1. Dataset Setup

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Data Analysis with Pandas using California Housing Dataset:

```
from sklearn.datasets import fetch_california_housing
import pandas as pd

data = fetch_california_housing(as_frame=True)
df = data.frame
```

Data Setup

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   MedInc       20640 non-null  float64
 1   HouseAge     20640 non-null  float64
 2   AveRooms     20640 non-null  float64
 3   AveBedrms    20640 non-null  float64
 4   Population   20640 non-null  float64
 5   AveOccup     20640 non-null  float64
 6   Latitude     20640 non-null  float64
 7   Longitude    20640 non-null  float64
 8   MedHouseVal  20640 non-null  float64
dtypes: float64(9)
memory usage: 1.4 MB
```

```
df.describe()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
count	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	3.870671	28.639486	5.429000	1.096675	1425.476744	3.070655	35.631861	-119.569704	2.068558
std	1.899822	12.585558	2.474173	0.473911	1132.462122	10.386050	2.135952	2.003532	1.153956
min	0.499900	1.000000	0.846154	0.333333	3.000000	0.692308	32.540000	-124.350000	0.149990
25%	2.563400	18.000000	4.440716	1.006079	787.000000	2.429741	33.930000	-121.800000	1.196000
50%	3.534800	29.000000	5.229129	1.048780	1166.000000	2.818116	34.260000	-118.490000	1.797000
75%	4.743250	37.000000	6.052381	1.099526	1725.000000	3.282261	37.710000	-118.010000	2.647250
max	15.000100	52.000000	141.909091	34.066667	35682.000000	1243.333333	41.950000	-114.310000	5.000010

Problem 1– Sorting

1. Create a DataFrame med
2. Create a DataFrame pop f irst 5 rows. income containing only the MedInc column. Display the first 5 rows. lat with columns Population and Latitude (in that order). Display the first five rows.
3. Create a DataFrame house age rooms with columns HouseAge and AveRooms. Display the first 5 rows,

```
med_income = df[['MedInc']]
print(med_income.head())

pop_lat = df[['Population', 'Latitude']]
print(pop_lat.head())

house_age_rooms = df[['HouseAge', 'AveRooms']]
print(house_age_rooms.head())
```

```
   MedInc
0  8.3252
1  8.3014
2  7.2574
3  5.6431
4  3.8462
   Population  Latitude
0       322.0     37.88
1      2401.0     37.86
2       496.0     37.85
3       558.0     37.85
4       565.0     37.85
   HouseAge  AveRooms
0      41.0  6.984127
1      21.0  6.238137
2      52.0  8.288136
3      52.0  5.817352
4      52.0  6.281853
```

Problem 2 – Subsetting

```
high_income = df[df['MedInc'] > 8.0]
print(high_income)

north_california = df[df['Latitude'] > 37]
print(north_california)

spacious_low_occupancy = df[(df['AveRooms'] > 6.0) & (df['AveOccup'] < 2.0)]
print(spacious_low_occupancy)
```

```
        MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
0       8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1       8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
131    11.6017      18.0  8.335052   1.082474       533.0  2.747423
37.84
134     8.2049      28.0  6.978947   0.968421       463.0  2.436842
37.83
135     8.4010      26.0  7.530806   1.056872       542.0  2.568720
37.83
...        ...       ...       ...        ...         ...       ...
...
20426  10.0472      11.0  9.890756   1.159664       415.0  3.487395
34.18
20427   8.6499       4.0  7.236059   1.032528      5495.0  2.553439
34.19
20428   8.7288       6.0  8.715842   1.102970      3385.0  3.351485
34.23
20436  12.5420      10.0  9.873315   1.102426      1179.0  3.177898
34.21
20503   8.2787      27.0  6.935065   1.103896       243.0  3.155844
34.33

       Longitude  MedHouseVal
0         -122.23      4.52600
1         -122.22      3.58500
131       -122.19      3.92600
134       -122.19      3.35200
135       -122.20      3.51200
...           ...          ...
20426     -118.69      5.00001
20427     -118.80      5.00001
20428     -118.83      4.25800
20436     -118.69      5.00001
20503     -118.75      3.30000

[690 rows x 9 columns]
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude \
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467
```

```
37.85
...        ...        ...        ...        ...          ...        ...
...
20635  1.5603       25.0  5.045455   1.133333        845.0  2.560606
39.48
20636  2.5568       18.0  6.114035   1.315789        356.0  3.122807
39.49
20637  1.7000       17.0  5.205543   1.120092       1007.0  2.325635
39.43
20638  1.8672       18.0  5.329513   1.171920        741.0  2.123209
39.43
20639  2.3886       16.0  5.254717   1.162264       1387.0  2.616981
39.37

        Longitude  MedHouseVal
0          -122.23        4.526
1          -122.22        3.585
2          -122.24        3.521
3          -122.25        3.413
4          -122.25        3.422
...            ...          ...
20635      -121.09        0.781
20636      -121.21        0.771
20637      -121.22        0.923
20638      -121.32        0.847
20639      -121.24        0.894

[7558 rows x 9 columns]
       MedInc  HouseAge    AveRooms  AveBedrms  Population  AveOccup
Latitude \
418    6.1593       41.0   6.215470   1.077348        356.0  1.966851
37.89
648    6.5095       25.0   6.631579   0.894737        340.0  1.988304
37.72
710    2.4196       26.0   8.518248   2.700730        253.0  1.846715
37.68
1024   3.1500       16.0  29.852941   5.323529        202.0  1.980392
38.52
1102   2.4028       17.0  31.777778   9.703704         47.0  1.740741
40.06
...        ...        ...        ...        ...          ...        ...
...
17155  7.4542       16.0   6.483333   1.066667        512.0  1.706667
37.42
17878  3.7614       14.0   9.363636   2.185687        813.0  1.572534
37.40
19362  3.3125        9.0  16.541284   3.116208        594.0  1.816514
38.70
20355  1.9811       16.0   6.104730   1.168919        587.0  1.983108
34.19
```

```
20423  5.4346      17.0    6.261168    1.505155       578.0  1.986254
34.08

       Longitude  MedHouseVal
418       -122.25      3.44000
648       -122.13      3.71200
710       -122.08      2.75000
1024      -120.00      1.40600
1102      -121.54      0.67500
...           ...          ...
17155     -122.23      5.00001
17878     -122.01      1.37500
19362     -123.49      2.95400
20355     -118.96      1.62500
20423     -119.00      4.28600

[106 rows x 9 columns]
```

