Load library needed in project

```
[1] import pandas as pd
    import numpy as np
    from google.colab import drive
    import seaborn as sns
    import matplotlib.pyplot as plt
    import geopandas as gpd
```

```
[2] drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

1. Load the housing.csv file using pandas

```
[3] data = pd.read_csv('drive/MyDrive/advanced_programming/housing.csv')
    data
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322	126	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401	1138	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496	177	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558	219	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565	259	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845	330	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356	114	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007	433	1.7000	NaN	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741	349	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387	530	2.3886	89400.0	INLAND

20640 rows × 10 columns

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

```
[4] data.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322	126	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401	1138	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496	177	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558	219	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565	259	3.8462	342200.0	NEAR BAY

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

```
[5] data.info() # show info about data --  we notice some colunms has null values like housing_median_age ,median_house_value
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20640 non-null  float64
 1   latitude            20640 non-null  float64
 2   housing_median_age  20631 non-null  float64
 3   total_rooms         20637 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64
 6   households          20640 non-null  int64
 7   median_income       20640 non-null  float64
 8   median_house_value  20633 non-null  float64
 9   ocean_proximity     20640 non-null  object
dtypes: float64(7), int64(2), object(1)
memory usage: 1.6+ MB
```

```
[6] data.describe() # show statistical data about numerics features
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20631.000000	20637.000000	20433.000000	20640.000000	20640.000000	20640.000000	20633.000000
mean	-119.569704	35.631861	28.638069	2635.761448	537.870553	1425.476744	499.539680	3.870671	206853.926283
std	2.003532	2.135952	12.584813	2181.767746	421.385070	1132.462122	382.329753	1.899822	115403.000793
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.000000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264700.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

2. Drop all rows with at least one missing value.

```
[7] data.isna().sum() # number if null values in each at
```

	0
longitude	0
latitude	0
housing_median_age	9
total_rooms	3
total_bedrooms	207
population	0
households	0
median_income	0
median_house_value	7
ocean_proximity	0

dtype: int64

```
[8] data.isna().any(axis=1).sum() # number of rows has null values to delete it
```

```
226
```

```
[9] # Drop rows with any missing values and assigne to new datafram
    housing = data.dropna()
```

```
[10] housing.info() # show info about cleaned data from null values
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 20414 entries, 0 to 20639
Data columns (total 10 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20414 non-null  float64
 1   latitude            20414 non-null  float64
 2   housing_median_age  20414 non-null  float64
 3   total_rooms         20414 non-null  float64
 4   total_bedrooms      20414 non-null  float64
 5   population          20414 non-null  int64
 6   households          20414 non-null  int64
 7   median_income       20414 non-null  float64
 8   median_house_value  20414 non-null  float64
 9   ocean_proximity     20414 non-null  object
dtypes: float64(7), int64(2), object(1)
memory usage: 1.7+ MB
```

3. Add The additional feature “rooms_per_household”

(Hint: rooms_per_household = total_rooms/households)

```
[12] # add rooms_per_household attribute we can use also apply() and pass method or lambda expression .. or direct excute like i do
     housing['rooms_per_household']  = housing['total_rooms'] / housing['households']
     housing[['total_rooms','households','rooms_per_household']].head(3)
```

	total_rooms	households	rooms_per_household
0	880.0	126	6.984127
1	7099.0	1138	6.238137
2	1467.0	177	8.288136

More Accurate: round value of rooms_per_houseload "الأسرة الوحدة الها عدد صحيح من الغرف"

```
[13] # make the values of feature rooms_per_household "integer" each hooushlod has own room
     housing['rooms_per_household'] = (housing['total_rooms'] / housing['households']).round().astype(int)

     # Display the updated DataFrame
     housing[['total_rooms', 'households', 'rooms_per_household']].head(3)
```

	total_rooms	households	rooms_per_household
0	880.0	126	7
1	7099.0	1138	6
2	1467.0	177	8

4. Add the additional feature “population_per_household”

(Hint: population_per_household =population/households)

More Accurate: round value of population_per_household "سكان الأسرة الوحدة عدد صحيح"

```
[14] # Calculate population_per_household
     housing['population_per_household'] = (housing['population'] / housing['households']).round().astype(int)
     housing[['population','households','population_per_household']].head(3)
```

	population	households	population_per_household
0	322	126	3
1	2401	1138	2
2	496	177	3

5. Calculate the average house value for houses that are 50 years old or less

```
[15] # 1. get houses that are 50 years old or less and store in new df
     houses_age_less_50 = housing[housing['housing_median_age'] <= 50]
     houses_age_less_50.head(10)['housing_median_age']
```

	housing_median_age
0	41.0
1	21.0
8	42.0
15	50.0
18	50.0
21	42.0
25	41.0
26	49.0
28	50.0
30	49.0

dtype: float64

```
[16] #2 . calculate the average house value
     meen_house_value = houses_age_less_50['median_house_value'].mean()
     print(f"Average house value for houses 50 years old or less: {meen_house_value:.4f}")
```

```
Average house value for houses 50 years old or less: 202288.8914
```

```
[17] # direct excution
     meen_house_value = housing[housing['housing_median_age'] <= 50]['median_house_value'].mean()
     print(f"Average house value for houses 50 years old or less: {meen_house_value:.4f}")
```

```
Average house value for houses 50 years old or less: 202288.8914
```

6. Calculate the correlation between the “median_house_value” and all features. Which factors seem to influence house prices?

Way #1

```
[18] # correlation for median_house_value and sort by correlation to get sorted factors influence in house prices
     median_house_value_corr = data.select_dtypes(include=['float64', 'int64']).corr()['median_house_value'].sort_values(ascending=False)
     median_house_value_corr
```

```
median_house_value
```

median_house_value	1.000000
median_income	0.688105
total_rooms	0.134114
housing_median_age	0.105529
households	0.065819
total_bedrooms	0.049660
population	-0.024725
longitude	-0.045907
latitude	-0.144183

dtype: float64

Way #2

```
[19] correlation = housing.loc[:, housing.dtypes != 'object'].corr() # just numerics value
     correlation['median_house_value'].sort_values(ascending=False)
```

	median_house_value
median_house_value	1.000000
median_income	0.688308
rooms_per_household	0.151165
total_rooms	0.133148
housing_median_age	0.106439
households	0.064785
total_bedrooms	0.049581
population_per_household	-0.023182
population	-0.025416
longitude	-0.045463
latitude	-0.144550

dtype: float64

```
[20] # OR specify numeric fetures only
     numeric_features_housing = housing.select_dtypes(include=['float64', 'int64']) # to avoid not convert string to float error in corr computing
     # Calculate correlation
     correlation = numeric_features_housing.corr()
     correlation['median_house_value']
```

	median_house_value
longitude	-0.045463
latitude	-0.144550
housing_median_age	0.106439
total_rooms	0.133148
total_bedrooms	0.049581
population	-0.025416
households	0.064785
median_income	0.688308
median_house_value	1.000000
rooms_per_household	0.151165
population_per_household	-0.023182

dtype: float64

Factors influence in house prices "median_house_value"

