Ananya Dahal
2408840

HCAI5DS02_AnanyaDahal_2408840_Pre-Requisite.i...

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM
Disk

Ananya Dahal

[50]
```
#Importing python package
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Exercise: Data Ananlysis with Pandas using California Housing Dataset

Dataset setup:

[2]
```
#Importing the calfornia housing dataset
from sklearn.datasets import fetch_california_housing

data = fetch_california_housing(as_frame=True)

df = data.frame
```

Warm-up Exercise:

[3]
```
df.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

Warm-up Exercise:

[3]
```
df.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 df | View recommended plots | New interactive sheet

```
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
```

```python
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 income containing only the MedInc column. Display first 5 rows.

[6]

```python
med_income = df[['MedInc']]

med_income.head()
```

	MedInc
0	8.3252
1	8.3014
2	7.2574
3	5.6431
4	3.8462

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

2. Create a DataFrame pop lat with columns Population and Latitude (in that order). Display the first 5 rows

```
pop_lat=df[['Population','Latitude']]

pop_lat.head()
```

	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

3. Create a DataFrame house age rooms with columns HouseAge and AveRooms. Display the first 5 rows.

```
house_age_rooms=df[['HouseAge','AveRooms']]

house_age_rooms.head()
```

	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

Problem-2: Subsetting

1.Filter houses where MedInc > 8.0, save as high income. Display the result.

```
[9] high_income=df[df['MedInc']>8.0]

    high_income
```

	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	4.52600
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.58500
131	11.6017	18.0	8.335052	1.082474	533.0	2.747423	37.84	-122.19	3.92600
134	8.2049	28.0	6.978947	0.968421	463.0	2.436842	37.83	-122.19	3.35200
135	8.4010	26.0	7.530806	1.056872	542.0	2.568720	37.83	-122.20	3.51200
...	...	...	...	...	...	...	...	...	...
20426	10.0472	11.0	9.890756	1.159664	415.0	3.487395	34.18	-118.69	5.00001
20427	8.6499	4.0	7.236059	1.032528	5495.0	2.553439	34.19	-118.80	5.00001
20428	8.7288	6.0	8.715842	1.102970	3385.0	3.351485	34.23	-118.83	4.25800
20436	12.5420	10.0	9.873315	1.102426	1179.0	3.177898	34.21	-118.69	5.00001
20503	8.2787	27.0	6.935065	1.103896	243.0	3.155844	34.33	-118.75	3.30000

690 rows × 9 columns

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

2. Filter houses where Latitude > 37, save as north california. Display the result.

```
[10] north_california=df[df['Latitude']>37]

     north_california
```

	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. Filter houses where Latitude > 37, save as north california. Display the result.

```
north_california=df[df['Latitude']>37]
north_california
```

	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
...	...	...	...	...	...	...	...	...	...
20635	1.5603	25.0	5.045455	1.133333	845.0	2.560606	39.48	-121.09	0.781
20636	2.5568	18.0	6.114035	1.315789	356.0	3.122807	39.49	-121.21	0.771
20637	1.7000	17.0	5.205543	1.120092	1007.0	2.325635	39.43	-121.22	0.923
20638	1.8672	18.0	5.329513	1.171920	741.0	2.123209	39.43	-121.32	0.847
20639	2.3886	16.0	5.254717	1.162264	1387.0	2.616981	39.37	-121.24	0.894

7558 rows × 9 columns

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

[12]
```
#3)Filter houses where AveRooms > 6.0 and AveOccup < 2.0, save as spacious low occupancy. Display the result.
spacious_low_occupancy=df[(df['AveRooms']>6.0) & (df['AveOccup']<2.0)]
spacious_low_occupancy
```

	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	3.44000
648	6.5095	25.0	6.631579	0.894737	340.0	1.988304	37.72	-122.13	3.71200
710	2.4196	26.0	8.518248	2.700730	253.0	1.846715	37.68	-122.08	2.75000
1024	3.1500	16.0	29.852941	5.323529	202.0	1.980392	38.52	-120.00	1.40600

[12]
```python
#3)Filter houses where AveRooms > 6.0 and AveOccup < 2.0, save as spacious low occupancy. Display the result.

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

spacious_low_occupancy
```

	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	3.44000
648	6.5095	25.0	6.631579	0.894737	340.0	1.988304	37.72	-122.13	3.71200
710	2.4196	26.0	8.518248	2.700730	253.0	1.846715	37.68	-122.08	2.75000
1024	3.1500	16.0	29.852941	5.323529	202.0	1.980392	38.52	-120.00	1.40600
1102	2.4028	17.0	31.777778	9.703704	47.0	1.740741	40.06	-121.54	0.67500
...	...	...	...	...	...	...	...	...	...
17155	7.4542	16.0	6.483333	1.066667	512.0	1.706667	37.42	-122.23	5.00001
17878	3.7614	14.0	9.363636	2.185687	813.0	1.572534	37.40	-122.01	1.37500
19362	3.3125	9.0	16.541284	3.116208	594.0	1.816514	38.70	-123.49	2.95400
20355	1.9811	16.0	6.104730	1.168919	587.0	1.983108	34.19	-118.96	1.62500
20423	5.4346	17.0	6.261168	1.505155	578.0	1.986254	34.08	-119.00	4.28600

106 rows × 9 columns

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

Subsetting Categorical Equivalents:

[14]
```python
#1) Create a new column Region based on Latitude values: • 'North' if Latitude > 37 • 'Central' if 35 < Latitude ≤ 37 • 'South' other

conditions = [
    (df['Latitude'] > 37),
    (df['Latitude'] > 35) & (df['Latitude'] <= 37),
    (df['Latitude'] <= 35),
]

choices=['North','Central','South']
```

Subsetting Categorical Equivalents:

[14]

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

conditions = [
    (df['Latitude'] > 37),
    (df['Latitude'] > 35) & (df['Latitude'] <= 37),
    (df['Latitude'] <= 35),
]

choices=['North','Central','South']

df['Region']=np.select(conditions,choices,default='South')
df.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

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

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

north_central_region
```

	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:

[17]
```python
#1) Which house has the highest value per room?
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)

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

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

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

Problem-4: Group by exercises:

```
#1)  What percent of total house value comes from each Region?
total_MedHouseVal=df['MedHouseVal'].sum()

regions_total=df.groupby('Region')['MedHouseVal'].sum()

regions_total_percentage=regions_total/total_MedHouseVal*100

regions_total_percentage
```