Region column and filtering

```
def region_label(lat):
    if lat > 37:
        return 'North'
    elif lat > 35:
        return 'Central'
    else:
        return 'South'

df['Region'] = df['Latitude'].apply(region_label)
north_central_region = df[df['Region'].isin(['North', 'Central'])]
print(north_central_region.head())
```

```
   MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup
Latitude  \
0  8.3252      41.0  6.984127   1.023810       322.0  2.555556
37.88
1  8.3014      21.0  6.238137   0.971880      2401.0  2.109842
37.86
2  7.2574      52.0  8.288136   1.073446       496.0  2.802260
37.85
3  5.6431      52.0  5.817352   1.073059       558.0  2.547945
37.85
4  3.8462      52.0  6.281853   1.081081       565.0  2.181467
37.85

   Longitude  MedHouseVal Region
0    -122.23        4.526  North
1    -122.22        3.585  North
2    -122.24        3.521  North
```

```
3    -122.25         3.413   North
4    -122.25         3.422   North
```

Problem–3 ExploratoryDataAnalysis:

```
df['value_per_room'] = df['MedHouseVal'] / df['AveRooms']
high_vpr = df[df['value_per_room'] > 1]
high_vpr_sorted = high_vpr.sort_values(by='value_per_room',
ascending=False)
print(high_vpr_sorted[['MedHouseVal', 'AveRooms',
'value_per_room']].head())

df['income_per_person'] = df['MedInc'] / df['Population']
dense_areas = df[df['Population'] > 5000]
rich_dense_areas = dense_areas.sort_values(by='income_per_person',
ascending=False)
print(rich_dense_areas[['MedInc', 'Population',
'income_per_person']].head())
```

```
       MedHouseVal   AveRooms   value_per_room
15660      5.00001   1.824719         2.740153
15654      4.50000   1.902087         2.365823
4559       5.00001   2.148148         2.327591
15652      5.00001   2.237474         2.234667
15661      5.00001   2.297872         2.175930
       MedInc  Population  income_per_person
9004   9.1232      5452.0           0.001673
20427  8.6499      5495.0           0.001574
9027   7.7848      5175.0           0.001504
5724   8.1657      5459.0           0.001496
9013   9.1228      6214.0           0.001468
```

Problem–4 GroupByExercises:

```
total_value = df['MedHouseVal'].sum()
region_value = df.groupby('Region')['MedHouseVal'].sum()
region_percent = (region_value / total_value) * 100
print(region_percent)

# Age group
df['AgeGroup'] = pd.cut(df['HouseAge'], bins=[0, 20, 40,
df['HouseAge'].max()], labels=['New', 'Mid', 'Old'])
age_counts = df['AgeGroup'].value_counts(normalize=True) * 100
print(age_counts)
```

```
Region
Central     5.195671
North      36.267665
South      58.536663
```

```
Name: MedHouseVal, dtype: float64
AgeGroup
Mid    50.721899
New    30.489341
Old    18.788760
Name: proportion, dtype: float64
```

4 Exercises on Numpy: Problem 1 – Array Creation

```
a = np.arange(20)
b = a.reshape(4, 5)
identity = np.eye(5)
filled = np.full((3, 3), 7)
print(a, b, identity, filled, sep='\n\n')
```

```
[ 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19]

[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]]

[[1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0.]
 [0. 0. 1. 0. 0.]
 [0. 0. 0. 1. 0.]
 [0. 0. 0. 0. 1.]]

[[7 7 7]
 [7 7 7]
 [7 7 7]]
```

Problem 2– Basic Operations:

```
A = np.random.randint(0,  10,  (3,  3))
B = np.random.randint(0,  10,  (3,  3))

print("A+B:\n", A + B)
print("A*B:\n", A * B)
print("A/B:\n", A / B)
print("A @ B:\n", A @ B)
print("Stats A: ", np.mean(A), np.median(A), np.std(A), np.sum(A))
print("Stats B: ", np.mean(B), np.median(B), np.std(B), np.sum(B))
```

```
A+B:
 [[ 5 13 11]
 [ 3 12  3]
 [10  7 10]]
A*B:
```

```
 [[ 6 42 30]
 [ 2 32  2]
 [16 10 21]]
A/B:
 [[0.66666667 0.85714286 0.83333333]
 [0.5        2.         0.5       ]
 [4.         0.4        0.42857143]]
A @ B:
 [[28 63 59]
 [21 44 29]
 [34 79 73]]
Stats A:  4.0 3.0 2.6666666666666665 36
Stats B:  4.222222222222222 4.0 1.9876159799998132 38
```

Problem 3– Indexing and Slicing:

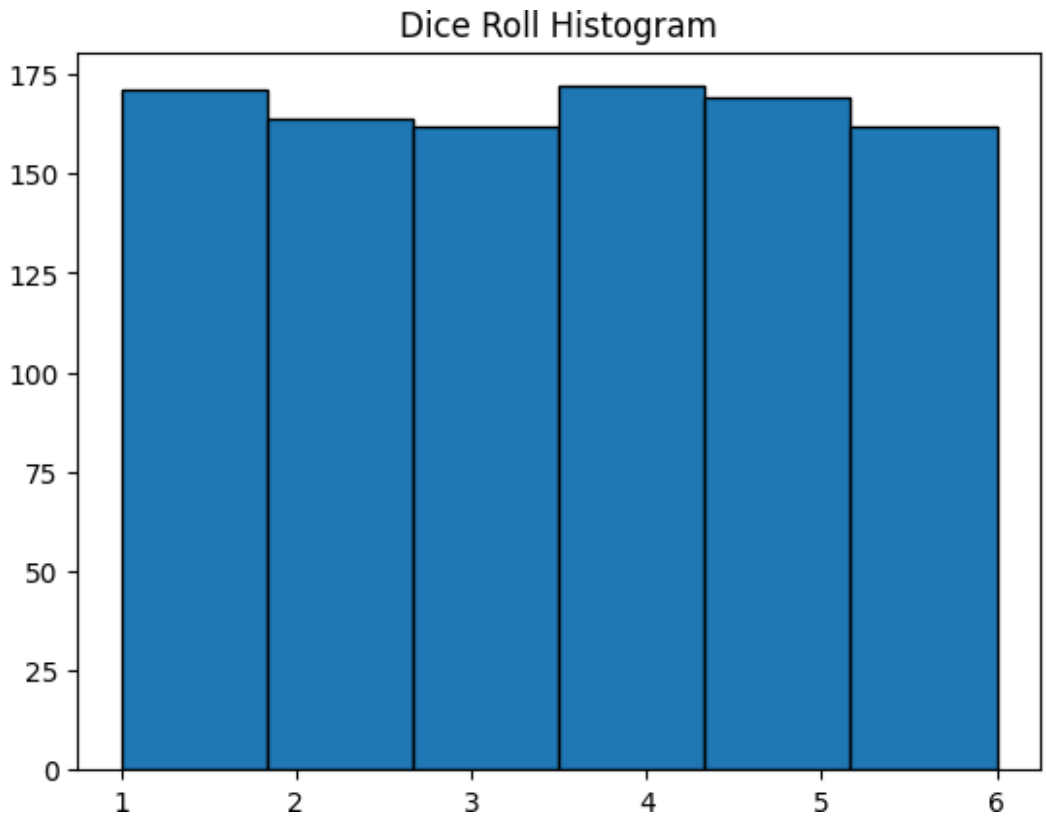
```
print("First two rows of A:\n", A[:2])
print("Elements > 5:\n", A[A > 5])
A[A % 2 == 0] = -1
print("A with evens replaced:\n", A)

First two rows of A:
 [[2 6 5]
 [1 8 1]]
Elements > 5:
 [6 8 8]
A with evens replaced:
 [[-1 -1  5]
 [ 1 -1  1]
 [-1 -1  3]]
```

1. Numpy: Advanced Exercises:

Advanced: Broadcasting, Simulation, Vectorization

```
# Broadcasting
col = np.arange(1, 4).reshape(3, 1)
row = np.arange(1, 5).reshape(1, 4)
print("Broadcasted table:\n", col * row)

# Vectorization
def square_loop(arr):
    return [x**2 for x in arr]

def square_np(arr):
    return np.square(arr)

arr = np.arange(1, 1000)
```

```
%timeit square_loop(arr)
%timeit square_np(arr)

# Coin toss
coin = np.random.choice(['H', 'T'], size=1000)
print("Heads proportion:", np.mean(coin == 'H'))

# Dice
dice = np.random.randint(1, 7, 1000)
plt.hist(dice, bins=6, edgecolor='black')
plt.title("Dice Roll Histogram")
plt.show()

# Solve equations: 3x + y = 9, x + 2y = 8
A = np.array([[3, 1], [1, 2]])
b = np.array([9, 8])
sol = np.linalg.solve(A, b)
print("Solution:", sol)

Broadcasted table:
 [[ 1  2  3  4]
 [ 2  4  6  8]
 [ 3  6  9 12]]
148 µs ± 40.6 µs per loop (mean ± std. dev. of 7 runs, 10000 loops
each)
1.63 µs ± 459 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops
each)
Heads proportion: 0.482
```