- Median Income: 0.688308
- Rooms Per Household: 0.151237
- Total Rooms: 0.133148
- housing_median_age: 0.106439

Result Explain

way #3

addition: draw heat map

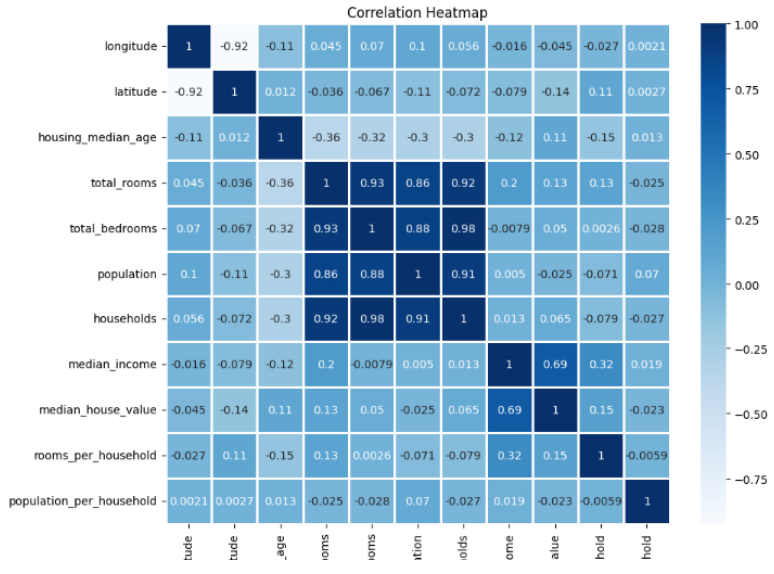
---

we notice most bold in :

1) median_income: 0.68 then 2) rooms_per_household 0.15 then 3) total_rooms: 0.13 then 4) housing_median_age: 0.10

```
[21] # Draw heatmap
     plt.figure(figsize=(10, 8))
     sns.heatmap(correlation, annot=True, cmap='Blues', linewidths=0.8)
     plt.title('Correlation Heatmap')
```

```
Text(0.5, 1.0, 'Correlation Heatmap')
```

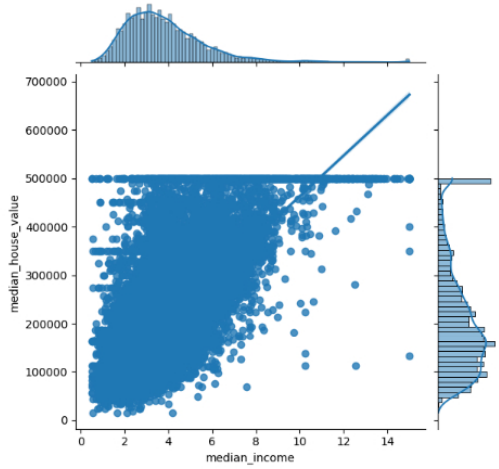


longi | lati | housing_median_ | total_rc | total_bedrc | popula | househ | median_inc | median_house_v | rooms_per_house | population_per_house

7. Create a Seaborn regression plot (jointplot) with income in the a-axis and house value on the y-axis

```
[22] # jointplot
     sns.jointplot(x= 'median_income', y= 'median_house_value', data= housing , kind='reg')
```

```
<seaborn.axisgrid.JointGrid at 0x7c540c4d3c10>
```



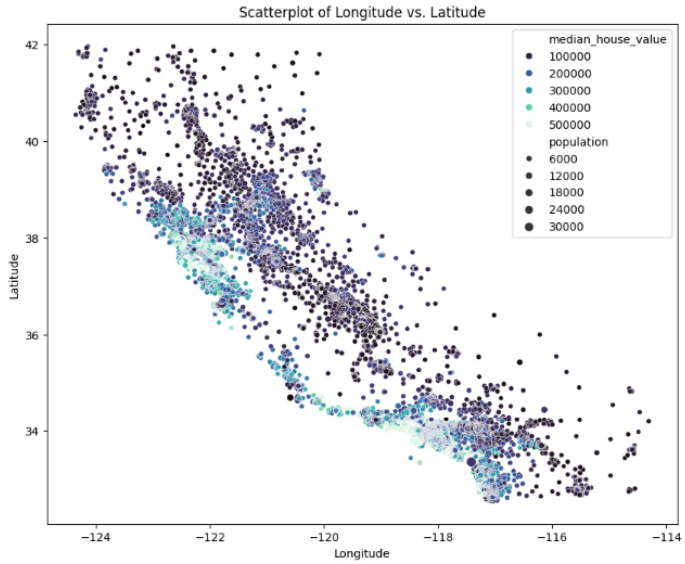
8. Create the following scatterplot:

- Longitude on x-axis
- Latitude on y-axis
- Size of datapoints is determined by population
- Color of datapoints is determined by median_house_value

1

```
[23] # Using Seberon
     plt.figure(figsize=(10, 8))
     scatter = sns.scatterplot(x='longitude', y='latitude', data=housing,
                               size='population',  # size of datapoints determined by population
                               hue='median_house_value',  # color of datapoints determined by median_house_value uses the hue to map the median_house_value column to a color scale
                               palette='mako',
                               )
     plt.xlabel('Longitude')
     plt.ylabel('Latitude')
     plt.title('Scatterplot of Longitude vs. Latitude')
```

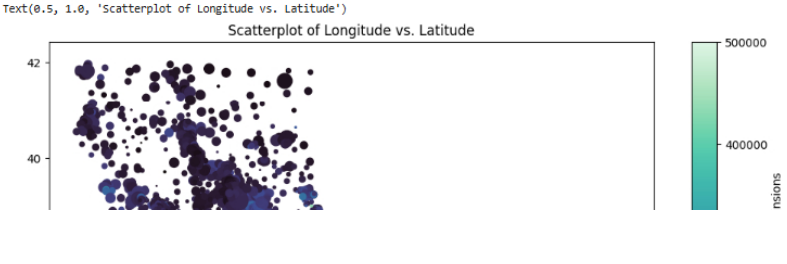
```
Text(0.5, 1.0, 'Scatterplot of Longitude vs. Latitude')
```

