462134075654],
       [0.9307781597401553, 0.04885281230366967, 0.2752652403856555, ...,
        0.002956251639411489, 0.5352968436617144, 0.9052336521663139]],
      dtype=object)
```

In [125…
```
newdf.to_numpy()
```

Out[125…
```
array([['satedner', 0.5610761589842846, 0.001192114587430293, ...,
        0.7416324294616016, 0.3911937244361793, 0.3820896402352686],
       [0.046055937342585174, 0.254787362097771, 0.42719452008559466,
        ..., 0.14364003953672966, 0.5092546822021528, 0.4510891827054363],
       [0.8547835023621424, 0.7935318872325342, 0.03111625504524096, ...,
        0.987459426960006, 0.09606897531240688, 0.43054322731525496],
       ...,
       [0.5319647386673982, 0.7080038836479031, 0.17935934394912378, ...,
        0.2025064014616803, 0.9421295704371534, 0.4446792476788721],
       [0.060028950655676594, 0.03337944486345812, 0.814435289632806,
        ..., 0.010618131787275442, 0.5489742252985179,
        0.06630462134075654],
       [0.9307781597401553, 0.04885281230366967, 0.2752652403856555, ...,
        0.002956251639411489, 0.5352968436617144, 0.9052336521663139]],
      dtype=object)
```

In [126…
```
newdf.T
```

Out[126…

	0	1	2	3	4	5	6	7	8	9	...	3333	3334	
0	satedner	0.046056	0.854784	0.562486	0.702867	0.95931	0.024801	0.01371	0.793867	0.763245	...	0.704641	0.790551	0.
1	0.561076	0.254787	0.793532	0.79098	0.270176	0.940277	0.527591	0.241154	0.863165	0.120385	...	0.024275	0.678563	0.
2	0.001192	0.427195	0.031116	0.74474	0.268086	0.411395	0.615717	0.916726	0.084959	0.306412	...	0.155299	0.001609	0.
3	0.450341	0.969418	0.471809	0.23814	0.076133	0.207079	0.650445	0.498439	0.524331	0.414657	...	0.945955	0.546366	0.
4	0.79585	0.213716	0.772358	0.171643	0.174052	0.966039	0.54581	0.639105	0.250488	0.946433	...	0.024077	0.238793	0.
5	0.930623	0.912283	0.447498	0.136453	0.712256	0.672888	0.432509	0.16736	0.097094	0.225367	...	0.945789	0.88016	0
6	0.767605	0.974008	0.452217	0.936933	0.812254	0.175457	0.530555	0.812364	0.481824	0.303791	...	0.189091	0.665103	0.
7	0.894239	0.314478	0.644761	0.931844	0.648388	0.859932	0.942829	0.088588	0.859426	0.758608	...	0.428781	0.158825	0.
8	0.597057	0.691099	0.218107	0.380153	0.217212	0.902214	0.769244	0.720237	0.132585	0.048448	...	0.594817	0.221407	0.
9	0.125559	0.603735	0.125786	0.264501	0.763016	0.452349	0.999601	0.576733	0.412966	0.371679	...	0.415409	0.531016	0.
10	0.75109	0.979347	0.519473	0.673073	0.771966	0.565652	0.403283	0.669826	0.737933	0.70599	...	0.473757	0.329678	0.
11	0.27811	0.785053	0.22909	0.232032	0.055136	0.911236	0.617311	0.338826	0.633086	0.012504	...	0.822269	0.301242	0.
12	0.459299	0.310106	0.934016	0.122061	0.151104	0.498092	0.357427	0.603305	0.112751	0.385815	...	0.138073	0.008323	0.
13	0.171059	0.672351	0.107259	0.273927	0.818651	0.641221	0.162805	0.180483	0.963244	0.947437	...	0.779108	0.481304	0.
14	0.037895	0.709704	0.00347	0.399441	0.847048	0.281745	0.223323	0.999188	0.025439	0.420454	...	0.372698	0.708471	0.
15	0.065044	0.082982	0.085419	0.192537	0.466855	0.687869	0.025836	0.74007	0.53407	0.828131	...	0.324419	0.228337	0
16	0.596998	0.125456	0.02301	0.233651	0.159128	0.2913	0.774809	0.599471	0.631992	0.013455	...	0.500413	0.180724	0.
