Name: Monika Kaphle

Pre-requisites

Week:01

HCAI5DS02_Monika_Kaphle_Pre_Requisites.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

1. Dataset Setup:

[1]
```
import pandas as pd
```

[2]
```
from sklearn.datasets import fetch_california_housing
data = fetch_california_housing(as_frame=True)
df = data.frame
```

[3]
```
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
```

[4]
```
df.describe()
```

[4]
```
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

[5]
```
# Q1 MedInc Seprated from the whole database.
med_income=df['MedInc']
med_income.head(5)
```

	MedInc
0	8.3252
1	8.3014
2	7.2574
3	5.6431
4	3.8462

dtype: float64

[6]
```
# Q2 Create a DataFrame pop lat with columns Population and Latitude.
pop_lat=df[['Population','Latitude']]
pop_lat.head(5)
```

	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

Next steps: Generate code with pop_lat | View recommended plots | New interactive sheet

[7]
```
# 3. Create a DataFrame house age rooms with columns HouseAge and AveRooms.
house_age_rooms=df[['HouseAge','AveRooms']]
house_age_rooms.head(5)
```

	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

Next steps: Generate code with house_age_rooms | View recommended plots | New interactive sheet

Problem 2 – Subsetting

[8]
```
# 1. Filter houses where MedInc > 8.0, save as high income.
high_income=df[df['MedInc']>8.0]
high_income.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
131	11.6017	18.0	8.335052	1.082474	533.0	2.747423	37.84	-122.19	3.926
134	8.2049	28.0	6.978947	0.968421	463.0	2.436842	37.83	-122.19	3.352
135	8.4010	26.0	7.530806	1.056872	542.0	2.568720	37.83	-122.20	3.512

Next steps: Generate code with high_income | View recommended plots | New interactive sheet

```
[9] # 2. Filter houses where Latitude > 37, save as north california.
north_california=df[df['Latitude']>37]
north_california.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

Next steps: Generate code with north_california | View recommended plots | New interactive sheet

```
[10] # 3. Filter houses where AveRooms > 6.0 and AveOccup < 2.0, save as spacious low occupancy.
spacious_low_occupancy=df[(df['AveRooms']>6)&(df['AveOccup']<2)]
spacious_low_occupancy.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
418	6.1593	41.0	6.215470	1.077348	356.0	1.966851	37.89	-122.25	3.440
648	6.5095	25.0	6.631579	0.894737	340.0	1.988304	37.72	-122.13	3.712
710	2.4196	26.0	8.518248	2.700730	253.0	1.846715	37.68	-122.08	2.750
1024	3.1500	16.0	29.852941	5.323529	202.0	1.980392	38.52	-120.00	1.406
1102	2.4028	17.0	31.777778	9.703704	47.0	1.740741	40.06	-121.54	0.675

Next steps: Generate code with spacious_low_occupancy | View recommended plots | New interactive sheet

Subsetting Categorical Equivalents:

```
#1. Create a new column Region based on Latitude values:
# 'North' if Latitude > 37
# • 'Central' if 35 < Latitude ≤ 37
# • 'South' otherwise

df['Region']=df['Latitude'].apply(lambda x: 'North' if x>37 else ('Central' if 35<x<=37 else 'South'))
df[['Latitude','Region']].head()
```

	Latitude	Region
0	37.88	North
1	37.86	North
2	37.85	North
3	37.85	North
4	37.85	North

```
[12] # 2. Filter houses where Region is 'North' or 'Central', save as north central region
north_central_region=df[(df['Region']=='North')|(df['Region']=='Central')]
north_central_region.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526	North
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585	North
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521	North
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413	North
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422	North

Problem – 3 Exploratory Data Analysis:

Q1. Which house has the highest value per room?

[13]
```
# 1. Create a new column value per room = MedHouseVal / AveRooms.
df['value_per_room']=df['MedHouseVal']/df['AveRooms']
df[['MedHouseVal','AveRooms','value_per_room']].head()
```

	MedHouseVal	AveRooms	value_per_room
0	4.526	6.984127	0.648041
1	3.585	6.238137	0.574691
2	3.521	8.288136	0.424824
3	3.413	5.817352	0.586693
4	3.422	6.281853	0.544744

[14]
```
# 2. Filter rows where value per room > 1, save as high vpr.
high_vpr=df[df['value_per_room']>1]
high_vpr.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room
89	1.2434	52.0	2.929412	0.917647	396.0	4.658824	37.80	-122.27	5.00001	North	1.706831
104	2.8406	34.0	3.281722	1.069871	2689.0	1.503074	37.81	-122.26	3.35700	North	1.022939
395	1.7375	37.0	3.312771	1.006494	2556.0	2.766234	37.88	-122.34	3.50000	North	1.056517
458	0.9490	52.0	2.524109	0.964361	2016.0	4.226415	37.87	-122.25	3.50000	North	1.386628
459	1.1696	52.0	2.436000	0.944000	1349.0	5.396000	37.87	-122.25	5.00001	North	2.052549