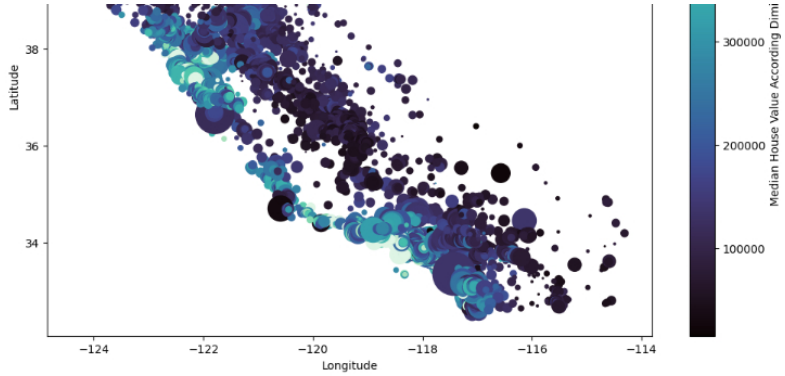


2

```
[24] # Using plt
     plt.figure(figsize=(12, 8))
     plt.scatter(x='longitude', y='latitude',
                 s=housing['population']/25,  # size of datapoints determined by population / 25 "more clear" s mean The marker size in points**2
                 c='median_house_value',  # color of datapoints determined by median_house_value
                 data= housing,
                 cmap='mako')
     plt.xlabel('Longitude')
     plt.ylabel('Latitude')
     plt.colorbar(label='Median House Value According Diminsions')
     plt.title('Scatterplot of Longitude vs. Latitude')
```

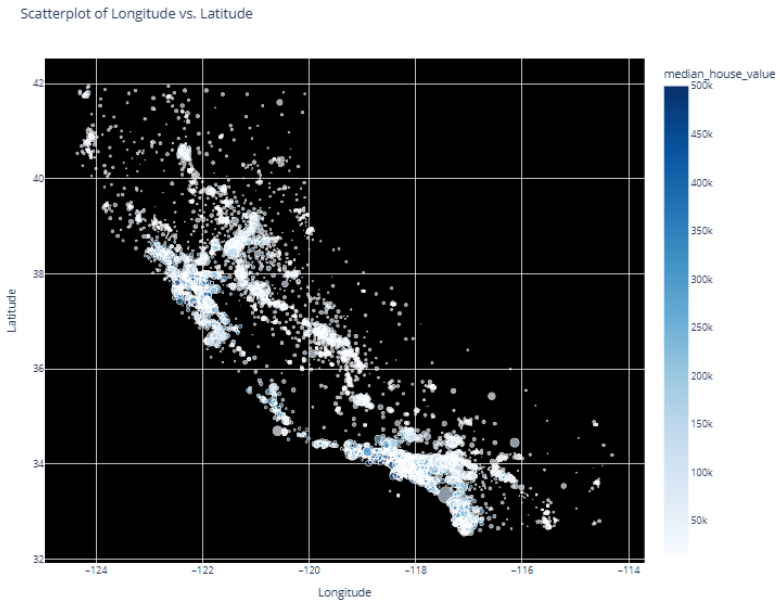
```
Text(0.5, 1.0, 'Scatterplot of Longitude vs. Latitude')
```





3

```
[25] import plotly.express as px

     # using plotly "intractive visualization"
     fig = px.scatter(
         housing, x='longitude', y='latitude',
         size='population',  # size of datapoints determined by population
         color='median_house_value',  # color of datapoints determined by median_house_value
         title='Scatterplot of Longitude vs. Latitude',
         color_continuous_scale='blues',
     )

     fig.update_layout(
         xaxis_title='Longitude',
         yaxis_title='Latitude',
         plot_bgcolor='black',
         width=1000,
         height=800
     )
```



9. Use geo_pandas to draw the previous scatter plot on California map.

```
# !pip install geopandas==0.12.2 fiona==1.9.3  # take versions from cousred vedio lect 6 part 2
# gpd.show_versions()
# gpd.datasets.available ['naturalearth_lowres', 'naturalearth_cities', 'nybb']
# world = gpd.read_file(gpd.datasets.get_path('naturalearth_lowres'))
# world
```

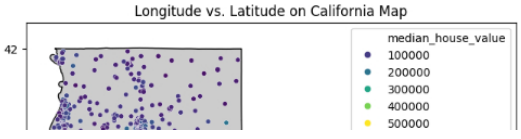
```
[27] points = gpd.points_from_xy(housing['longitude'], housing['latitude'])
     gdf = gpd.GeoDataFrame(housing, geometry=points)
     # gdf.head(3)
```

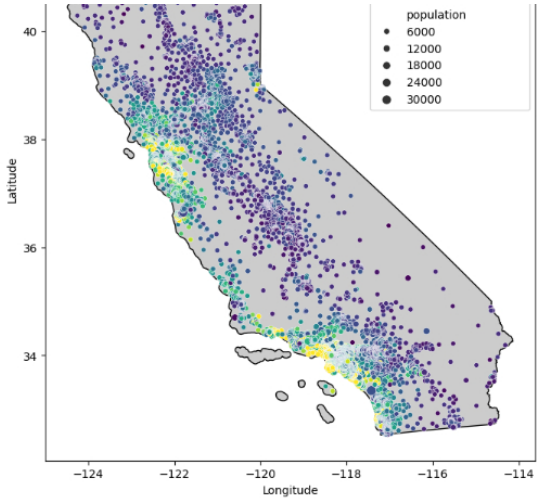
```
[28] usa = gpd.read_file('drive/MyDrive/advanced_programming/maps/tl_2022_us_state.shp')
     usa.to_crs(epsg='3857')[usa['NAME'] == 'California'] # confirm it exsist
```

	REGION	DIVISION	STATEFP	STATENS	GEOID	STUSPS	NAME	LSAD	MTFCC	FUNCSTAT	ALAND	AWATER	INTPTLAT	INTPTLON	geometry
13	4	9	06	01779778	06	CA	California	00	G4000	A	403673617862	20291712025	+37.1551773	-119.5434183	MULTIPOLYGON (((-13358323.978 5039515.776, -13...

```
[29] # put map and scatterplot
     plt.figure(figsize=(12, 9))

     # 1 plot California map
     usa[usa['NAME'] == 'California'].plot(figsize=(12, 9) , edgecolor='black', color='0.8')

     # 2 plot scatterplot
     sns.scatterplot(x='longitude', y='latitude', data=housing, size='population', # size of datapoints determined by population
                     hue='median_house_value', # color of datapoints determined by median_house_value
                     palette='viridis',
                    )

     plt.xlabel('Longitude')
     plt.ylabel('Latitude')
     plt.title('Longitude vs. Latitude on California Map')
```

```
Text(0.5, 1.0, 'Longitude vs. Latitude on California Map')
<Figure size 1200x900 with 0 Axes>
```





10.Add the additional column “income_cat” with the following income categories:

- Lowest 25% -> “Low”
- 25th to 50th percentile -> “Below average”
- 50th to 75th percentile -> “Above average”
- 75th to 95Th percentile -> “High”
- Above 95th percentile -> “Very high”