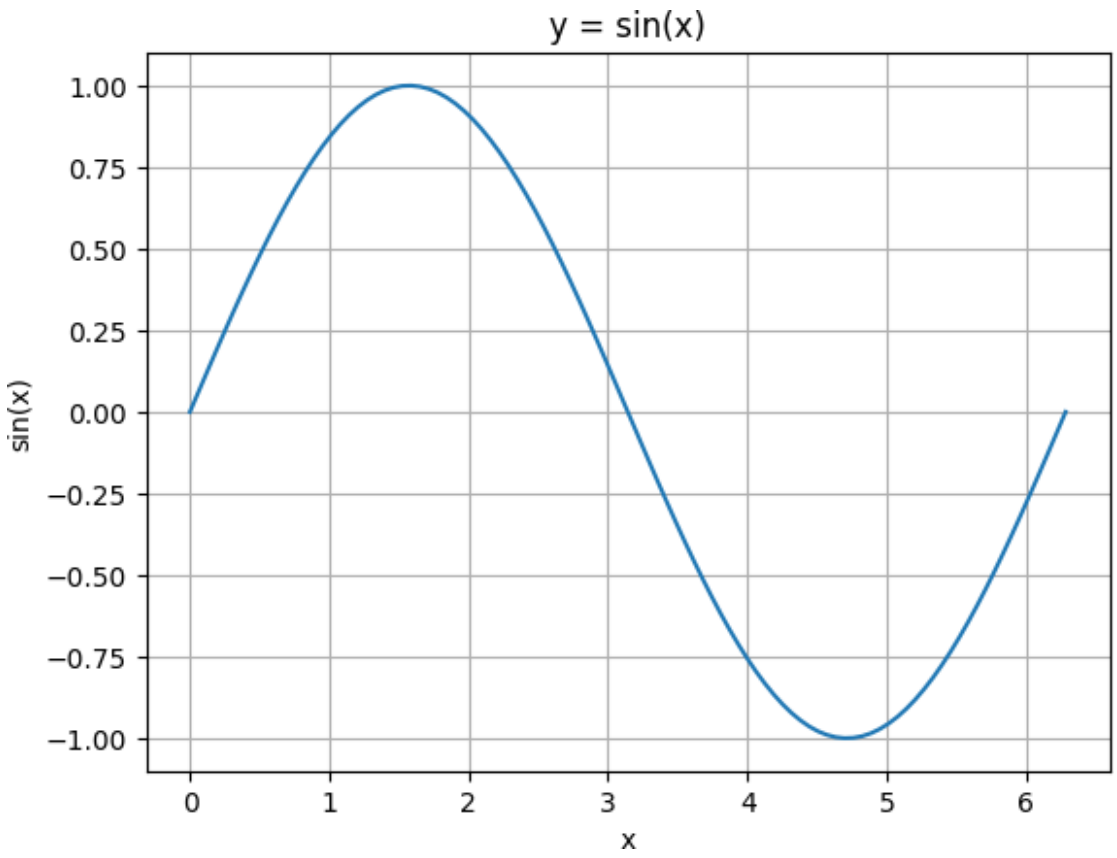

```
Solution: [2. 3.]
```

5 Exercises on Visualization with Matplotlib or Seaborn:

Problem 1– Basic Plotting with Matplotlib

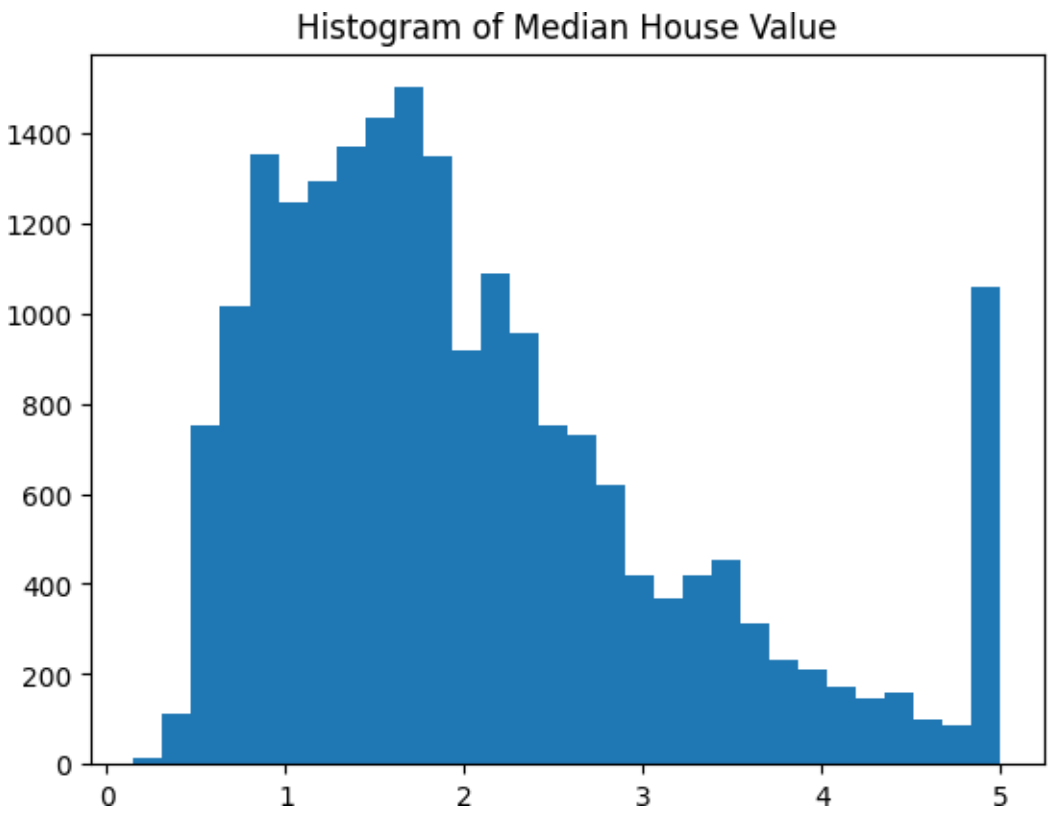
1. Generate a line plot of the function y = sin(x) over the interval [0,2π].