[15]
```
#3. Sort high vpr by descending value per room, save as high vpr sorted.
high_vpr_sorted=high_vpr.sort_values(by='value_per_room',ascending=False)
high_vpr_sorted.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room
15660	2.3304	26.0	1.824719	1.139326	628.0	1.411236	37.78	-122.42	5.00001	North	2.740153
15654	1.4552	52.0	1.902087	1.059390	1007.0	1.616372	37.79	-122.40	4.50000	North	2.365823
4559	4.0972	52.0	2.148148	1.925926	41.0	1.518519	34.05	-118.26	5.00001	South	2.327591
15652	0.9000	52.0	2.237474	1.053535	3260.0	2.237474	37.80	-122.41	5.00001	North	2.234667
15661	0.8543	27.0	2.297872	1.175532	1211.0	1.610372	37.78	-122.42	5.00001	North	2.175930

Next steps: Generate code with high_vpr_sorted | View recommended plots | New interactive sheet

[16]
```
# 4. Display the top 5 rows with columns MedHouseVal, AveRooms, and value per room
high_vpr_sorted[['MedHouseVal','AveRooms','value_per_room']].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

Which house has the highest value per room?

[17]
```
high_vpr_sorted.head(1)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room
15660	2.3304	26.0	1.824719	1.139326	628.0	1.411236	37.78	-122.42	5.00001	North	2.740153

Next steps: Generate code with high_vpr_sorted | View recommended plots | New interactive sheet

15660 number house has the highest value per room as by calculatio=ng data above.

Q2.Among high-population areas (Population > 5000), which have the highest median income per person?

[18]
```
# 1. Create a column income per person = MedInc / Population.
df['income_per_person']=df['MedInc']/df['Population']
df[['MedInc','Population','income_per_person']].head()
```

	MedInc	Population	income_per_person
0	8.3252	322.0	0.025855
1	8.3014	2401.0	0.003457
2	7.2574	496.0	0.014632
3	5.6431	558.0	0.010113
4	3.8462	565.0	0.006807

[19]
```
# 2. Filter rows where Population > 5000, save as dense areas.
dense_areas=df[df['Population']>5000]
dense_areas.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room	income_per_person
570	7.6110	5.0	6.855776	1.061442	7427.0	2.732524	37.72	-122.24	3.507	North	0.511539	0.001025
576	7.2634	12.0	7.133034	1.018934	5781.0	2.880419	37.77	-122.06	3.416	North	0.478899	0.001256
780	3.8171	18.0	5.119733	1.043679	5613.0	2.884378	37.63	-122.10	1.872	North	0.365644	0.000680
799	2.5158	22.0	4.006152	1.036227	5436.0	3.715653	37.64	-122.07	1.349	North	0.336732	0.000463
864	5.8322	14.0	5.689849	1.051282	8117.0	2.851071	37.57	-122.01	2.818	North	0.495268	0.000719

Next steps: Generate code with dense_areas | View recommended plots | New interactive sheet

[20]
```
# 3. Sort dense areas by descending income per person, save as rich dense areas.
rich_dense_areas=dense_areas.sort_values(by='income_per_person',ascending=False)
rich_dense_areas.head()
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room	income_per_person
9004	9.1232	20.0	7.621867	1.031467	5452.0	2.907733	34.13	-118.60	4.72000	South	0.619271	0.001673
20427	8.6499	4.0	7.236059	1.032528	5495.0	2.553439	34.19	-118.80	5.00001	South	0.690985	0.001574
9027	7.7848	19.0	7.358491	1.201565	5175.0	2.381500	34.02	-118.88	5.00001	South	0.679489	0.001504
5724	8.1657	32.0	7.216658	1.029747	5459.0	2.706495	34.18	-118.26	5.00001	South	0.692843	0.001496
9013	9.1228	17.0	7.811143	1.041549	6214.0	2.933900	34.16	-118.67	5.00001	South	0.640112	0.001468

Next steps: Generate code with rich_dense_areas | View recommended plots | New interactive sheet

```
[21] # 4. Display the top 5 rows with MedInc, Population, and income per person
rich_dense_areas[['MedInc','Population','income_per_person']].head()
```

	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

Among high-population areas (Population > 5000), which have the highest median income per person?