Region	MedHouseVal
Central	5.195671
North	36.267665
South	58.536663

dtype: float64

2. What percent of total houses belong to different age groups? Hint:

Define AgeGroup based on HouseAge: • 'New': HouseAge < 20 • 'Mid': 20 ≤ HouseAge < 40 • 'Old': HouseAge ≥ 40 a. Count total houses. b. Group by AgeGroup and count. c. Compute percentage shares for each group.

```
[20] conditions=[
         (df['HouseAge']<20),
         (df['HouseAge']>=20) & (df['HouseAge']<40),
         (df['HouseAge']>=40)
     ]
     #Define the agegroup values
     choices=['New','Mid','Old']
     #Create new column based on agegroup values
     df['AgeGroup']=np.select(conditions,choices,default='Unknown')
     #Count total houses
     total_houses=len(df)
     #Group by AgeGroup and count
     agegroup_counts=df['AgeGroup'].value_counts()
     #Calculate percentage share for each group
     agegroup_counts_percentage=agegroup_counts/total_houses*100
     #Display the result
     agegroup_counts_percentage
```

3: Advance exercises

[21]
```python
#1) Correlation Analysis
#a. Compute Pearson correlation coefficients between MedHouseVal and all other numerical features.

#b. Identify which features have the strongest positive and negative correlations with house value.

#c. Interpret these relationship

#Compute pearson coorrelation coffecient
correlation=df.corr(numeric_only=True)
#Calculate correlations of MedHouseVal with other feature
MedHouseVal_corr=correlation['MedHouseVal'].drop('MedHouseVal')
#Identify strongest negative and positive correlations
strongest_negative=MedHouseVal_corr.idxmin(),MedHouseVal_corr.min()
strongest_positive=MedHouseVal_corr.idxmax(),MedHouseVal_corr.max()
#Display the result
print("Correlation with MedHouseVal:\n",MedHouseVal_corr)
print("Strongest negative correlation:",strongest_negative)
print("Strongest positive correlation:",strongest_positive)
```

```
Correlation with MedHouseVal:
 MedInc             0.688075
HouseAge           0.105623
AveRooms           0.151948
AveBedrms         -0.046701
Population        -0.024650
AveOccup          -0.023737
Latitude          -0.144160
Longitude         -0.045967
value_per_room     0.823007
income_per_person  0.114455
Name: MedHouseVal, dtype: float64
Strongest negative correlation: ('Latitude', -0.14416027687465632)
Strongest positive correlation: ('value_per_room', 0.823006856834075)
```

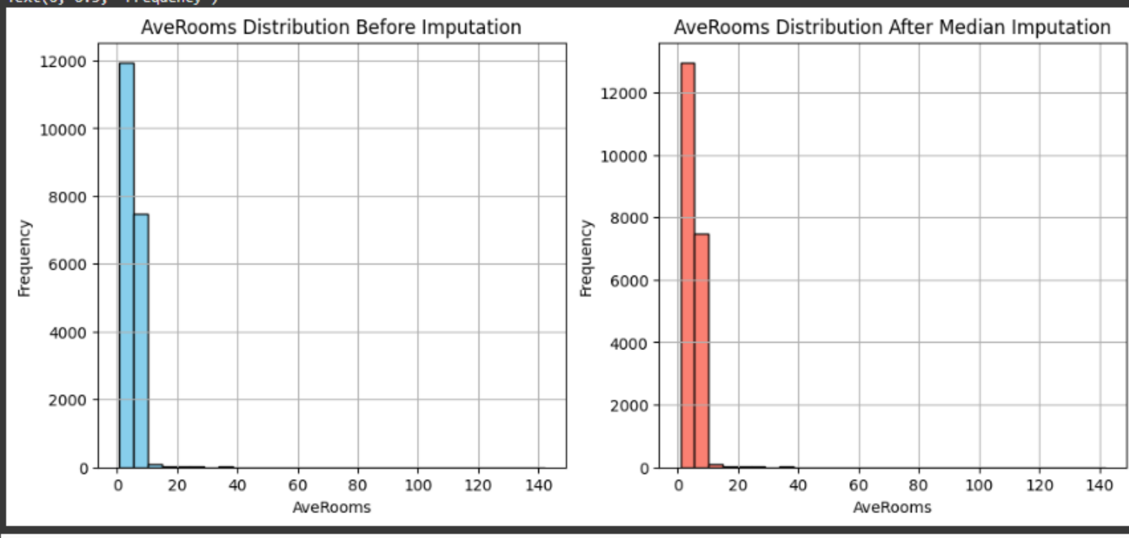
MedInc has positive correlation with MedHouseVal as it is closer to one and latititude has strongest negative correlation with MedHouseVal. Income affects the value of house.

[22]
```python
#2)Handling Missing Data:
```

```
[22] #2)Handling Missing Data:

     #Randomly set 5% of AveRooms values to NaN (simulate missingness).

     # Impute missing values using median imputation.

     #Visualize and compare distributions of AveRooms before and after imputation using histograms or boxplots.

     # Discuss the effect of imputation on data distribution

     #Randomly set 5% of AVeRooms values to NaN
     np.random.seed(42) #Reproducibility
     missing_val=np.random.rand(len(df))<0.05
     df.loc[missing_val,'AveRooms']=np.nan
     #Save a copy of original data for comparidion
     ave_rooms_original = df['AveRooms'].copy()
     #Impute missing values with median
     median_val=df['AveRooms'].median()
     df['AveRooms'].fillna(median_val,inplace=True)
     #Visaulize and Compare distributions of AveRooms before and after imputation
     plt.figure(figsize=(12,5))
     # Histogram before imputation (excluding NaNs)
     plt.subplot(1, 2, 1)
     ave_rooms_original.dropna().hist(bins=30, color='skyblue', edgecolor='black')
     plt.title('AveRooms Distribution Before Imputation')
     plt.xlabel('AveRooms')
     plt.ylabel('Frequency')
     # Histogram after imputation
     plt.subplot(1, 2, 2)
     df['AveRooms'].hist(bins=30, color='salmon', edgecolor='black')
     plt.title('AveRooms Distribution After Median Imputation')
     plt.xlabel('AveRooms')
     plt.ylabel('Frequency')
```

```
/tmp/ipython-input-22-4035776825.py:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting va

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].


  df['AveRooms'].fillna(median_val,inplace=True)
Text(0, 0.5, 'Frequency')
```

```
/tmp/ipython-input-22-4035776825.py:19: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained
The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting va

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].


  df['AveRooms'].fillna(median_val,inplace=True)
Text(0, 0.5, 'Frequency')
```