2. Customize the plot with title, axis labels, and grid.
3. Save the plot to a file.

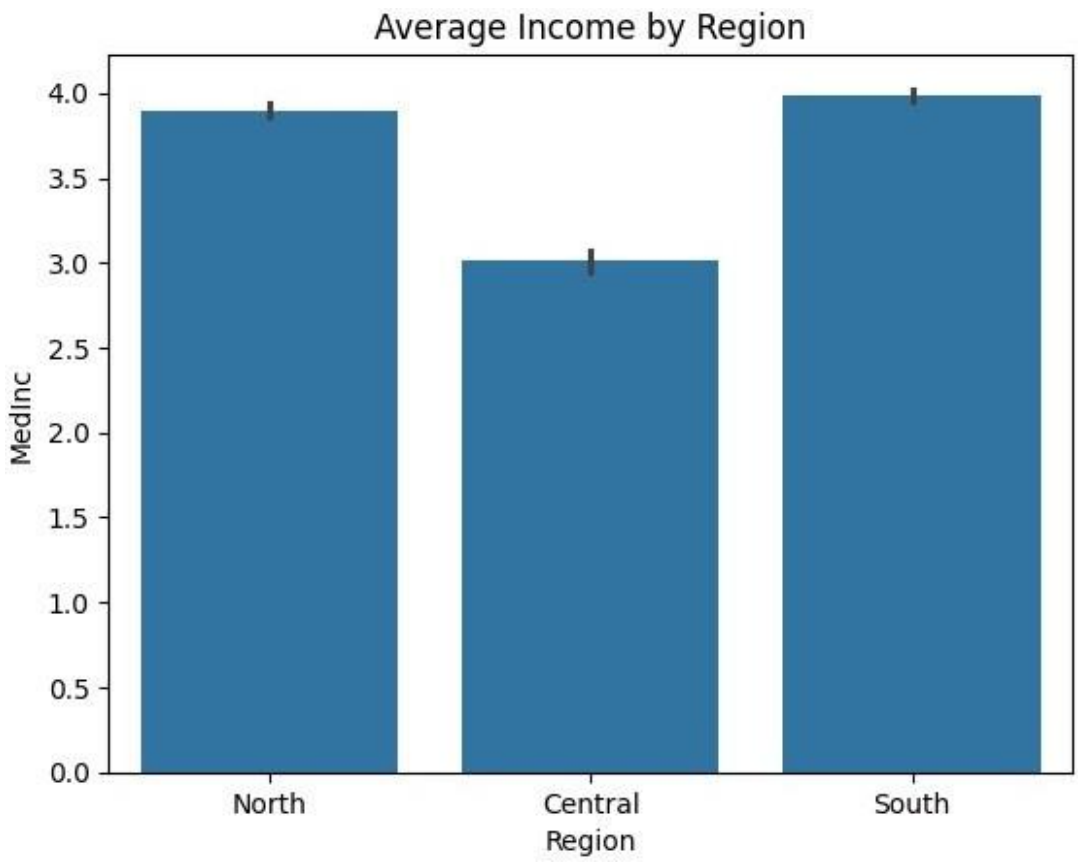
```
x = np.linspace(0, 2*np.pi, 100)
y = np.sin(x)

plt.plot(x, y)
plt.title("y = sin(x)")
plt.xlabel("x")
plt.ylabel("sin(x)")
plt.grid(True)
plt.savefig("sine_plot.png")
plt.show()
```