1

```
[30] # make fun
     def categorize_income(median_income):
         if median_income <= housing['median_income'].quantile(0.25):
             return 'Low'
         elif median_income <= housing['median_income'].quantile(0.50):
             return 'Below_Average'
         elif median_income <= housing['median_income'].quantile(0.75):
             return 'Above_Average'
         elif median_income <= housing['median_income'].quantile(0.95):
             return 'High'
         else:
             return 'Very_High'
```

```
[31] housing['median_income'].describe() # show quantile data about median_income features
```

	median_income
count	20414.000000
mean	3.871645
std	1.899395
min	0.499900
25%	2.564075
50%	3.537700
75%	4.743925
max	15.000100

dtype: float64

```
[32] housing['income_cat'] = housing['median_income'].apply(categorize_income)
     housing[['median_income','income_cat']].head(10)
```

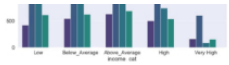
	median_income	income_cat
0	8.3252	Very_High
1	8.3014	Very_High
2	7.2574	High
3	5.6431	High
4	3.8462	Above_Average
5	4.0368	Above_Average
6	3.6591	Above_Average
7	3.1200	Below_Average
8	2.0804	Low
9	3.6912	Above_Average

2

```
[33] # or direct make bins
     housing['income_cat'] = pd.cut(housing['median_income'],
         bins=[0,housing['median_income'].quantile(0.25), housing['median_income'].quantile(0.5),housing['median_income'].quantile(0.75),housing['median_income'].quantile(0.95),housing['median_income'].max()],
         labels=['Low', 'Below_Average', 'Above_Average', 'High', 'Very_High'], # [0,25] low , [25,50] Below_Average , [50,75] Above_Average , [75,95] High ,  [95 - max] Very_High
     )
     housing[['median_income','income_cat']].head(10)
```

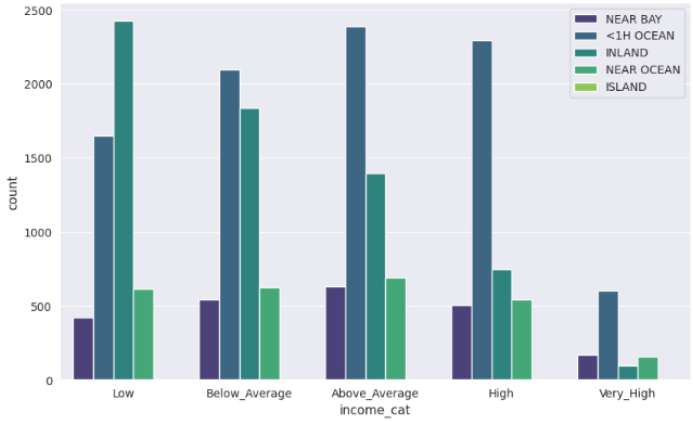
	median_income	income_cat
0	8.3252	Very_High
1	8.3014	Very_High
2	7.2574	High
3	5.6431	High
4	3.8462	Above_Average
5	4.0368	Above_Average
6	3.6591	Above_Average
7	3.1200	Below_Average
8	2.0804	Low
9	3.6912	Above_Average

11. Create the following plot using seaborn

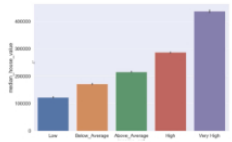


```
plt.figure(figsize=(10, 6))
sns.set_style("darkgrid")
sns.countplot(x='income_cat', hue='ocean_proximity', data=housing, palette='viridis')
plt.xlabel('income_cat', fontsize=11)
plt.ylabel('count', fontsize=11)
plt.legend(title=False)
```

```
<matplotlib.legend.Legend at 0x7c53bbd3bcd0>
```

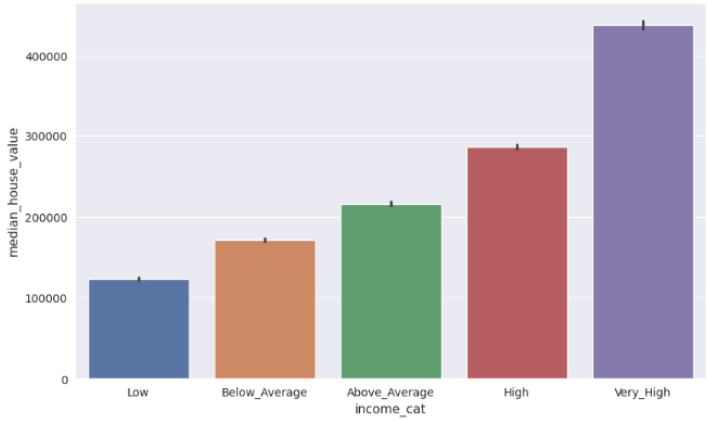


12. Create the following plot using Seaborn

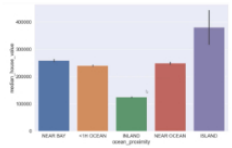


```
[35] # create same privious plot
plt.figure(figsize=(10, 6))
sns.set_style("darkgrid")
sns.barplot(x='income_cat', y='median_house_value', data=housing, palette='deep', hue='income_cat')
plt.xlabel('income_cat', fontsize=11)
plt.ylabel('median_house_value', fontsize=11)
```

```
Text(0, 0.5, 'median_house_value')
```

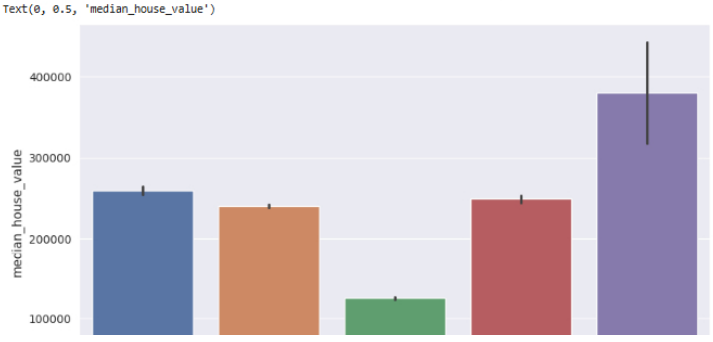


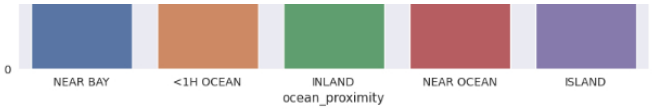
13.Create the following plot using Seaborn



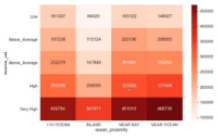
```
[36] # create same privious plot
plt.figure(figsize=(10, 6))
sns.set_style("darkgrid")
sns.barplot(x='ocean_proximity', y='median_house_value', data=housing, palette='deep', hue='ocean_proximity')
plt.xlabel('ocean_proximity', fontsize=11)
plt.ylabel('median_house_value', fontsize=11)
```

```
Text(0, 0.5, 'median_house_value')
```