```
[22] rich_dense_areas.head(1)
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal	Region	value_per_room	income_per_person
9004	9.1232	20.0	7.621867	1.031467	5452.0	2.907733	34.13	-118.6	4.72	South	0.619271	0.001673

Next steps: Generate code with rich_dense_areas | View recommended plots | New interactive sheet

Among high-population areas 9004 have the highest median income per person

Problem – 4 Group By Exercises:

Q1. What percent of total house value comes from each Region?

```
[23] # 1. Calculate total MedHouseVal for all houses.
total_house_value=df['MedHouseVal'].sum()
print('Total MedHouseVal for all houses is',total_house_value)
```

```
Total MedHouseVal for all houses is 42695.04061
```

```
[25] # 2. Group by Region and sum MedHouseVal.
region_house_value=df.groupby('Region')['MedHouseVal'].sum()
display(region_house_value)
```

Region	MedHouseVal
Central	2218.29405
North	15484.49440
South	24992.25216

dtype: float64

```python
[26] # 3. Divide each region's total by the overall total to get percentage contributions.
region_house_value_percentage=(region_house_value/total_house_value)*100
display(region_house_value_percentage)
```

	MedHouseVal
Region	
Central	5.195671
North	36.267665
South	58.536663

dtype: float64

What percent of total house value comes from each Region?

```python
[27] region_house_value_percentage.head()
```

	MedHouseVal
Region	
Central	5.195671
North	36.267665
South	58.536663

dtype: float64

5.19% of total house value comes from Central, 36.26% of total house value comes from Nortn and 58.53% of total house value comes from South.

Q2. What percent of total houses belong to different age groups?

```python
[28] #1. Define AgeGroup based on HouseAge:
# 'New': HouseAge < 20
# 'Mid': 20 ≤ HouseAge < 40
# 'Old': HouseAge ≥ 40
df['AgeGroup']=df['HouseAge'].apply(lambda x: 'New' if x<20 else ('Mid' if 20<=x<40 else 'Old'))
df[['HouseAge','AgeGroup']].head()
```

	HouseAge	AgeGroup
0	41.0	Old
1	21.0	Mid
2	52.0	Old
3	52.0	Old
4	52.0	Old

```python
[29] # 2. Count total houses.
total_houses=len(df)
print('Total houses are',total_houses)
```

```
Total houses are 20640
```

[30]
```python
# 3. Group by AgeGroup and count.
age_group_count=df.groupby('AgeGroup')['HouseAge'].count()
print(age_group_count)
```

```
AgeGroup
Mid    10630
New     5828
Old     4182
Name: HouseAge, dtype: int64
```

[31]
```python
# 4. Compute percentage shares for each group.
age_group_percentage=(age_group_count/total_houses)*100
print(age_group_percentage)
```

```
AgeGroup
Mid    51.501938
New    28.236434
Old    20.261628
Name: HouseAge, dtype: float64
```

What percent of total houses belong to different age groups?

= 51.5% of total houses belong to Mid age, 28.23% of total houses belong to New age and 20.26% of total houses belong to Old age.

4 Exercises on Numpy:

1. Numpy Foundations - Warm Up Exercises:

Problem 1 – Array Creation:

[32]
```python
import numpy as np
```

[33]
```python
#1. Create a 1D NumPy array containing integers from 0 to 19
arr1=np.arange(20)
```

[34]
```python
# 2. Reshape it into a 4x5 matrix
arr2=arr1.reshape(4, 5)
print(arr2)
```

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

[35]
```python
# 3. Generate a 5x5 identity matrix and a 3x3 matrix filled with 7.
arr3=np.identity(5)
arr4=np.full((3,3),7)
```

Problem 2 – Basic Operations:

[36]
```python
# 1. Create two 3x3 matrices A and B with random integers (0-9)
A=np.random.randint(0,10,size=(3,3))
B=np.random.randint(0,10,size=(3,3))
```

```
[37] print('A=',A)
     print('\nB=',B)
```

```
A= [[9 3 2]
 [9 5 4]
 [7 3 9]]

B= [[1 2 6]
 [5 0 8]
 [7 8 3]]
```

```
[38] # 2. Perform:
     # Element-wise addition, multiplication, and division.
     # Matrix multiplication (A @ B).

     addition=A+B
     multiplication=A*B
     division=A/(B + 1e-10)
     matrix_multiplication=A@B
```

```
[39] # 3. Compute mean, median, standard deviation, and sum for each matrix.
     mean_A=np.mean(A)
     median_A=np.median(A)
     std_A=np.std(A)
     sum_A=np.sum(A)
```