Histograms and Bar Plots

```
plt.hist(df['MedHouseVal'], bins=30)
plt.title("Histogram of Median House Value")
plt.show()

sns.barplot(x=df['Region'], y=df['MedInc'], estimator=np.mean)
plt.title("Average Income by Region")
plt.show()
```

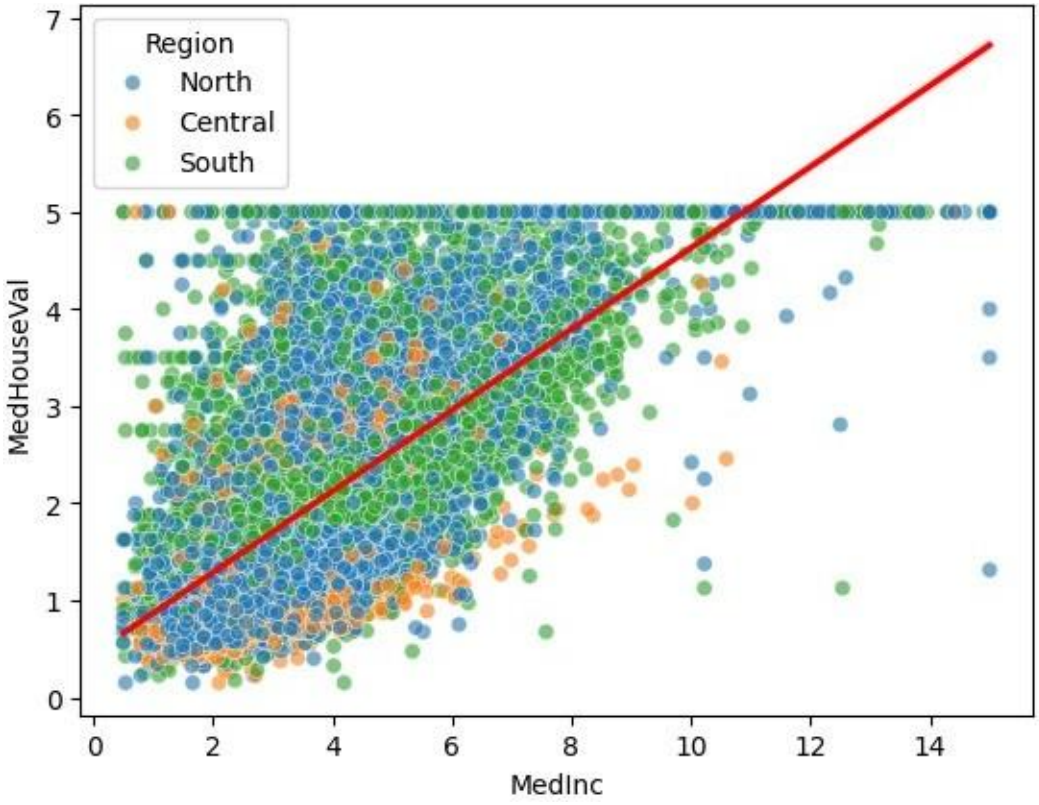

Histogram of Median House Value
1400
1200
1000
800
600
400
200
0
0
1
2
3
4
5

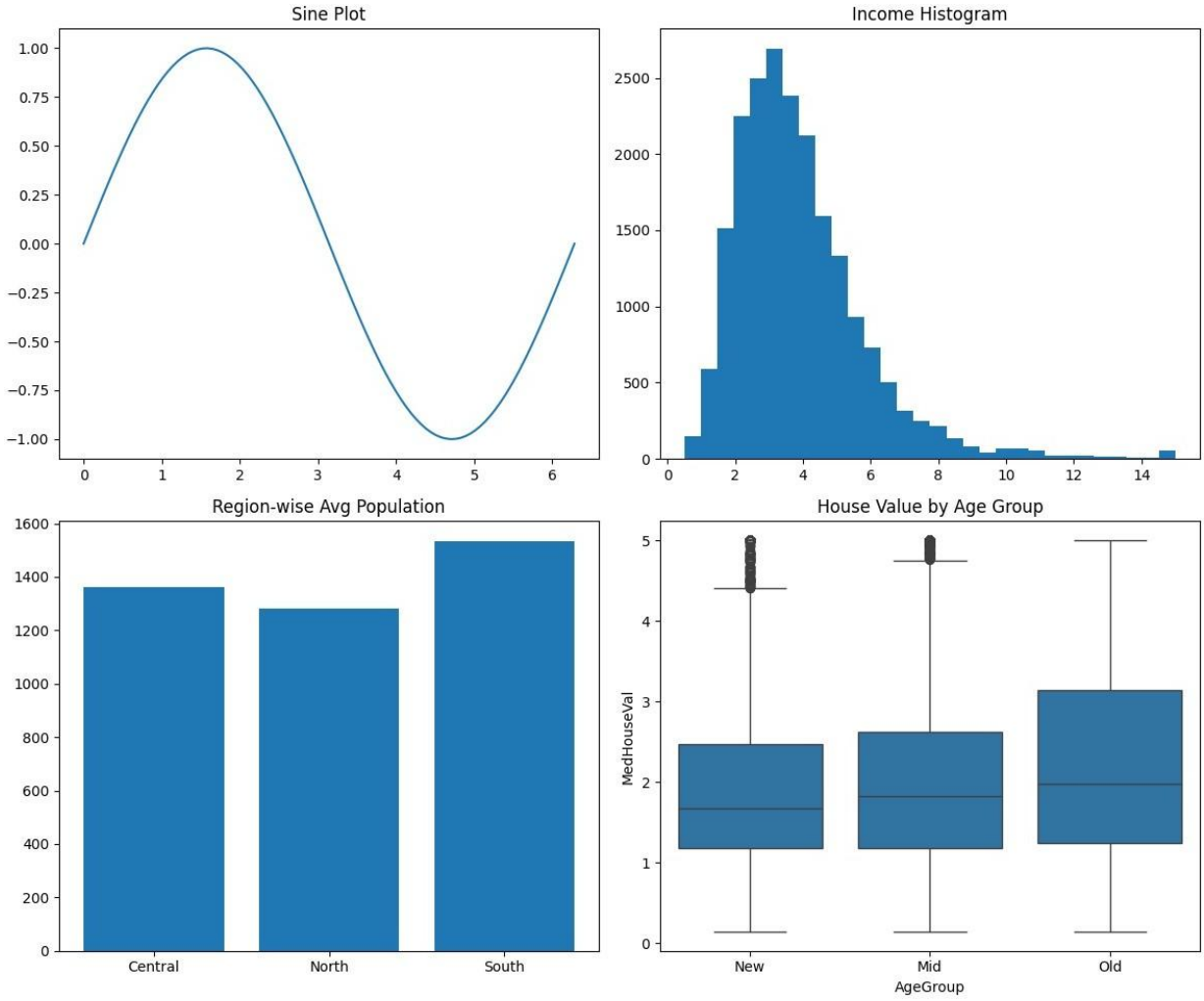


Problem 3– Scatter Plots

```
sns.scatterplot(data=df, x='MedInc', y='MedHouseVal', hue='Region',
alpha=0.6)
sns.regplot(data=df, x='MedInc', y='MedHouseVal', scatter=False,
color='red')
plt.show()

# Subplots
fig, axs = plt.subplots(2, 2, figsize=(12, 10))

axs[0, 0].plot(x, y)
axs[0, 0].set_title("Sine Plot")

axs[0, 1].hist(df['MedInc'], bins=30)
axs[0, 1].set_title("Income Histogram")

region_pop = df.groupby('Region')['Population'].mean()
axs[1, 0].bar(region_pop.index, region_pop.values)
axs[1, 0].set_title("Region-wise Avg Population")

sns.boxplot(x='AgeGroup', y='MedHouseVal', data=df, ax=axs[1, 1])
axs[1, 1].set_title("House Value by Age Group")
```