14.Create the following Seaborn Heatmap with mean house values for all combinations of income_cat and ocean_proximity



```
[37] # create same privious plot
plt.figure(figsize=(10, 6))
mean_combination_values = housing.groupby(['income_cat', 'ocean_proximity'])['median_house_value'].mean() # Output : Low  <1H OCEAN 161337.0769 [in form: income_cat ocean_proximity median_house_value mean]
mean_combination_values = mean_combination_values.unstack() # to make fit for map
mean_combination_values
```

```
<ipython-input-37-05e30e2492c3>:3: FutureWarning:

The default of observed=False is deprecated and will be changed to True in a future version of pandas. Pass observed=False to retain current behavior or observed=True to adopt the future default and silence this warning.
```

ocean_proximity	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
income_cat					
Low	161337.076923	84874.865456	450000.0	155425.612440	148027.826514
Below_Average	197238.100716	115091.935695	363050.0	219966.322284	208665.190096
Above_Average	232278.358759	147897.778495	NaN	262020.855556	255293.813584
High	292190.904969	208014.927419	NaN	322351.578218	337013.264815
Very_High	439784.235489	347571.736842	NaN	451015.078788	470915.291139

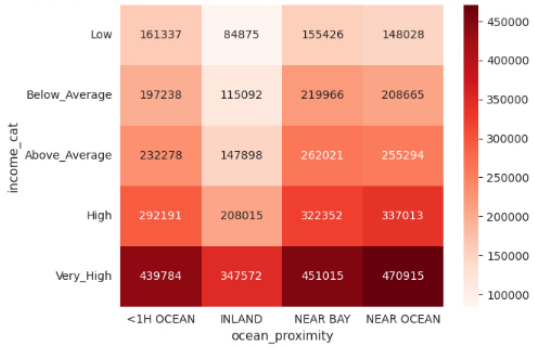
```
<Figure size 1000x600 with 0 Axes>
```

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

```
[38] mean_combination_values = mean_combination_values.drop(columns='ISLAND')
```

```
[39] sns.heatmap(mean_combination_values, annot=True, cmap='Reds', fmt='.0f')
plt.xlabel('ocean_proximity', fontsize=11)
plt.ylabel('income_cat', fontsize=11)
```

```
Text(50.58159722222221, 0.5, 'income_cat')
```



15.Reload the housing.csv file again, and handle missing data as the following:

```
[40] housing = pd.read_csv('drive/MyDrive/advanced_programming/housing.csv')
housing
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322	126	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401	1138	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496	177	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558	219	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565	259	3.8462	342200.0	NEAR BAY
...	...	...	...	...	...	...	...	...	...	...
20635	-121.09	39.48	25.0	1665.0	374.0	845	330	1.5603	78100.0	INLAND
20636	-121.21	39.49	18.0	697.0	150.0	356	114	2.5568	77100.0	INLAND
20637	-121.22	39.43	17.0	2254.0	485.0	1007	433	1.7000	NaN	INLAND
20638	-121.32	39.43	18.0	1860.0	409.0	741	349	1.8672	84700.0	INLAND
20639	-121.24	39.37	16.0	2785.0	616.0	1387	530	2.3886	89400.0	INLAND

20640 rows × 10 columns

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

- Missing values in the total_bedrooms column should be filled with the mean value.

```
[41] housing.isna().sum() # number if null values in total_bedrooms att
```

	0
longitude	0
latitude	0
housing_median_age	9
total_rooms	3
total_bedrooms	207
population	0
households	0
median_income	0
median_house_value	7
ocean_proximity	0

dtype: int64

```
[42] housing['total_bedrooms'].isna().sum() # number of null values in total_bedrooms att
```

```
207
```

```
[43] from sklearn.impute import SimpleImputer
     # housing['total_bedrooms'] = housing['total_bedrooms'].fillna(housing['total_bedrooms'].mean()) # filled with the mean value.

     # OR
     imputer_mean = SimpleImputer(strategy='mean')
     housing['total_bedrooms'] = imputer_mean.fit_transform(housing[['total_bedrooms']])

     housing['total_bedrooms'].isna().sum() # number of null values in total_bedrooms att after handle missing with mean
```

```
0
```

- Missing values in the total_rooms column should be filled with the median value.

```
[44] housing['total_rooms'].isna().sum() # number of null values in total_rooms att
```

```
3
```

```
# housing['total_rooms'] = housing['total_rooms'].fillna(housing['total_rooms'].median())# filled with the median value

# OR
imputer_median = SimpleImputer(strategy='median')
housing['total_rooms'] = imputer_median.fit_transform(housing[['total_rooms']])

housing['total_rooms'].isna().sum() # number of null values in total_rooms att after handle missing with median
```

```
0
```

- Missing values in the total_rooms column should be filled with the value calculated by KNN algorithm, with k=2.

```
[46] from sklearn.impute import KNNImputer

     KNN_imputer = KNNImputer(n_neighbors=2)
     housing[['total_rooms']] = KNN_imputer.fit_transform(housing[['total_rooms']])
```

try on another column

```
[47] housing['median_house_value'].isna().sum() # number of null values in median_house_value att
```

```
7
```

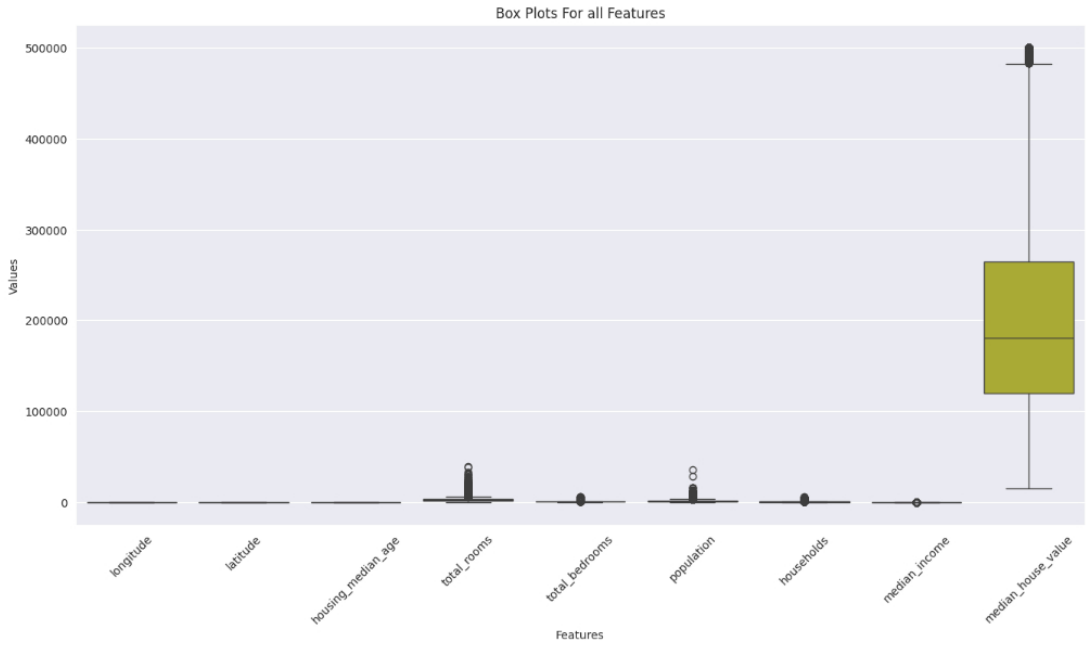
```
[48] from sklearn.impute import KNNImputer