Problem 3 – Indexing and Slicing:

```
[40] # 1. Slice the first two rows of matrix A.
     slice_A=A[:2]
     print(slice_A)
```

```
[[9 3 2]
 [9 5 4]]
```

```
[41] #2. Select elements greater than 5.
     element=A[A>5]
     print('Element greater than 5 are',element)
```

```
Element greater than 5 are [9 9 7 9]
```

```
[42] # 3. Replace all even numbers in A with -1.
     A[A%2==0]=-1
     print(A)
```

```
[[ 9  3 -1]
 [ 9  5 -1]
 [ 7  3  9]]
```

2. Numpy: Advanced Exercises:

1. Broadcasting Challenge

```
[43] #Create a 3x1 column vector and a 1x4 row vector.

     c_Vector = np.array([[1], [2], [3]])
     r_Vector = np.array([[4, 5, 6, 7]])

     print("Column Vector =\n", c_Vector)
     print("\nRow Vector =\n", r_Vector)
```

```
Column Vector =
 [[1]
 [2]
 [3]]

Row Vector =
 [[4 5 6 7]]
```

2. Vectorization vs Loops

[45]
```
#Write a function to compute element-wise square of an array using:
# a for-loop
# NumPy vectorized operation

def square_loop(arr):
    return np.array([x**2 for x in arr])

def square_vectorized(arr):
    return arr ** 2
```

```
# Compare their execution time using %%timeit or time module
import time

arr = np.random.randint(0, 100, 1000)

def square_loop(arr):
    return np.array([x**2 for x in arr])

def square_vectorized(arr):
    return arr ** 2

# Time the for-loop method
start = time.time()
square_loop(arr)
end = time.time()
print("For-loop time:", end - start)

# Time the vectorized method
start = time.time()
square_vectorized(arr)
end = time.time()
print("Vectorized time:", end - start)
```

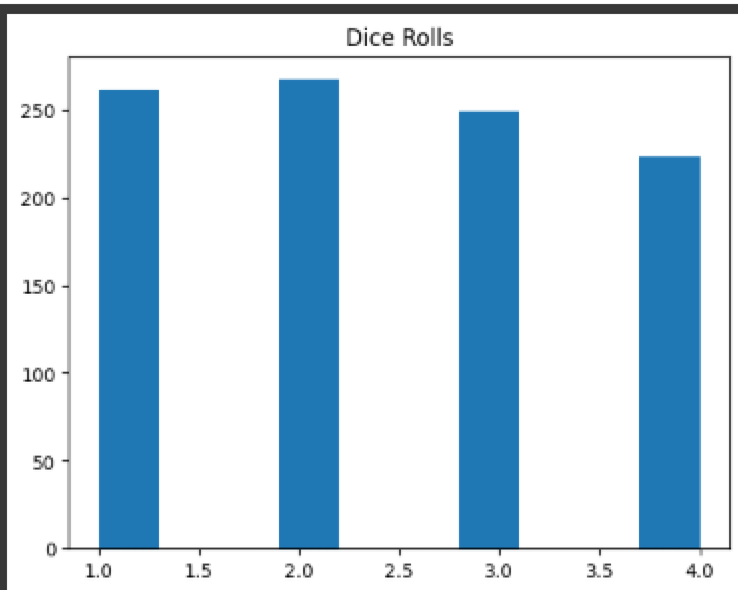
```
For-loop time: 0.0002288818359375
Vectorized time: 0.00010704994201660156
```

3. Simulation Task

[48]
```
# Simulate 1000 random coin tosses and calculate proportion of heads

tosses = np.random.choice(['H', 'T'], 1000)
proportion_heads = np.mean(tosses == 'H')

print("Proportion of heads:", proportion_heads)
```

```
Proportion of heads: 0.51
```

[49]
```
# Simulate 1000 dice rolls and plot histogram of outcomes.
import matplotlib.pyplot as plt

dice_rolls = np.random.randint(1, 5, 1000)
plt.hist(dice_rolls)
plt.title("Dice Rolls")
plt.show()
```



4. Solving Systems of Equations

```
[50] #Solve the system:
     #3x + y = 9x + 2y = 8
     # Use np.linalg.solve to find the solution

     A = np.array([[3, 1],[9, 2]])

     B = np.array([8, 8])

     # Solve the system Ax = B
     solution = np.linalg.solve(A, B)

     print("Solution =", solution)
```

```
Solution = [-2.66666667 16.        ]
```