17	0.741632	0.14364	0.987459	0.729097	0.219528	0.233315	0.391999	0.007596	0.069191	0.946884	...	0.754954	0.28647	0.
18	0.391194	0.509255	0.096069	0.557619	0.235784	0.311641	0.607267	0.936231	0.640682	0.851307	...	0.092456	0.156106	0.
19	0.38209	0.451089	0.430543	0.680914	0.922468	0.441176	0.553356	0.879719	0.612646	0.933405	...	0.17309	0.962706	0.

20 rows × 3343 columns

In [127…
```
newdf
```

Out[127…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [128…
```
newdf.head()
```

Out[128…

	0	1	2	3	4	5	6	7	8	9	10	11	1
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45929
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3101(
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.9340
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.1220(
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.1511(

In [129…
```
newdf.sort_index()
```

Out[129…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [130…
```
newdf
```

Out[130…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [131…
```
newdf
```

Out[131…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [132…
```
# Numbering by data in DataFrame - SK
```

In [133…
```
newdf[0]
```

Out[133…
```
0       satedner
1       0.046056
2       0.854784
3       0.562486
4       0.702867
          ...
3338    0.242589
3339    0.887061
3340    0.531965
3341    0.060029
3342    0.930778
Name: 0, Length: 3343, dtype: object
```

In [134…
```
newdf[1]
```

Out[134…
```
0       0.561076
1       0.254787
2       0.793532
3       0.790980
4       0.270176
          ...
3338    0.189695
3339    0.909481
3340    0.708004
3341    0.033379
3342    0.048853
Name: 1, Length: 3343, dtype: float64
```

In [135…
```
newdf[3]
```

Out[135…
```
0       0.450341
1       0.969418
2       0.471809
3       0.238140
4       0.076133
          ...
3338    0.300127
3339    0.454096
3340    0.905226
3341    0.271410
3342    0.415583
Name: 3, Length: 3343, dtype: float64