     KNN_imputer = KNNImputer(n_neighbors=2)
     housing[['median_house_value']] = KNN_imputer.fit_transform(housing[['median_house_value']])
     housing['median_house_value'].isna().sum() # number of null values in median_house_value att after handle missing
```

```
0
```

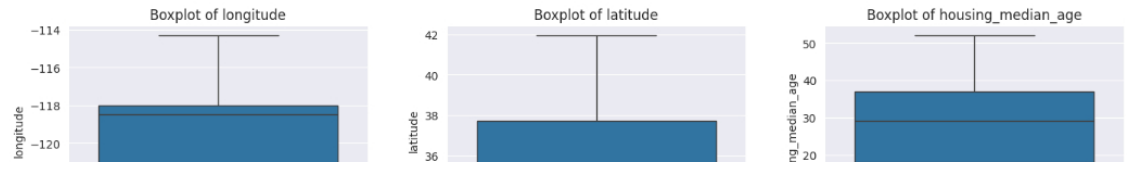
16.Identify columns that have outliers by using the boxplot of Seaborn or Matplotlib.

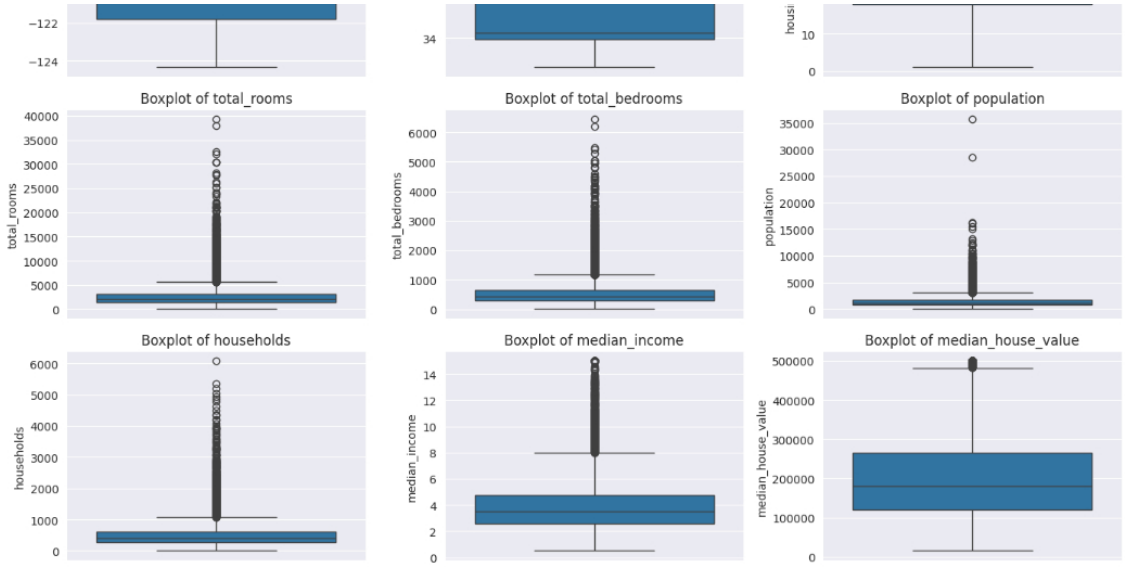
```
[49] # Create a single boxplot for all columns in the dataset
     plt.figure(figsize=(16, 8))
     sns.boxplot(housing)
     plt.title('Box Plots For all Features')
     plt.xticks(rotation=45)  # better readability
     plt.ylabel('Values')
     plt.xlabel('Features')
```

```
Text(0.5, 0, 'Features')
```




```
[50] plt.figure(figsize=(14, 12))

     # Boxplot for all columns in the dataset
     for i, column in enumerate(housing.columns[:-1], 1): # except last/object column
         plt.subplot(4, 3, i)  # display in 4 rows and 3 columns
         sns.boxplot(data=housing, y=column)
         plt.title(f'Boxplot of {column}')
     plt.tight_layout()  # spacing between subplots
```






```
[51] # function to handel outlier : drop / delete
     def remove_outliers(housing):
       for col in housing.columns:
         if housing[col].dtype != 'object':
           q1_value = housing[col].quantile(0.25)
           q3_value = housing[col].quantile(0.75)
           IQR = q3_value - q1_value
           max = q3_value + (1.5 * IQR)
           min = q1_value - (1.5 * IQR)
           housing = housing[(housing[col]>=min) & (housing[col]<=max)]
       return housing
```

```
[52] # other function to handel outlier : Clip
     def clip_outliers(housing):
       for col in housing.columns:
         if housing[col].dtype != 'object':
           q1_value = housing[col].quantile(0.25)
           q3_value = housing[col].quantile(0.75)
           IQR = q3_value - q1_value
           max = q3_value + (1.5 * IQR)
           min = q1_value - (1.5 * IQR)
           housing[col] = housing[col].clip(lower=min, upper=max)
       return housing
```

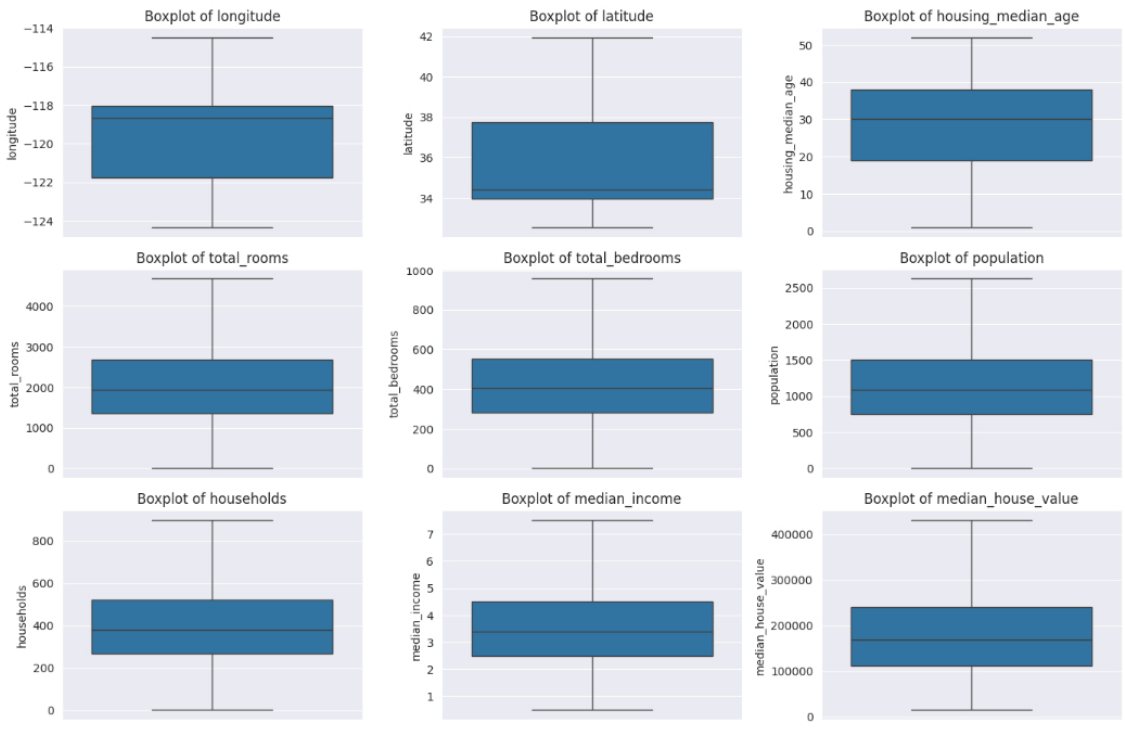
```
[53] print(len(housing))
     housing = remove_outliers(housing)
     print(len(housing))
```