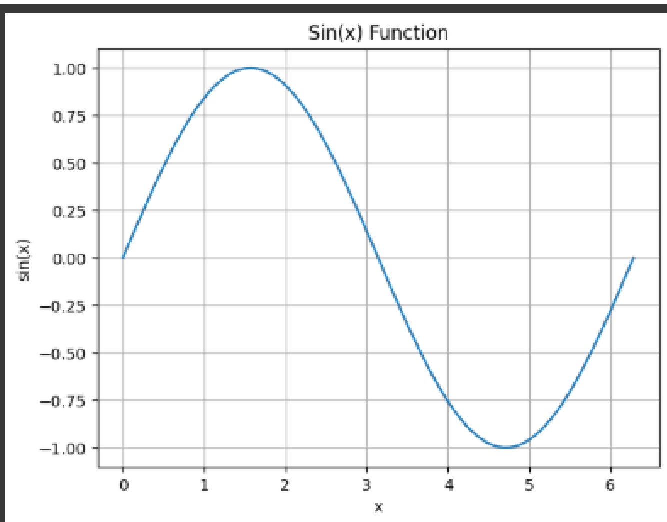
5 Exercises on Visualization with Matplotlib or Seaborn:

```
[51] import seaborn as sns
```

1. Warm - Up Exercises:

Problem 1 – Basic Plotting with Matplotlib

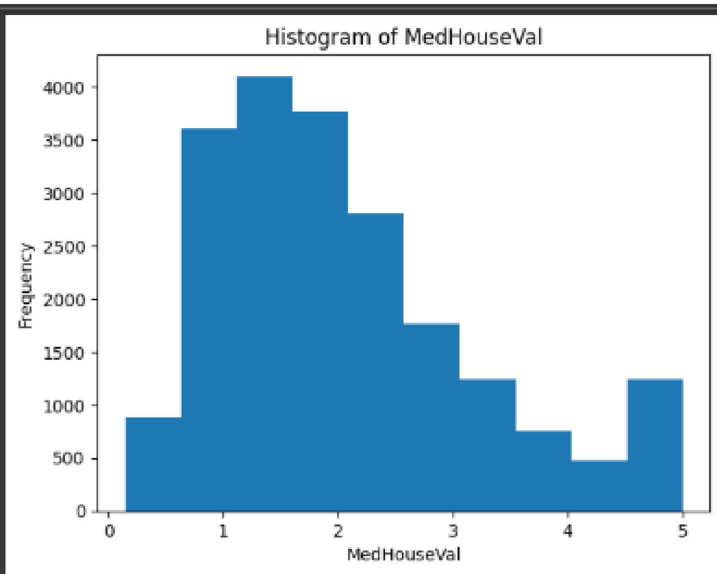
```
[52] # 1. Generate a line plot of the function y = sin(x) over the interval [0, 2π].

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

2# . Customize the plot with title, axis labels, and grid.
plt.plot(x, y)
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.title('Sin(x) Function')
plt.grid(True)
plt.show()

# prompt: 3. Save the plot to a file.

plt.savefig('sin_function_plot.png')
```




```
<Figure size 640x480 with 0 Axes>
```

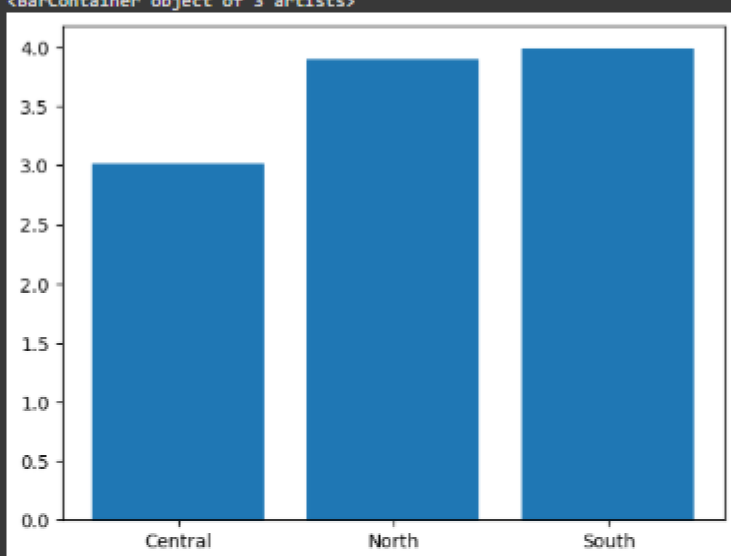
Problem 2 – Histograms and Bar Plots

```
[53] # 1. Plot a histogram of the MedHouseVal column from the California dataset.
plt.hist(df['MedHouseVal'])
plt.xlabel('MedHouseVal')
plt.ylabel('Frequency')
plt.title('Histogram of MedHouseVal')
plt.show()
```