4: Exercises on Numpy

problem-1: array creation

```
[23] #1 Create a 1D NumPy array containing integers from 0 to 19.
     array=np.arange(20)
     print(array)
```

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

[24]
```
#2 Reshape it into a 4x5 matrix.
reshape_arr=array.reshape(4,5)
print(reshape_arr)
```

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

[25]
```
#3 Generate a 5x5 identity matrix and a 3x3 matrix filled with 7.
#Identity matrix of 5*5
identity_matrix=np.eye(5)
#3*3 Matrix filled with 7
matrix_7=np.full((3,3),7)
print("Identity Matrix:\n")
print(identity_matrix)
print("\n3*3 matrixfilled with 7:\n")
print(matrix_7)
```

```
Identity Matrix:

[[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.]]

3*3 matrixfilled with 7:

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

Problem-2: Basic operation

```
#1 Create two 3x3 matrices A and B with random integers (0-9).

matrix_A=np.random.randint(0,10,size=(3,3))
matrix_B=np.random.randint(0,10,size=(3,3))

print("Matrix A:\n")
print(matrix_A)
print("\nMatrix B:\n")
print(matrix_B)
```

```
Matrix A:
```

```
Matrix A:

[[9 5 8]
 [7 9 2]
 [5 6 5]]

Matrix B:

[[2 8 3]
 [1 8 2]
 [5 4 1]]
```

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

add_matrix=matrix_A+matrix_B
mul_matrix=matrix_A*matrix_B
div_matrix=matrix_A/matrix_B
matmul_result= matrix_A@matrix_B
#Display the result
print("Addition:\n")
print(add_matrix)
print("\nMultiplication:\n")
print(mul_matrix)
print("\nDivision:\n")
print(div_matrix)
print("\nMatrix Multiplication:\n")
print(matmul_result)
```

```
Addition:

[[11 13 11]
 [ 8 17  4]
 [10 10  6]]

Multiplication:

[[18 40 24]
 [ 7 72  4]
 [25 24  5]]

Division:

[[4.5        0.625      2.66666667]
 [7.         1.125      1.        ]
 [1.         1.5        5.        ]]

Matrix Multiplication:

[[ 63 144  45]
 [ 33 136  41]
 [ 41 108  32]]
```

```python
#3 Compute mean, median, standard deviation, and sum for each matrix.
#Compute mean , median, standard deviation of matrix A
mean_A=np.mean(matrix_A)
median_A=np.median(matrix_A)
std_A=np.std(matrix_A)
sum_A=np.sum(matrix_A)
#Display the result of matrix A
print("Mean of matrix A:",mean_A)
print("Median of matrix A:",median_A)
print("Standard Deviation of matrix A:",std_A)
print("Sum of matrix A:",sum_A)

#Compute mean , median, standard deviation of matrix B
mean_B=np.mean(matrix_B)
median_B=np.median(matrix_B)
std_B=np.std(matrix_B)
sum_B=np.sum(matrix_B)
#Display the result of matrix B
print("Mean of matrix B:",mean_B)
print("Median of matrix B:",median_B)
print("Standard Deviation of matrix B:",std_B)
print("Sum of matrix B:",sum_B)
```

```
Mean of matrix A: 6.222222222222222
Median of matrix A: 6.0
Standard Deviation of matrix A: 2.1487866228681907
Sum of matrix A: 56
Mean of matrix B: 3.7777777777777777
Median of matrix B: 3.0
Standard Deviation of matrix B: 2.57240820062005
Sum of matrix B: 34
```

problem-3: indexing and slicing

[30]
```python
#Slice the first two rows of matrix A
slice_A=matrix_A[:2]
print("First two rows of A:\n",slice_A)
```

```
First two rows of A:
 [[9 5 8]
 [7 9 2]]
```

[31]
```python
#Select element greater than 5
elem_greater_than5=matrix_A[matrix_A>5]
print("Elements greater than 5:\n",elem_greater_than5)
```

```
[31] #Select element greater than 5
     elem_greater_than5=matrix_A[matrix_A>5]
     print("Elements greater than 5:\n",elem_greater_than5)
```

```
Elements greater than 5:
 [9 8 7 9 6]
```

```
[32] #Replace all even numbers with -1
     new_matrix=matrix_A.copy()
     new_matrix[new_matrix%2==0]=-1
     print("Matrix A after replacing all even numbers with -1:\n",new_matrix)
```

```
Matrix A after replacing all even numbers with -1:
 [[ 9  5 -1]
 [ 7  9 -1]
 [ 5 -1  5]]
```

numpy advanced exercise

```
[33] #3*1 column vector
     col_vector=np.array([[1],[2],[3]])
     #1*4 row vector
     row_vector=np.array([4,5,6,7])
     #Display the vector
     print("3*1 column vector:\n",col_vector)
     print("1*4 row vector:\n",row_vector)
```

```
3*1 column vector:
 [[1]
 [2]
 [3]]
1*4 row vector:
 [4 5 6 7]
```

```
#Use broadcasting to generate a 3*4 multiplication table
multiplication_table=col_vector*row_vector
print("Multiplication Table:\n",multiplication_table)
```

```
Multiplication Table:
 [[ 4  5  6  7]
 [ 8 10 12 14]
 [12 15 18 21]]
```

loop vs vectorization

loop vs vectorization

```
[35] #Function using for loop
     def square_loop(arr):
         squared_arr=[]
         for num in arr:
             squared_arr.append(num**2)
         return np.array(squared_arr)

     #Function using Numpy vectorized operation
     def square_vectorized(arr):
         return np.square(arr)
```

```
[36] import time
     #Create a array
     arr=np.random.rand(1000)
     #Time for loop function
     start_time=time.time()
     result_loop=square_loop(arr)
     end_time=time.time()
     print(f"Time for loop function:{end_time - start_time:.6f} seconds")
     #Time for vectorized function
     start_time=time.time()
     result_loop=square_vectorized(arr)
     end_time=time.time()
     print(f"Time for vectorized function:{end_time - start_time:.6f}seconds")
```