```
20640
16886
```

```
[54] print(len(housing))
     cleand_housing = clip_outliers(housing)
     print(len(cleand_housing)) # just cap values not drop it
```

```
16886
16886
```

```
[55] plt.figure(figsize=(14, 12))

     # Boxplot for all columns in the dataset
     for i, column in enumerate(cleand_housing.columns[:-1], 1): # except last/object column
         plt.subplot(4, 3, i)  # display in 4 rows and 3 columns
         sns.boxplot(data=cleand_housing, y=column)
         plt.title(f'Boxplot of {column}')
     plt.tight_layout()  # spacing between subplots
```



18.We want to develop a model that should predict the median house value given the other

attributes. Separate the median_house_value so that it can be used as the labels columns. Other columns will be used as features for machine learning.

```
[56] x = cleand_housing.drop(columns=['median_house_value'])
     y = cleand_housing['median_house_value']
```

19.Split into train and test sets.

```
[57] from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)
```

```
[58] print("X_train:", X_train.shape)
     print("X_test:", X_test.shape)
     print("y_train:", y_train.shape)
     print("y_test:", y_test.shape)
```

```
X_train: (11820, 9)
X_test: (5066, 9)
y_train: (11820,)
y_test: (5066,)
```

20.Normalize numeric columns by using MinMaxScaler from sklearn.

```
[59] from sklearn.preprocessing import MinMaxScaler # import library

     scaler = MinMaxScaler() # Scaler


     numeric_columns = X_train.select_dtypes(include=['float64', 'int64']).columns #numeric columns
     # fit_transform training data
     X_train[numeric_columns] = scaler.fit_transform(X_train[numeric_columns])
     # transform testing data
     X_test[numeric_columns] = scaler.transform( X_test[numeric_columns])

     X_train.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
8479	0.611562	0.147715	0.647059	0.32680	0.282918	0.304563	0.317549	0.635029	<1H OCEAN
12446	0.950304	0.136026	0.274510	0.13677	0.132585	0.050951	0.055710	0.385691	INLAND

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

```
[60] X_test.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity
17137	0.222110	0.522848	0.607843	0.566894	0.687981	0.532319	0.814485	0.536136	NEAR BAY
16609	0.393509	0.340064	0.156863	0.908607	0.817434	0.784030	0.767688	0.459789	<1H OCEAN

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

```
[61] # save scaler to next un seen data
     import pickle
     pickle.dump(scaler, open('numeric_scaler','wb'))
```

21.Use OneHotEncoding to encode the ocean_proximity column

```
[62] from sklearn.preprocessing import OneHotEncoder

     encoder = OneHotEncoder(sparse_output=False)
     x_train_encoded = encoder.fit_transform(X_train[['ocean_proximity']])
     x_test_encoded = encoder.transform(X_test[['ocean_proximity']])
```

```
[63] encoded_columns = encoder.get_feature_names_out(['ocean_proximity'])
     encoded_columns
```

```
array(['ocean_proximity_<1H OCEAN', 'ocean_proximity_INLAND',
       'ocean_proximity_ISLAND', 'ocean_proximity_NEAR BAY',
       'ocean_proximity_NEAR OCEAN'], dtype=object)
```

```
[64] x_train_encoded
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 1.],
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.]])
```

```
[65] x_test_encoded
```

```
array([[0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       ...,
       [0., 1., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [1., 0., 0., 0., 0.]])
```

```
[66] # addition -  retrive encoded data as df
     ocean_proximity_train_df = pd.DataFrame(x_train_encoded, columns=encoded_columns, index=X_train.index)
     ocean_proximity_test_df = pd.DataFrame(x_test_encoded, columns=encoded_columns, index=X_test.index)

     # drop ocean_proximity column and add 5 encoded column each represent value
     X_train = X_train.drop('ocean_proximity', axis=1).join(ocean_proximity_train_df)
     X_test = X_test.drop('ocean_proximity', axis=1).join(ocean_proximity_test_df)
     X_train.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY	ocean_proximity_I O
8479	0.611562	0.147715	0.647059	0.32680	0.282918	0.304563	0.317549	0.635029	1.0	0.0	0.0	0.0	
12446	0.950304	0.136026	0.274510	0.13677	0.132585	0.050951	0.055710	0.385691	0.0	1.0	0.0	0.0	

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

```
[67] X_test.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	ocean_proximity_NEAR BAY	ocean_proximity_I O
17137	0.222110	0.522848	0.607843	0.566894	0.687981	0.532319	0.814485	0.536136	0.0	0.0	0.0	1.0	
16609	0.393509	0.340064	0.156863	0.908607	0.817434	0.784030	0.767688	0.459789	1.0	0.0	0.0	0.0	

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

```
[68] # save encoder to next un seen data
     pickle.dump(encoder, open('category_encoder','wb'))
```

22.Use ColumnTransformer from sklearn to apply the previous two transformation processes on the features.