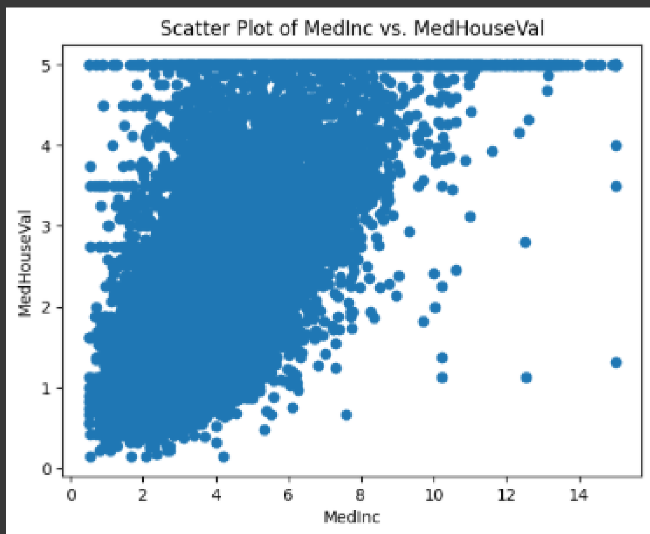

```python
# 2. Create a bar chart comparing average MedInc across Region

average_med_inc=df.groupby('Region')['MedInc'].mean()
plt.bar(average_med_inc.index,average_med_inc.values)
```

```
<BarContainer object of 3 artists>
```



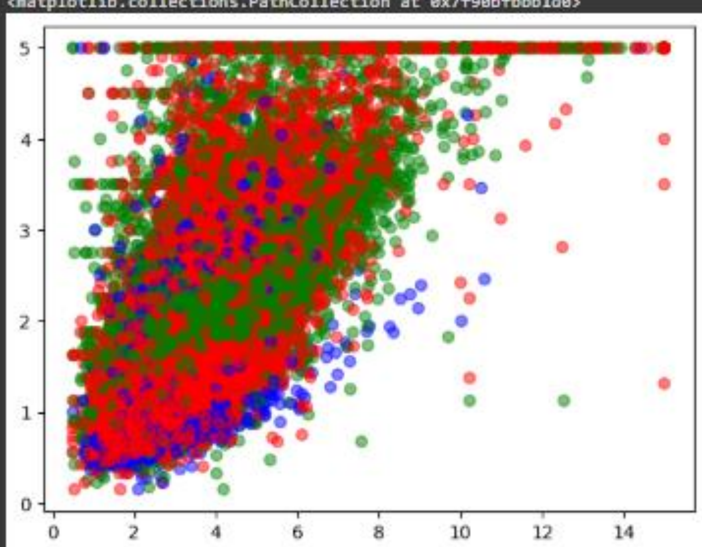
Problem 3 – Scatter Plots

[56]
```
# 1. Create a scatter plot of MedInc vs. MedHouseVal.

plt.scatter(df['MedInc'],df['MedHouseVal'])
plt.xlabel('MedInc')
plt.ylabel('MedHouseVal')
plt.title('Scatter Plot of MedInc vs. MedHouseVal')
plt.show()
```

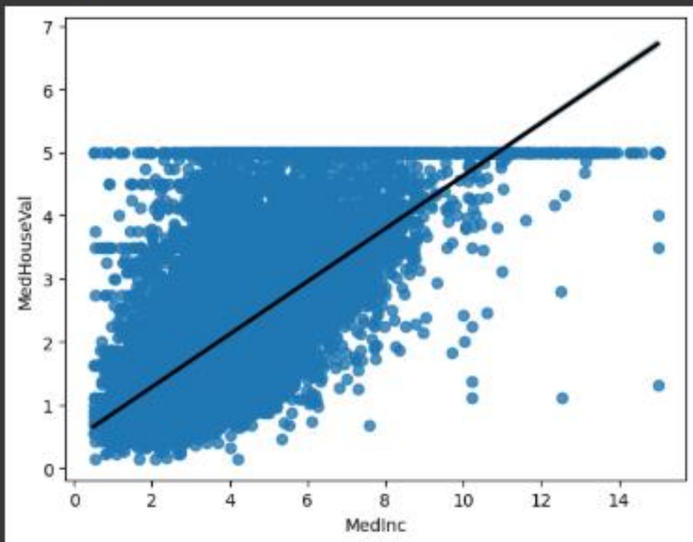


[58]
```
# 2. Color the points by Region and add transparency.

colors = {'North': 'red', 'Central': 'blue', 'South': 'green'}
plt.scatter(df['MedInc'], df['MedHouseVal'], c=df['Region'].map(colors), alpha=0.5)
```

```
<matplotlib.collections.PathCollection at 0x7f90bfbbb1d0>
```