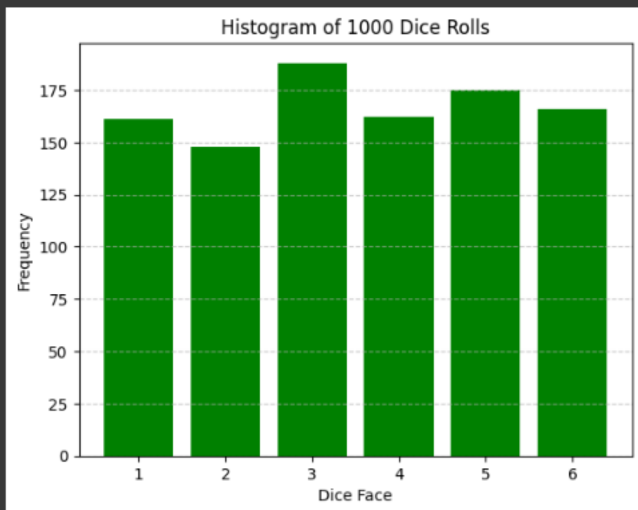
```
Time for loop function:0.000500 seconds
Time for vectorized function:0.000124seconds
```

simulation task

```
[37] # Simulate 1000 coin tosses:(True for heads, False for tail) using random numbers < 0.5 as heads (50% chance)
     tosses=np.random.random(size=1000)<0.5
     # Calculate proportion of heads
     proportion_heads=np.sum(tosses)/len(tosses)
     print(f"Proportion of heads in 1000 tosses:{proportion_heads:.3f}")
```

```
Proportion of heads in 1000 tosses:0.463
```

```
[39] # Simulate 1000 dice rolls (values from 1 to 6)
dice_rolls=np.random.randint(1,7,size=1000)
# Define bin edges centered on integers 1 through 6
bins=np.arange(1,8)-0.5
# Plot histogram
plt.hist(dice_rolls,bins=bins,color='green',rwidth=0.8)
plt.title('Histogram of 1000 Dice Rolls')
plt.xlabel('Dice Face')
plt.ylabel('Frequency')
plt.xticks(range(1, 7))
plt.grid(axis='y',linestyle='--',alpha=0.7)
plt.show()
```



solving system of equation

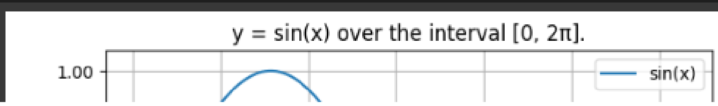
solving system of equation

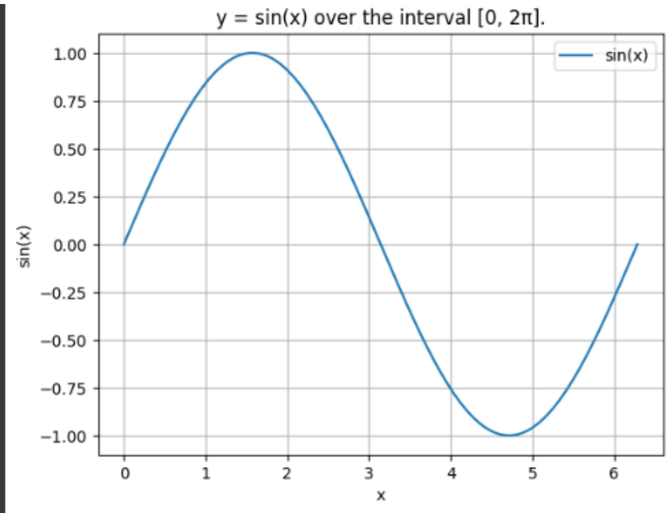
[40]
```python
# Coefficient matrix A
A=np.array([[3, 1],[1, 2]])
# Constants vector b
B=np.array([9, 8])
# Solve for x (which contains [x, y])
solution=np.linalg.solve(A,B)
print("Solution [x, y]:",solution)
```

```
Solution [x, y]: [2. 3.]
```

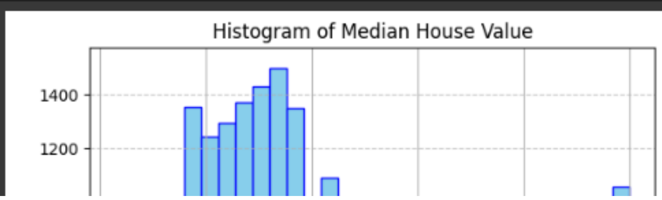
visualization with natplotib or seaborn

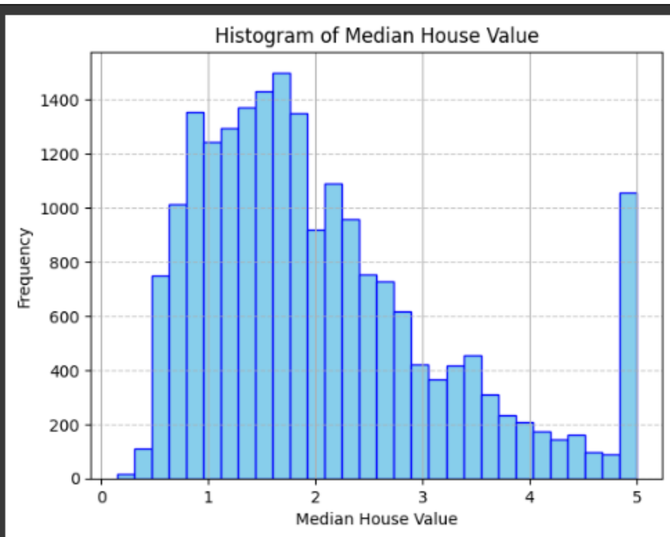
[41]
```python
#Generate x values from 0 to 2 pi
x=np.linspace(0,2*np.pi,500)
#Compute y=sinx
y=np.sin(x)
#Create a line plot
plt.plot(x,y,label='sin(x)')
#Customize the plot
plt.title('y = sin(x) over the interval [0, 2π].')
plt.xlabel('x')
plt.ylabel('sin(x)')
plt.grid(True)
plt.legend()
#Save the plot
plt.savefig('y=sin(x).png',dpi=300)
#Show the plot
plt.show()
```