```

In [136…
```
newdf[4]
```

Out[136…
```
0       0.795850
1       0.213716
2       0.772358
3       0.171643
4       0.174052
          ...
3338    0.902239
3339    0.124479
3340    0.234289
3341    0.507328
3342    0.517176
Name: 4, Length: 3343, dtype: float64
```

In [137…
```
newdf
```

Out[137…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [138…
```
newdf
```

Out[138…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	0.046056	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [139…
```
newdf.loc[1,0] = "Rahul"
```

In [140…
```
newdf
```

Out[140…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	Rahul	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [141…
```
type(newdf)
```

Out[141… pandas.core.frame.DataFrame

In [143…
```
newdf
```

Out[143…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	Rahul	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [144…
```
newdf
```

Out[144…

	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	Rahul	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

3343 rows × 20 columns

In [145…
```
ser = pd.Series(np.random.rand(34))
```

In [ ]:
```
newdf = pd.DataFrame(np.random.rand(3343,20), index=np.arange(3343))
```

In [146…
```
newdf
```

Out[146…

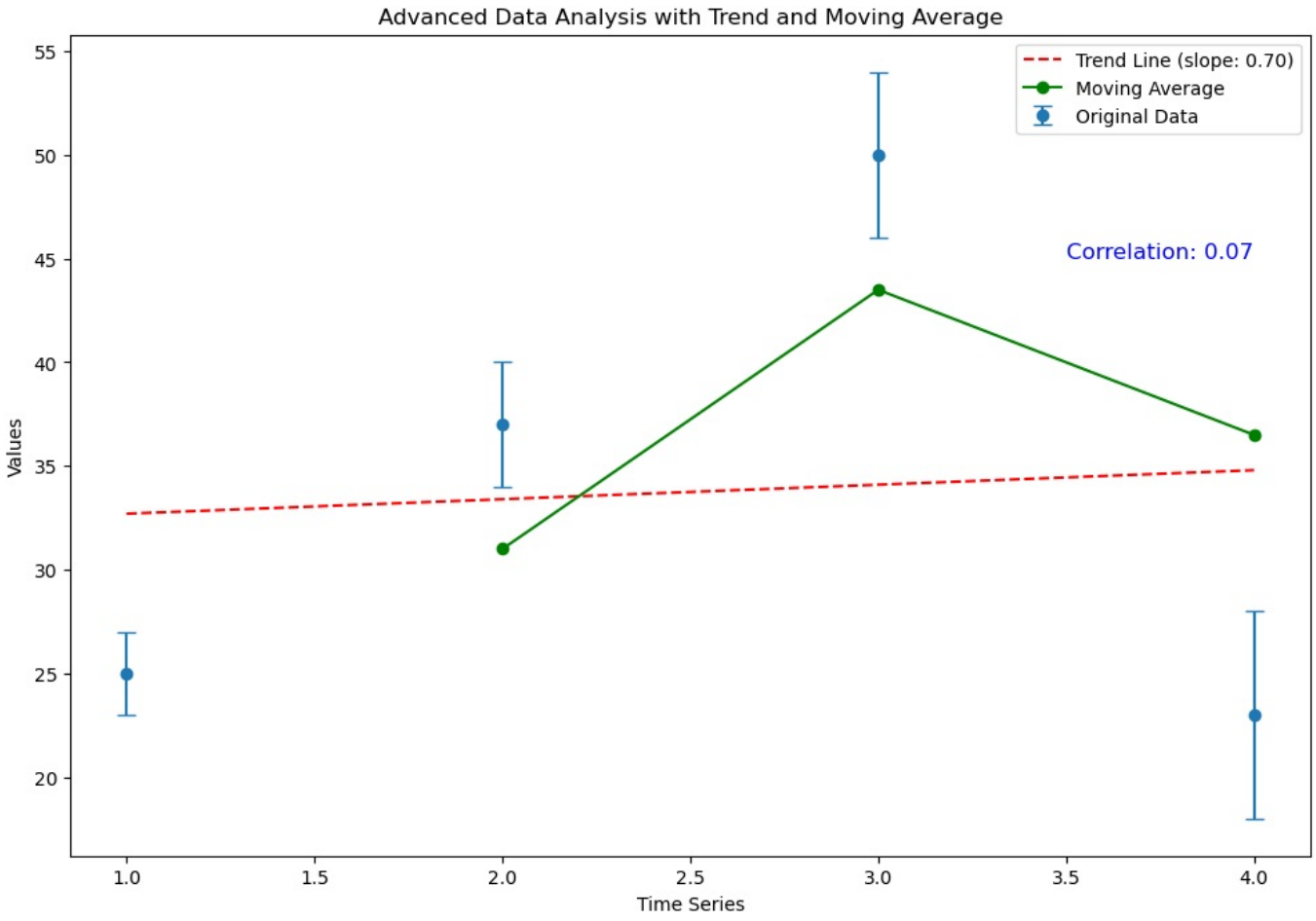
	0	1	2	3	4	5	6	7	8	9	10	11	
0	satedner	0.561076	0.001192	0.450341	0.795850	0.930623	0.767605	0.894239	0.597057	0.125559	0.751090	0.278110	0.45
1	Rahul	0.254787	0.427195	0.969418	0.213716	0.912283	0.974008	0.314478	0.691099	0.603735	0.979347	0.785053	0.3
2	0.854784	0.793532	0.031116	0.471809	0.772358	0.447498	0.452217	0.644761	0.218107	0.125786	0.519473	0.229090	0.93
3	0.562486	0.790980	0.744740	0.238140	0.171643	0.136453	0.936933	0.931844	0.380153	0.264501	0.673073	0.232032	0.12
4	0.702867	0.270176	0.268086	0.076133	0.174052	0.712256	0.812254	0.648388	0.217212	0.763016	0.771966	0.055136	0.15
...	...	...	...	...	...	...	...	...	...	...	...	...	...