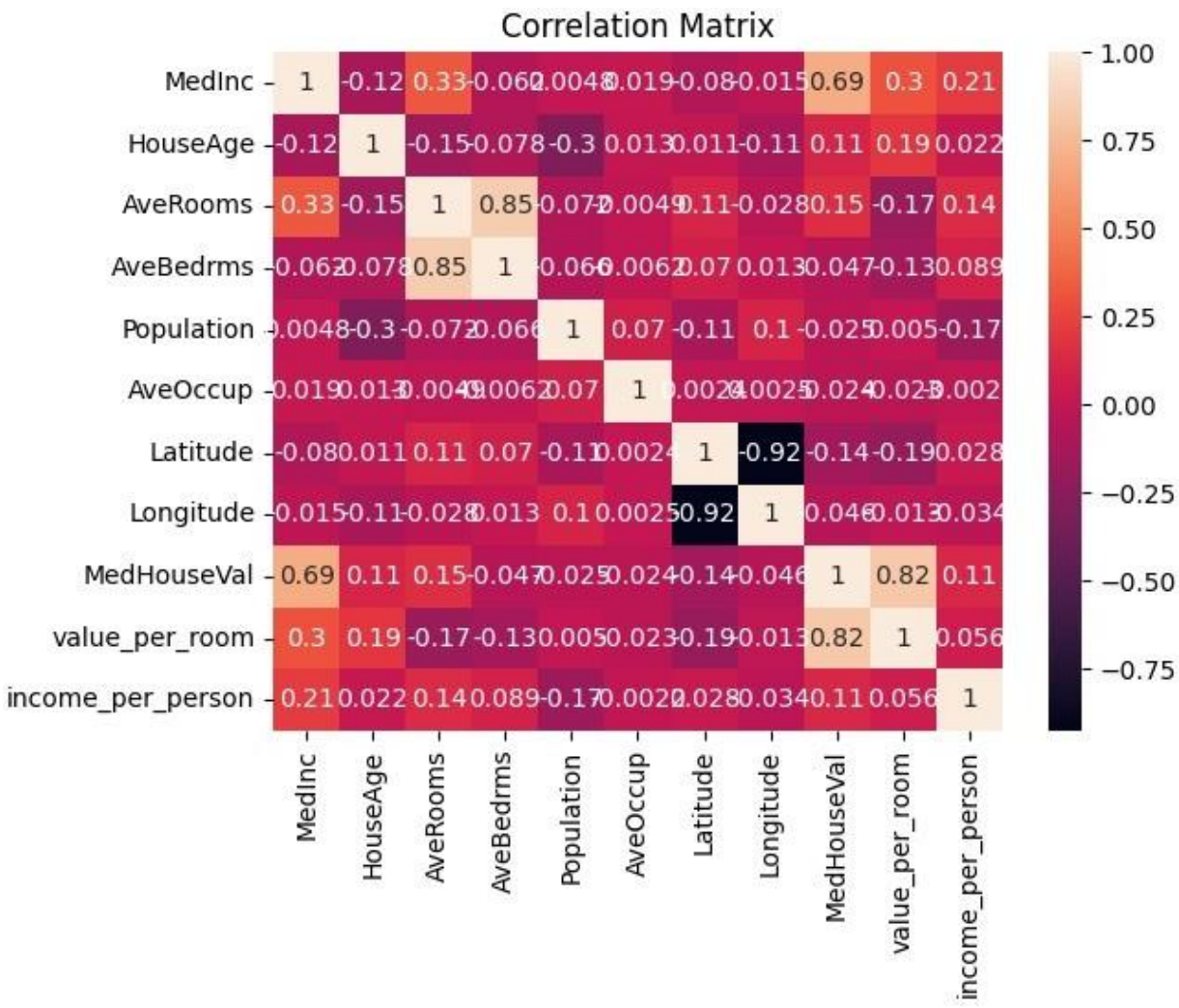
```
plt.tight_layout()
plt.show()
```



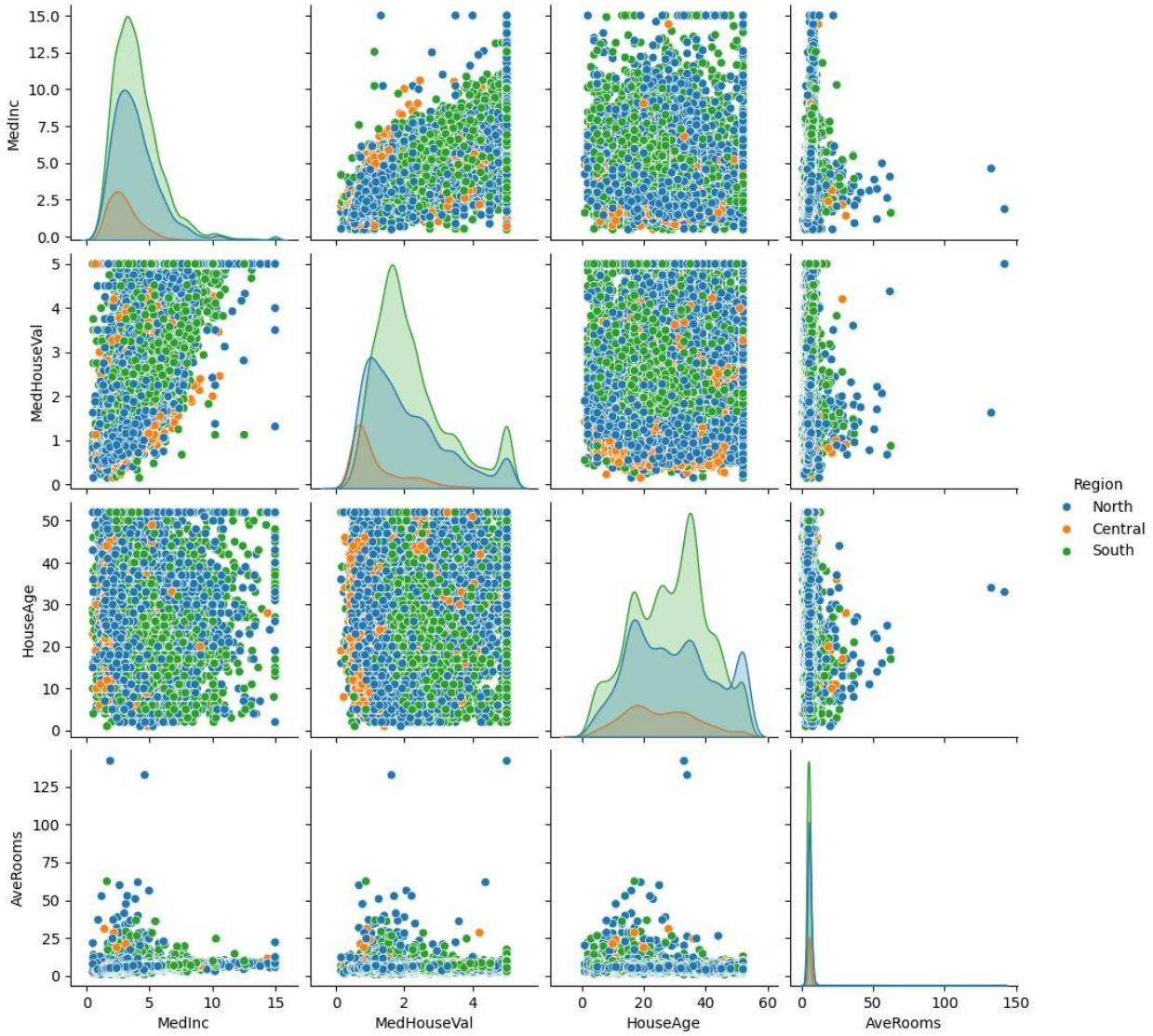


1. Advanced Exercise: Visualization

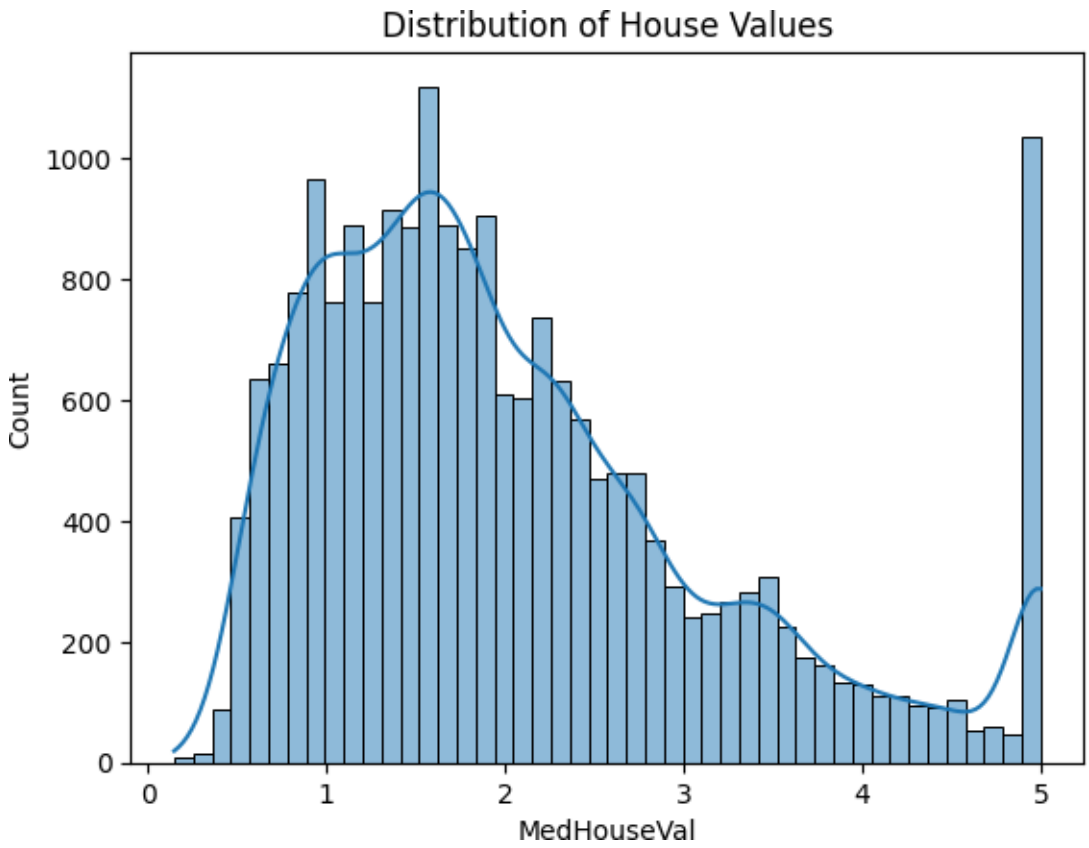
```
 # Heatmap
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True)
plt.title("Correlation Matrix")
plt.show()
```

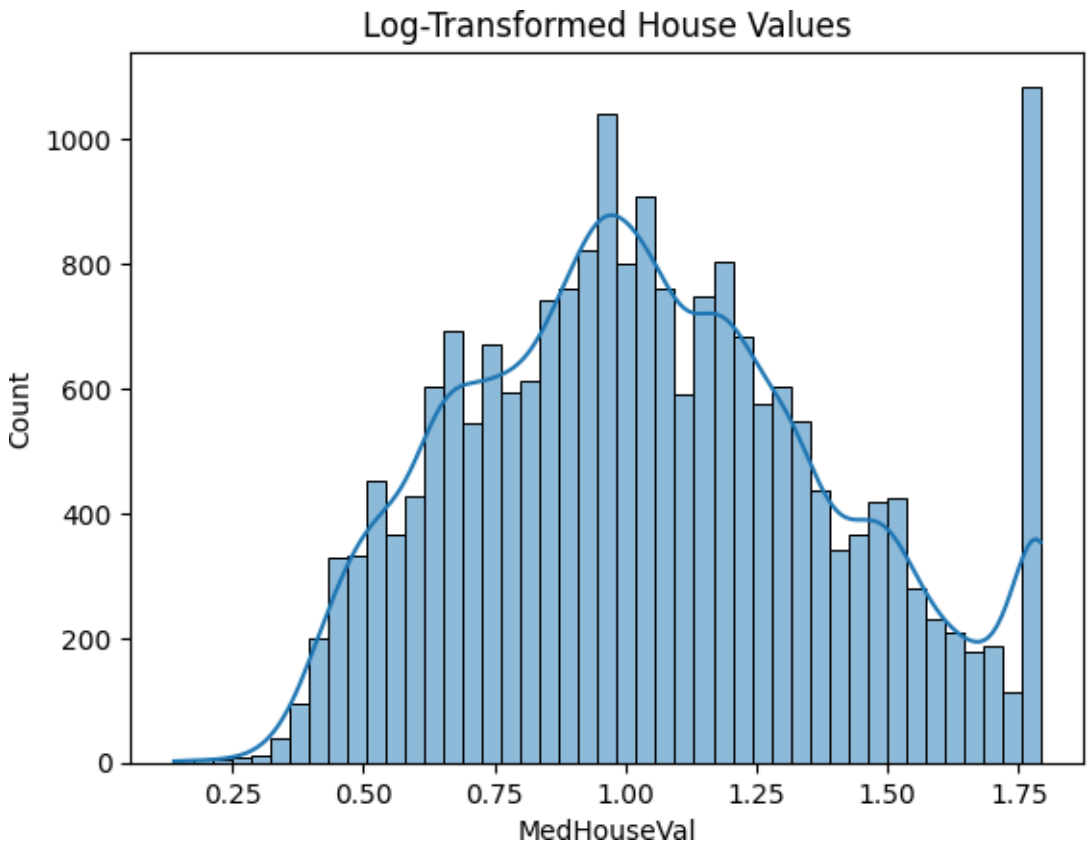



```
# Pairplot
sns.pairplot(df[['MedInc', 'MedHouseVal', 'HouseAge', 'AveRooms',
'Region']], hue='Region')
plt.show()
```




```
import seaborn as sns

# Distribution Analysis
sns.histplot(df['MedHouseVal'], kde=True)
plt.title("Distribution of House Values")
plt.show()

sns.histplot(np.log1p(df['MedHouseVal']), kde=True)
plt.title("Log-Transformed House Values")
plt.show()
```


Distribution of House Values
Count
MedHouseVal
0
200
400
600
800
1000
0
1
2
3
4
5


Log-Transformed House Values
Count
1000
800
600
400
200
0
0.25
0.50
0.75
1.00
1.25
1.50
1.75
MedHouseVal