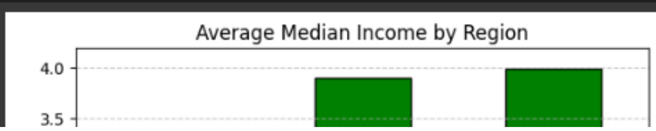


[42]
```
#Plot histogram of MedHouseVal
df['MedHouseVal'].hist(bins=30,edgecolor='blue',color='skyblue')
plt.title('Histogram of Median House Value')
plt.xlabel('Median House Value')
plt.ylabel('Frequency')
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

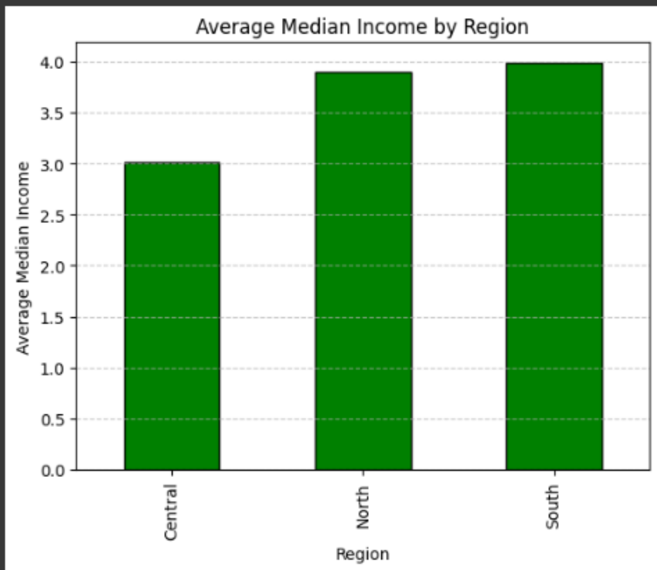




[44]
```
#Bar chart comparing average MedInc across Region
avg_medinc_by_region=df.groupby('Region')['MedInc'].mean()
avg_medinc_by_region.plot(kind='bar',color='green',edgecolor='black')
plt.title('Average Median Income by Region')
plt.xlabel('Region')
plt.ylabel('Average Median Income')
plt.grid(axis='y',linestyle='--',alpha=0.7)
plt.show()
```

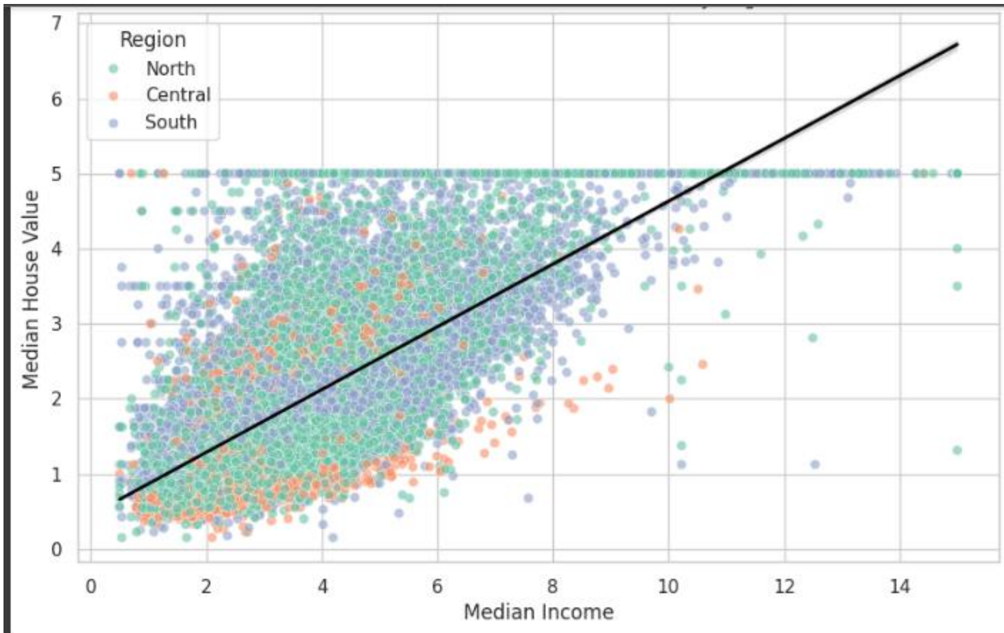


```
#Bar chart comparing average MedInc across Region
avg_medinc_by_region=df.groupby('Region')['MedInc'].mean()
avg_medinc_by_region.plot(kind='bar',color='green',edgecolor='black')
plt.title('Average Median Income by Region')
plt.xlabel('Region')
plt.ylabel('Average Median Income')
plt.grid(axis='y',linestyle='--',alpha=0.7)
plt.show()
```