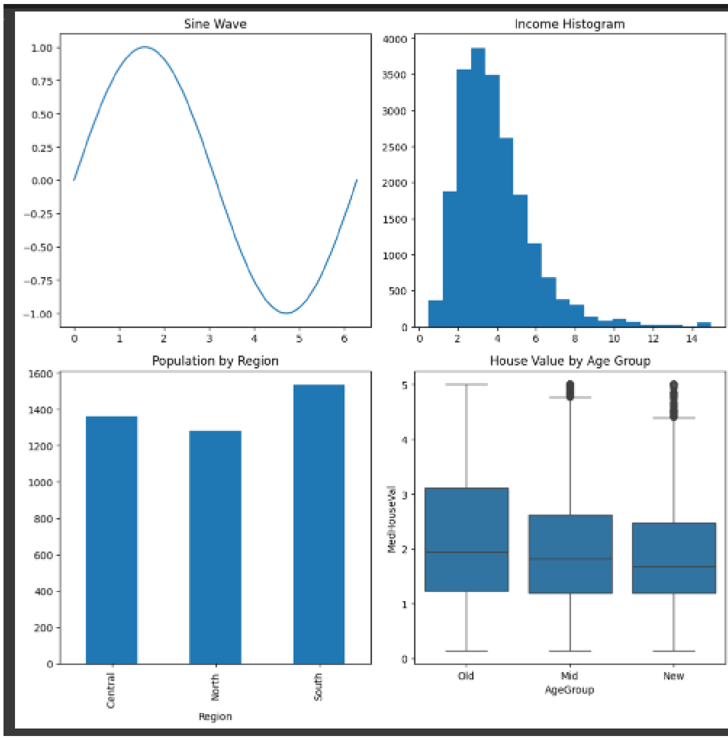


[59]
```
# 3. Add a regression line using Seaborn's regplot.

sns.regplot(x='MedInc',y='MedHouseVal',data=df)
sns.regplot(x='MedInc', y='MedHouseVal', data=df, scatter=False, color='black')
plt.show()
```



Problem 4 – Subplots

```
#1. Create a 2x2 subplot grid showing:
# Line plot of sine

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

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

# Histogram of income

axs[0, 1].hist(df['MedInc'], bins=20)
axs[0, 1].set_title("Income Histogram")

# Bar chart of region-wise population

df.groupby('Region')['Population'].mean().plot(kind='bar', ax=axs[1, 0])
axs[1, 0].set_title("Population by Region")

# Boxplot of house value grouped by age group

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()
```