3338	0.242589	0.189695	0.527391	0.300127	0.902239	0.026674	0.125397	0.918755	0.432655	0.011054	0.326422	0.905210	0.2
3339	0.887061	0.909481	0.001816	0.454096	0.124479	0.180765	0.284239	0.307691	0.909599	0.909509	0.890847	0.829765	0.06
3340	0.531965	0.708004	0.179359	0.905226	0.234289	0.676948	0.990071	0.162226	0.992508	0.565066	0.343011	0.244501	0.4
3341	0.060029	0.033379	0.814435	0.271410	0.507328	0.512107	0.554627	0.549610	0.743996	0.252561	0.118287	0.719574	0.23
3342	0.930778	0.048853	0.275265	0.415583	0.517176	0.010622	0.092440	0.338265	0.599631	0.229081	0.001652	0.961243	0.28

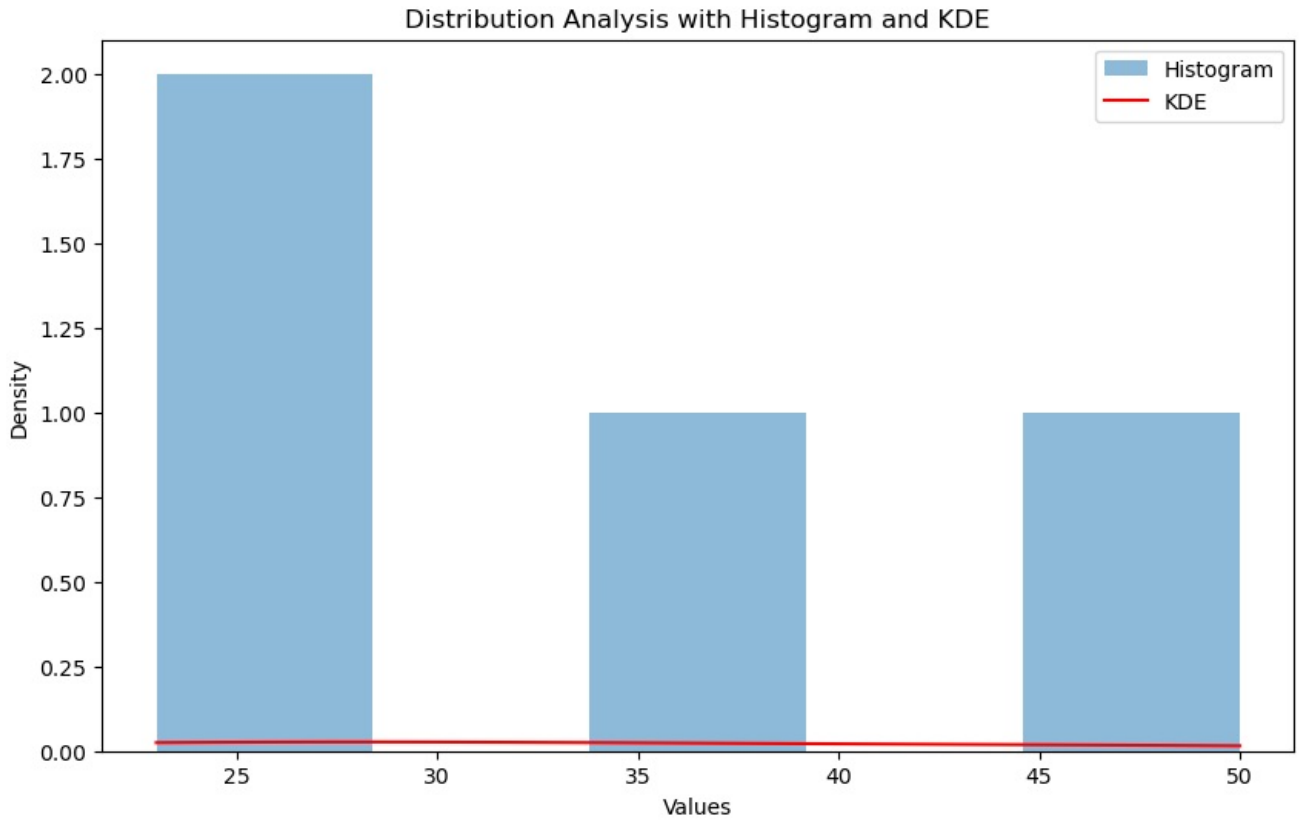
3343 rows × 20 columns

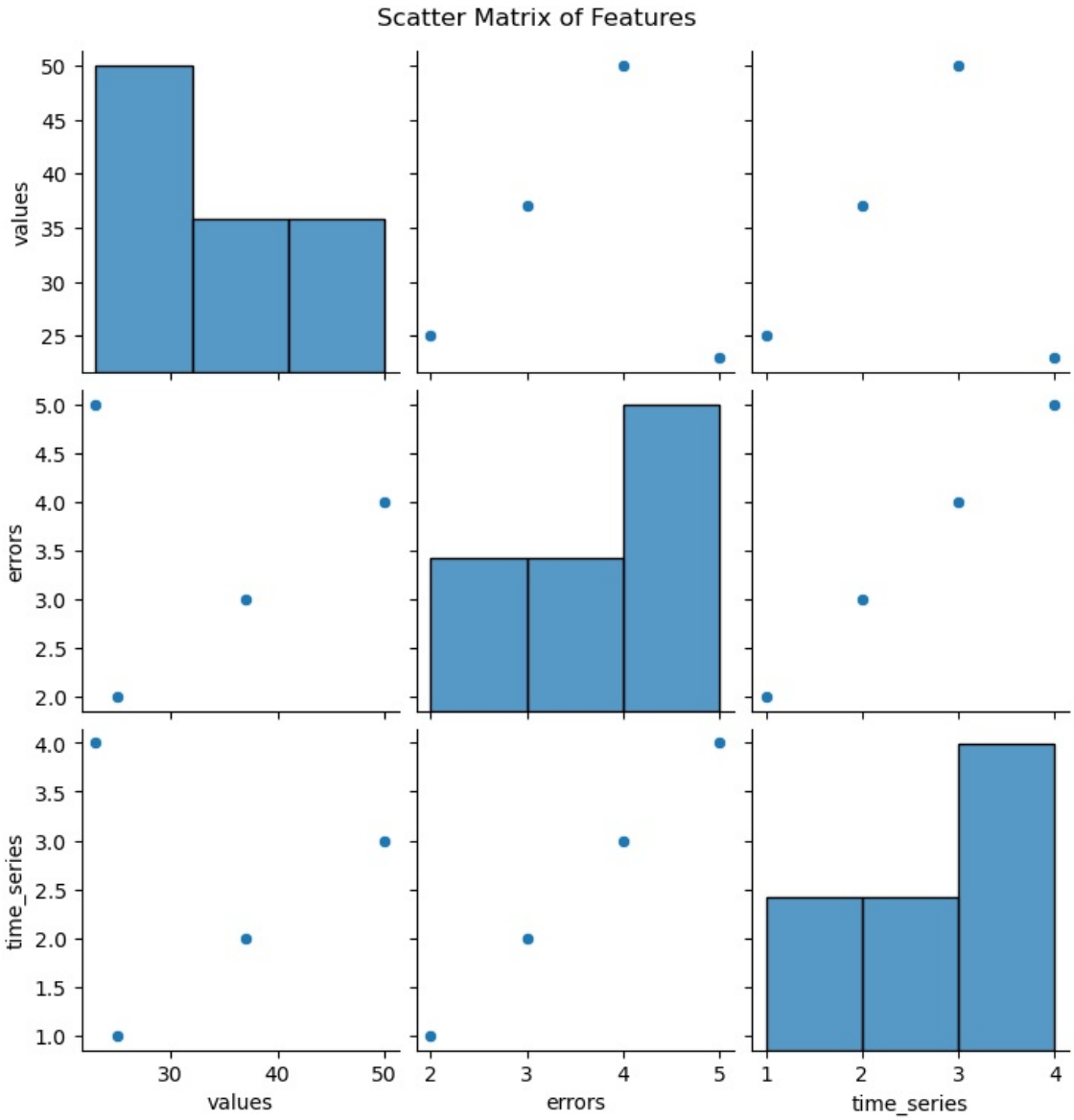
In [158…
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Sample Data Generation (Replace with your actual data loading)
# This time, let's assume we have time-series data
data = {
    'category': ['A', 'B', 'C', 'D'],
    'values': [25, 37, 50, 23],
    'errors': [2, 3, 4, 5],
    'time_series': [1, 2, 3, 4]
}

df = pd.DataFrame(data)

# Convert data to NumPy arrays for detailed analysis
values = np.array(df['values'])
errors = np.array(df['errors'])
time_series = np.array(df['time_series'])

# 1. Trend Analysis using Polyfit (Polynomial Fitting)
trend = np.polyfit(time_series, values, 1)  # Linear trend
trend_line = np.polyval(trend, time_series)

# 2. Correlation Calculation using NumPy
correlation = np.corrcoef(time_series, values)[0, 1]
print(f"Correlation between Time and Values: {correlation:.2f}")

# 3. Moving Average Calculation
window_size = 2
moving_avg = np.convolve(values, np.ones(window_size)/window_size, mode='valid')

# Visualization using Matplotlib

plt.figure(figsize=(12, 8))