```
[45] # Set plot style
sns.set(style="whitegrid")
#Create scatter plot with points colored by Region and transparency (alpha)
plt.figure(figsize=(10, 6))
scatter=sns.scatterplot(data=df,x='MedInc',y='MedHouseVal',hue='Region',alpha=0.6,palette='Set2')
#Add regression line (ignoring Region, overall trend)
sns.regplot(data=df,x='MedInc',y='MedHouseVal',scatter=False,ax=scatter.axes,color='black',line_kws={'linewidth': 2})
# Customize plot
plt.title('Median Income vs. Median House Value by Region')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.legend(title='Region')
plt.show()
```

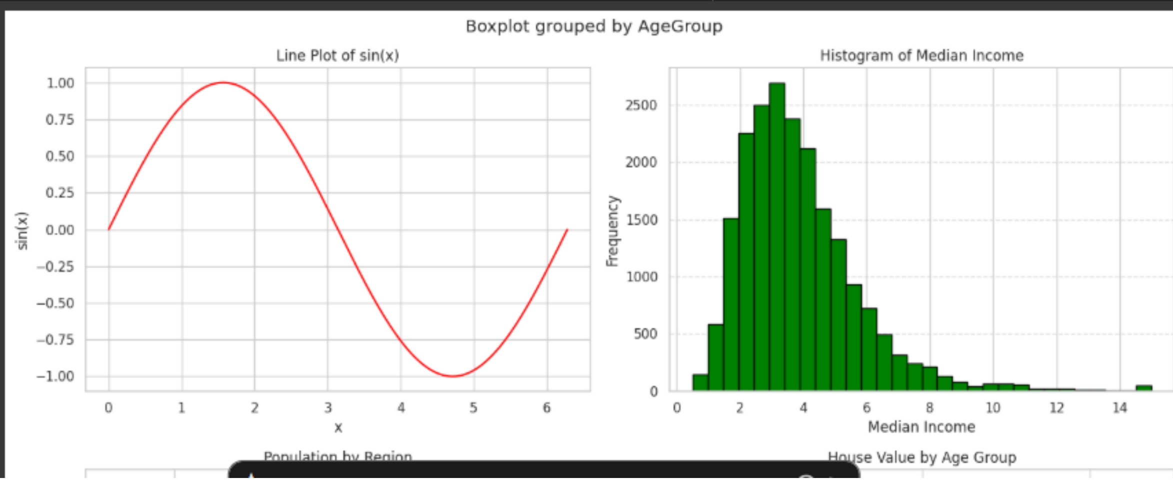
What can I help you build?

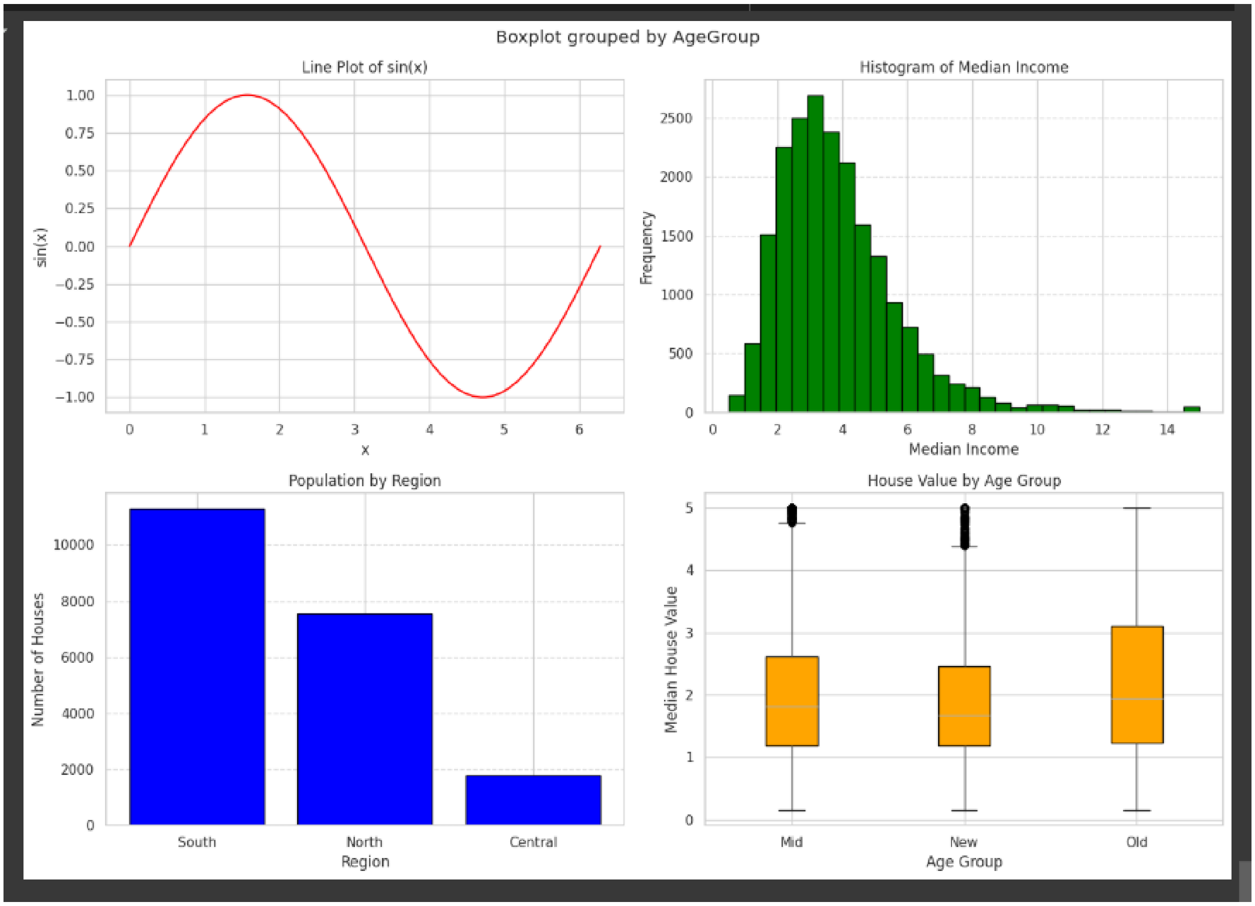


lots

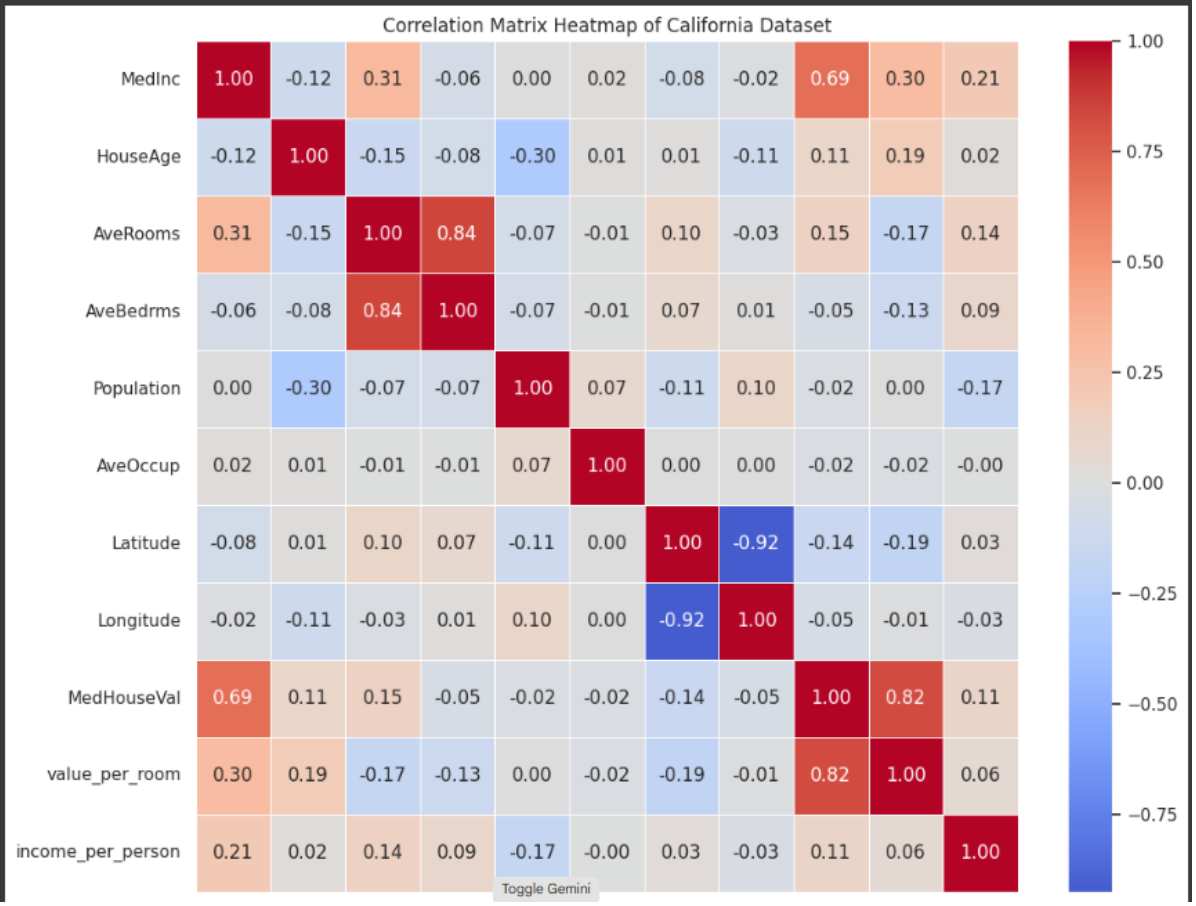
```
# Create figure and 2x2 grid of subplots
fig, axs = plt.subplots(2, 2, figsize=(14, 10))
fig.suptitle('California Housing Data Visualizations', fontsize=16)
#Line plot of sine
x=np.linspace(0,2*np.pi,500)
y=np.sin(x)
axs[0,0].plot(x,y,color='red')
axs[0,0].set_title('Line Plot of sin(x)')
axs[0,0].set_xlabel('x')
axs[0,0].set_ylabel('sin(x)')
axs[0,0].grid(True)
```