2. Advanced Exercise: Visualization

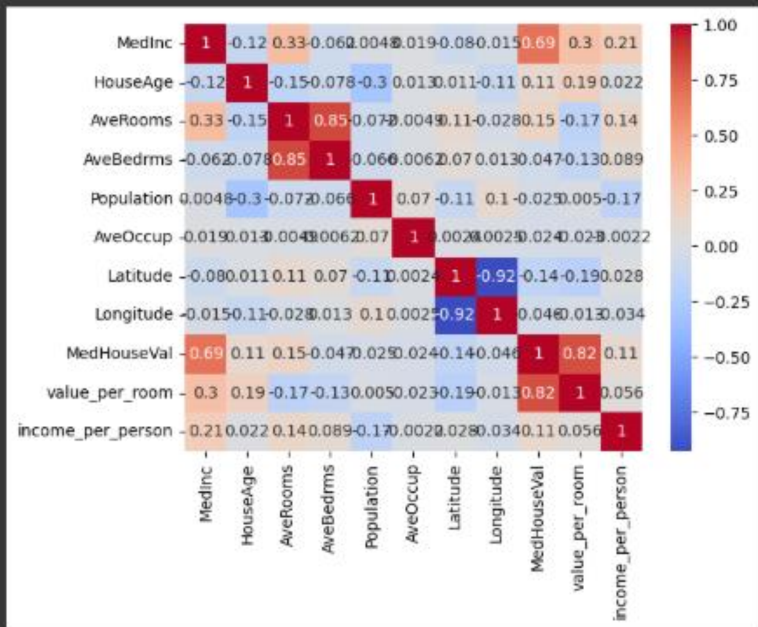
1. Heatmaps

[62]
```python
# Compute the correlation matrix of the California dataset.

correlation_Matrix = df.drop(['Region', 'AgeGroup'], axis=1).corr()
print(correlation_Matrix)
```

```
                    MedInc  HouseAge  AveRooms  AveBedrms  Population  \
MedInc            1.000000 -0.119034  0.326895  -0.062040    0.004834
HouseAge         -0.119034  1.000000 -0.153277  -0.077747   -0.296244
AveRooms          0.326895 -0.153277  1.000000   0.847621   -0.072213
AveBedrms        -0.062040 -0.077747  0.847621   1.000000   -0.066197
Population        0.004834 -0.296244 -0.072213  -0.066197    1.000000
AveOccup          0.018766  0.013191 -0.004852  -0.006181    0.069863
Latitude         -0.079809  0.011173  0.106389   0.069721   -0.108785
Longitude        -0.015176 -0.108197 -0.027540   0.013344    0.099773
MedHouseVal       0.688075  0.105623  0.151948  -0.046701   -0.024650
value_per_room    0.303433  0.194376 -0.172843  -0.128784    0.004976
income_per_person 0.213368  0.021600  0.142485   0.089145   -0.171101

                  AveOccup  Latitude  Longitude  MedHouseVal  value_per_room  \
MedInc            0.018766 -0.079809  -0.015176     0.688075        0.303433
HouseAge          0.013191  0.011173  -0.108197     0.105623        0.194376
AveRooms         -0.004852  0.106389  -0.027540     0.151948       -0.172843
AveBedrms        -0.006181  0.069721   0.013344    -0.046701       -0.128784
Population        0.069863 -0.108785   0.099773    -0.024650        0.004976
AveOccup          1.000000  0.002366   0.002476    -0.023737       -0.023478
Latitude          0.002366  1.000000  -0.924664    -0.144160       -0.190979
Longitude         0.002476 -0.924664   1.000000    -0.045967       -0.013439
MedHouseVal      -0.023737 -0.144160  -0.045967     1.000000        0.823007
value_per_room   -0.023478 -0.190979  -0.013439     0.823007        1.000000
income_per_person -0.002180  0.027979  -0.034378     0.114455        0.056247

                  income_per_person
MedInc                     0.213368
HouseAge                   0.021600
AveRooms                   0.142485
AveBedrms                  0.089145
Population                -0.171101
AveOccup                  -0.002180
Latitude                   0.027979
Longitude                 -0.034378
MedHouseVal                0.114455
value_per_room             0.056247
income_per_person          1.000000
```

[63]

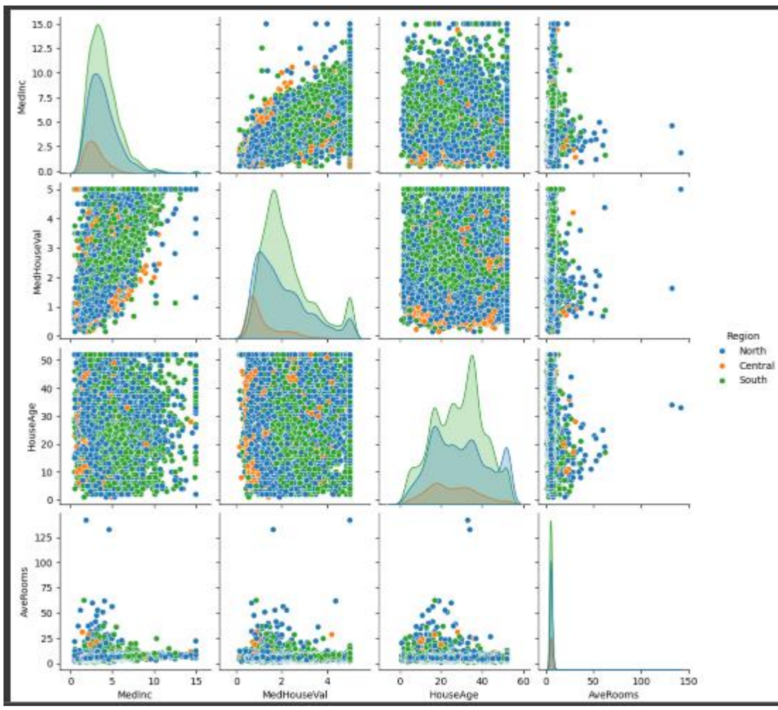
```
# Plot a heatmap using sns.heatmap with annotations.

sns.heatmap(correlation_Matrix, annot=True, cmap='coolwarm')
plt.show()
```



2. Pairplot

[64]

```
# Use Seaborn's pairplot to show pairwise relationships between MedInc, MedHouseVal, HouseAge, and AveRooms.
#Color points by Region.

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



3. Distribution Analysis

```
[65] # Use Seaborn's distplot or displot to visualize:
     #- Distribution of MedHouseVal
     #- Log-transformed version to see skewness reduction

     sns.distplot(np.log10(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()
```

```
For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751

  sns.distplot(np.log10(df['MedHouseVal']), kde=True)
```