# Original Data with Error Bars
plt.errorbar(df['time_series'], values, yerr=errors, fmt='o', label='Original Data', capsize=5)

# Trend Line
plt.plot(time_series, trend_line, label=f'Trend Line (slope: {trend[0]:.2f})', color='r', linestyle='--')

# Moving Average
plt.plot(time_series[window_size-1:], moving_avg, label='Moving Average', color='g', marker='o')

# Annotations for Correlation
plt.text(3.5, np.max(values) - 5, f'Correlation: {correlation:.2f}', fontsize=12, color='b')

# Adding titles and labels
plt.title('Advanced Data Analysis with Trend and Moving Average')
plt.xlabel('Time Series')
plt.ylabel('Values')
plt.legend()

# Show plot
```

```
plt.show()

# 4. Histogram and KDE Plot for Distribution Analysis
plt.figure(figsize=(10, 6))

# Histogram
plt.hist(values, bins=5, alpha=0.5, label='Histogram')

# Kernel Density Estimation (KDE) using Matplotlib
from scipy.stats import gaussian_kde
kde = gaussian_kde(values)
kde_x = np.linspace(min(values), max(values), 100)
kde_y = kde(kde_x)

plt.plot(kde_x, kde_y, color='r', label='KDE')

# Adding titles and labels
plt.title('Distribution Analysis with Histogram and KDE')
plt.xlabel('Values')
plt.ylabel('Density')
plt.legend()

# Show plot
plt.show()

# 5. Scatter Matrix for Pairwise Relationships
import seaborn as sns

# Assuming the dataframe has multiple numerical columns
sns.pairplot(df)
plt.suptitle('Scatter Matrix of Features', y=1.02)

plt.show()
```

```
Correlation between Time and Values: 0.07
```




Distribution Analysis with Histogram and KDE
Histogram
KDE
Density
Values
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
25
30
35
40
45
50



In [162…
```
newdf.describe()