```python
#Histogram of income (MedInc)
axs[0,1].hist(df['MedInc'],bins=30,color='green',edgecolor='black')
axs[0,1].set_title('Histogram of Median Income')
axs[0,1].set_xlabel('Median Income')
axs[0,1].set_ylabel('Frequency')
axs[0,1].grid(axis='y',linestyle='--',alpha=0.7)

#Bar chart of region-wise population (count of houses per Region)
region_counts=df['Region'].value_counts()
axs[1,0].bar(region_counts.index, region_counts.values,color='blue',edgecolor='black')
axs[1,0].set_title('Population by Region')
axs[1,0].set_xlabel('Region')
axs[1,0].set_ylabel('Number of Houses')
axs[1,0].grid(axis='y', linestyle='--', alpha=0.7)

#Boxplot of house value grouped by age group
df.boxplot(column='MedHouseVal',by='AgeGroup',ax=axs[1, 1],grid=True,patch_artist=True,boxprops=dict(facecolor='orange'))
axs[1,1].set_title('House Value by Age Group')
axs[1,1].set_xlabel('Age Group')
axs[1,1].set_ylabel('Median House Value')

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



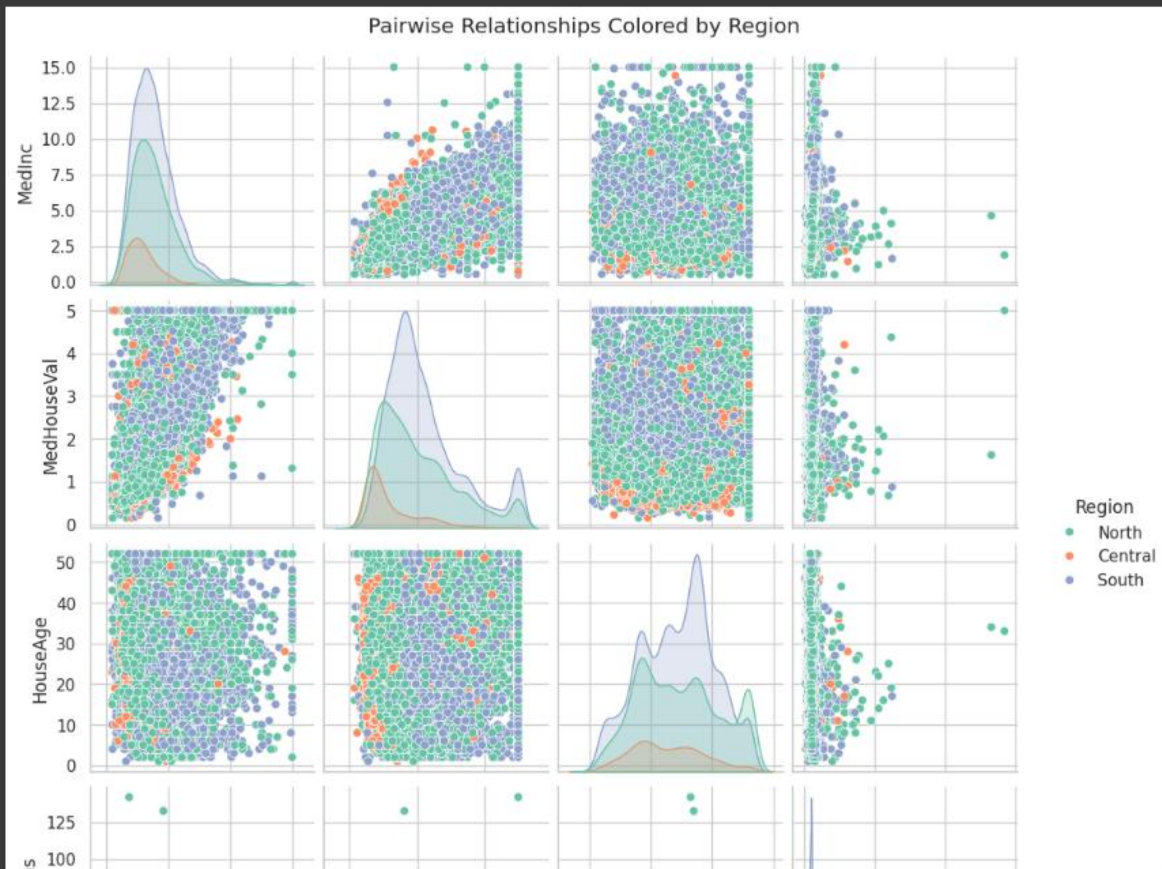

Boxplot grouped by AgeGroup
Line Plot of sin(x)
sin(x)
x
Histogram of Median Income
Frequency
Median Income
Population by Region
Number of Houses
South
North
Central
Region
House Value by Age Group
Median House Value
Mid
New
Old
Age Group

```python
#Compute the correlation matrix (Pearson by default)
corr_matrix=df.corr(numeric_only=True)

#Plot heatmap with annotations
plt.figure(figsize=(12, 10))
sns.heatmap(corr_matrix,annot=True,fmt=".2f",cmap="coolwarm",center=0,linewidths=0.5,linecolor='white')
plt.title("Correlation Matrix Heatmap of California Dataset")
plt.show()
```



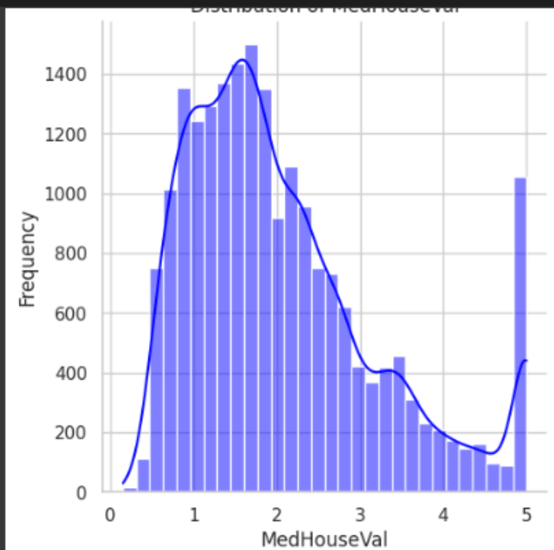
pairplot

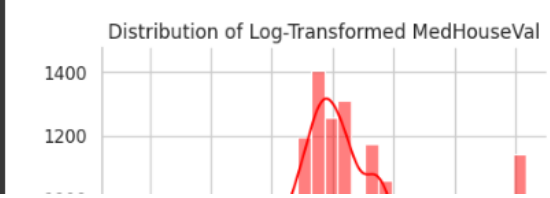
[48]
```
#Select columns of interest plus the categorical 'Region'
cols = ['MedInc','MedHouseVal','HouseAge','AveRooms','Region']

#Create the pairplot with hue='Region' to color points by region
sns.pairplot(df[cols],hue='Region',diag_kind='kde',palette='Set2')
plt.suptitle('Pairwise Relationships Colored by Region', y=1.02)
plt.show()
```

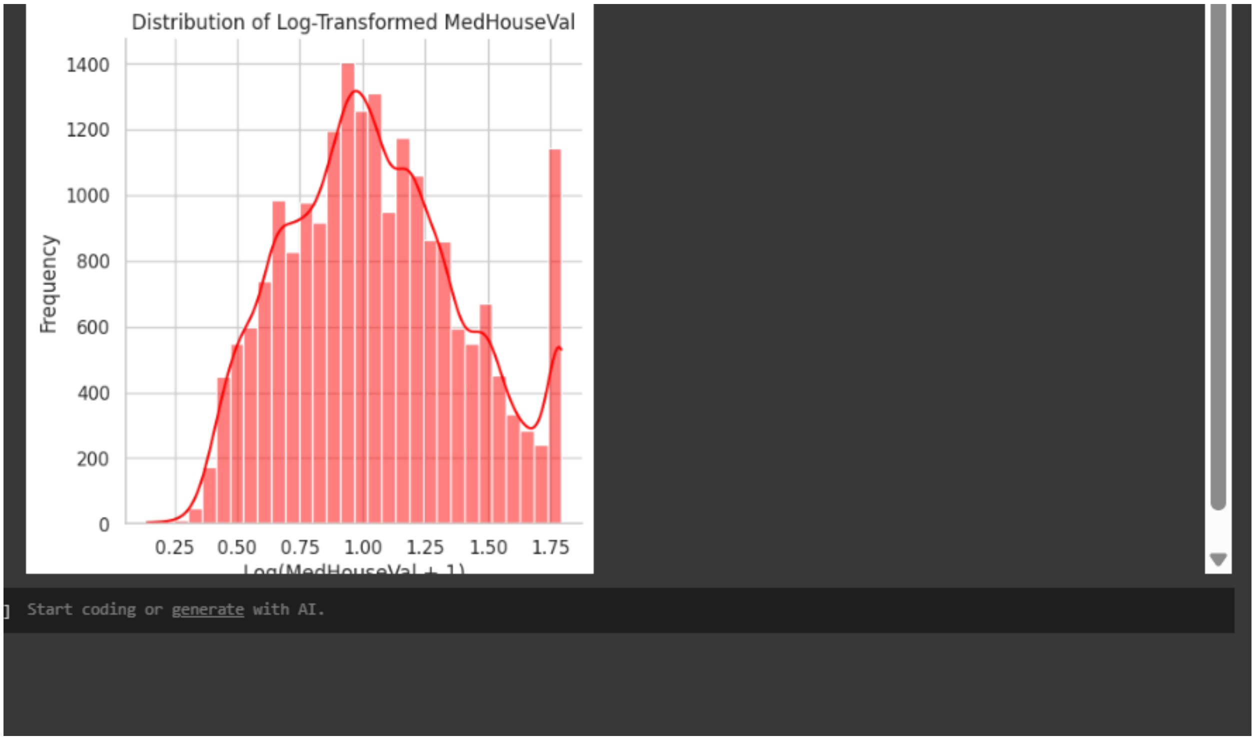


Variables Terminal ✓ 12:07 PM Python 3

```python
[49] # Original distribution of MedHouseVal
sns.displot(df['MedHouseVal'],kde=True,bins=30,color='blue')
plt.title('Distribution of MedHouseVal')
plt.xlabel('MedHouseVal')
plt.ylabel('Frequency')
plt.show()

# Log-transform MedHouseVal to reduce skewness
log_medhouseval = np.log1p(df['MedHouseVal'])
sns.displot(log_medhouseval,kde=True, bins=30,color='red')
plt.title('Distribution of Log-Transformed MedHouseVal')
plt.xlabel('Log(MedHouseVal + 1)')
plt.ylabel('Frequency')
plt.show()
```







Git hub repo link: https://github.com/AnanyaDahal/HCAI5DS02_AnanyaDahal