```

Out[162…

	1	2	3	4	5	6	7	8	9	
count	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	3343.000000	334
mean	0.491004	0.491591	0.501394	0.493043	0.505059	0.505016	0.503800	0.505401	0.503050	
std	0.288965	0.287712	0.286441	0.289059	0.291096	0.290702	0.291508	0.283214	0.287236	
min	0.000288	0.000137	0.000232	0.000127	0.000114	0.000035	0.000412	0.000371	0.000129	
25%	0.232902	0.241878	0.255746	0.245966	0.249273	0.253529	0.252640	0.265151	0.252090	
50%	0.490917	0.487950	0.502484	0.485434	0.510141	0.511370	0.509279	0.505908	0.511891	
75%	0.742270	0.741858	0.748114	0.742720	0.754881	0.763551	0.760554	0.745003	0.749311	
max	0.999383	0.999779	0.999549	0.999999	0.999499	0.999996	0.999889	0.999710	0.999620	

In [164…
```
newdf.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 3343 entries, 0 to 3342
Data columns (total 20 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   0       3343 non-null   object 
 1   1       3343 non-null   float64
 2   2       3343 non-null   float64
 3   3       3343 non-null   float64
 4   4       3343 non-null   float64
 5   5       3343 non-null   float64
 6   6       3343 non-null   float64
 7   7       3343 non-null   float64
 8   8       3343 non-null   float64
 9   9       3343 non-null   float64
 10  10      3343 non-null   float64
 11  11      3343 non-null   float64
 12  12      3343 non-null   float64
 13  13      3343 non-null   float64
 14  14      3343 non-null   float64
 15  15      3343 non-null   float64
 16  16      3343 non-null   float64
 17  17      3343 non-null   float64
 18  18      3343 non-null   float64
 19  19      3343 non-null   float64
dtypes: float64(19), object(1)
memory usage: 632.4+ KB
```