```
[69] from sklearn.compose import ColumnTransformer

     # take new data to apply the previous two transformation processes
     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.3)

     # transformers
     scaler2 = MinMaxScaler()
     encoder2 = OneHotEncoder(sparse_output=False)

     # ColumnTransformer
     CT = ColumnTransformer([
         ('numeric_scale', scaler2, [0, 1, 2, 3, 4, 5, 6, 7]),  # MinMaxScaler to numeric columns
         ('categorical_encode', encoder2, [8])  # OneHotEncoder to ocean_proximity
     ])

     # fit and transform on train set
     t_X_train = CT.fit_transform(X_train)
     # Transform on test set
     t_X_test = CT.transform(X_test)

     t_X_train[0] ,  t_X_test[0]
```

indices of numeric column

```
(array([0.71298174, 0.20935175, 0.29411765, 0.65402642, 0.63682631,
        0.48669202, 0.67075209, 0.34060057, 0.        , 1.        ,
        0.        , 0.        , 0.        ]),
 array([0.24137931, 0.52072264, 0.31372549, 0.02662974, 0.02714342,
        0.08212928, 0.02228412, 0.57189013, 1.        , 0.        ,
        0.        , 0.        , 0.        ]))
```

```
[70] all_columns= ['longitude', 'latitude', 'housing_median_age', 'total_rooms',
                   'total_bedrooms', 'population', 'households', 'median_income',
                   'ocean_proximity_<1H OCEAN','ocean_proximity_INLAND' ,'ocean_proximity_ISLAND',' ocean_proximity_NEAR BAY','ocean_proximity_NEAR OCEAN']
     ct_df_train = pd.DataFrame(t_X_train, columns=all_columns)
     ct_df_test = pd.DataFrame(t_X_test, columns=all_columns)
     ct_df_train.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	\tocean_proximity_NEAR BAY	ocean_proximity_NEA OCEA
0	0.712982	0.209352	0.294118	0.654026	0.636826	0.486692	0.670752	0.340601	0.0	1.0	0.0	0.0	0
1	0.624746	0.247609	0.019608	0.447593	0.404019	0.349810	0.374373	0.648379	0.0	1.0	0.0	0.0	0

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

```
[71] ct_df_test.head(2)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity_<1H OCEAN	ocean_proximity_INLAND	ocean_proximity_ISLAND	\tocean_proximity_NEAR BAY	ocean_proximity_NEA OCEA
0	0.241379	0.520723	0.313725	0.026630	0.027143	0.082129	0.022284	0.57189	1.0	0.0	0.0	0.0	0
1	0.735294	0.012752	0.509804	0.379207	0.582539	0.677567	0.621727	0.23546	0.0	0.0	0.0	0.0	1

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

```
[72] pickle.dump(CT,open("coulmn_transformer","wb"))
```

23.Read the sarcasm.json file by using pandas

```
[73] sarcasm = pd.read_json('drive/MyDrive/advanced_programming/sarcasm.json')
     sarcasm
```

	article_link	headline	is_sarcastic
0	https://www.huffingtonpost.com/entry/versace-b...	former versace store clerk sues over secret 'b...	0
1	https://www.huffingtonpost.com/entry/roseanne-...	the 'roseanne' revival catches up to our thorn...	0
2	https://local.theonion.com/mom-starting-to-fea...	mom starting to fear son's web series closest ...	1
3	https://politics.theonion.com/boehner-just-wan...	boehner just wants wife to listen, not come up...	1
4	https://www.huffingtonpost.com/entry/jk-rowlin...	j.k. rowling wishes snape happy birthday in th...	0
...	...	...	...
26704	https://www.huffingtonpost.com/entry/american-...	american politics in moral free-fall	0
26705	https://www.huffingtonpost.com/entry/americas-...	america's best 20 hikes	0
26706	https://www.huffingtonpost.com/entry/reparatio...	reparations and obama	0
26707	https://www.huffingtonpost.com/entry/israeli-b...	israeli ban targeting boycott supporters raise...	0
26708	https://www.huffingtonpost.com/entry/gourmet-g...	gourmet gifts for the foodie 2014	0

26709 rows × 3 columns

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

24.Add the additional column "domain_name". Extract the domain name from the article link

For example, if the article link is:

https://www.huffingtonpost.com/entry/roseanne-revivalreview_us_5ab3a497e4b054d118e04365,

the domain name will be: https://www.huffingtonpost.com

I use `split('.com')` nested of `split('/')` to handel cases like `https://www.huffingtonpost.comhttps`

```
[74] # sarcasm['domain_name'] = sarcasm['article_link'].apply(lambda x: '/'.join(x.split('/')[:3]))
     # 4 https://www.huffingtonpost.com  14403
     # 5 https://www.huffingtonpost.comhttp: 503
     # 6 https://www.huffingtonpost.comhttps:  79 considerd as 3 domin_name

     # add domain_name by extract from the article link
     # split using .com
     # example: 'https://www.huffingtonpost.com/entry/roseanne-revival-review_us_5ab3a497e4b054d118e04365'.split('.com')
     # results  ['https://www.huffingtonpost', '/entry/roseanne-revival-review_us_5ab3a497e4b054d118e04365']
     # then join the first part [0] with .com
     sarcasm['domain_name'] = sarcasm['article_link'].apply(lambda x: x.split('.com')[0] + '.com' if '.com' in x else x)
     sarcasm[['article_link','domain_name']].head(3)
```

	article_link	domain_name
0	https://www.huffingtonpost.com/entry/versace-b...	https://www.huffingtonpost.com
1	https://www.huffingtonpost.com/entry/roseanne-...	https://www.huffingtonpost.com
2	https://local.theonion.com/mom-starting-to-fea...	https://local.theonion.com

25.Group headlines by the domain name. Plot bar plot showing the number of headlines per the domain name

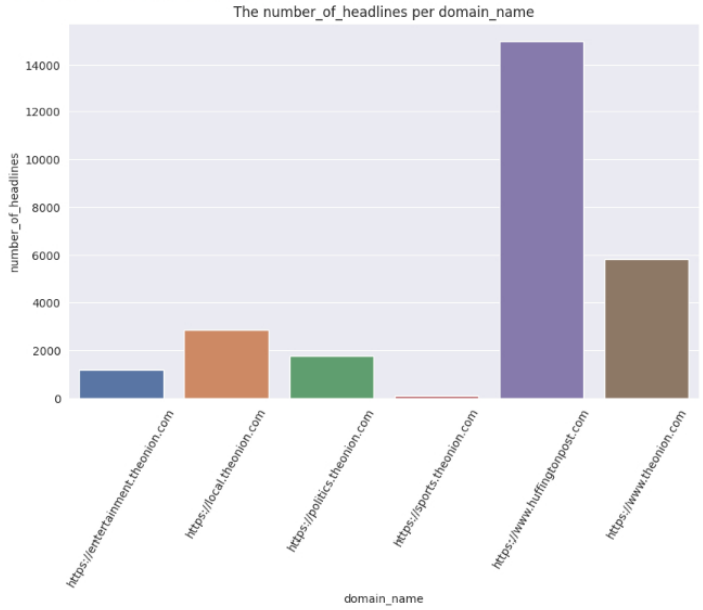
```
[75] headlines = sarcasm.groupby('domain_name').size().reset_index(name='number_of_headlines')
     #Converts the groupby result to a df using reset_index becouse barplot take df
     headlines
```

	domain_name	number_of_headlines
0	https://entertainment.theonion.com	1194
1	https://local.theonion.com	2852
2	https://politics.theonion.com	1767
3	https://sports.theonion.com	100
4	https://www.huffingtonpost.com	14985
5	https://www.theonion.com	5811

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

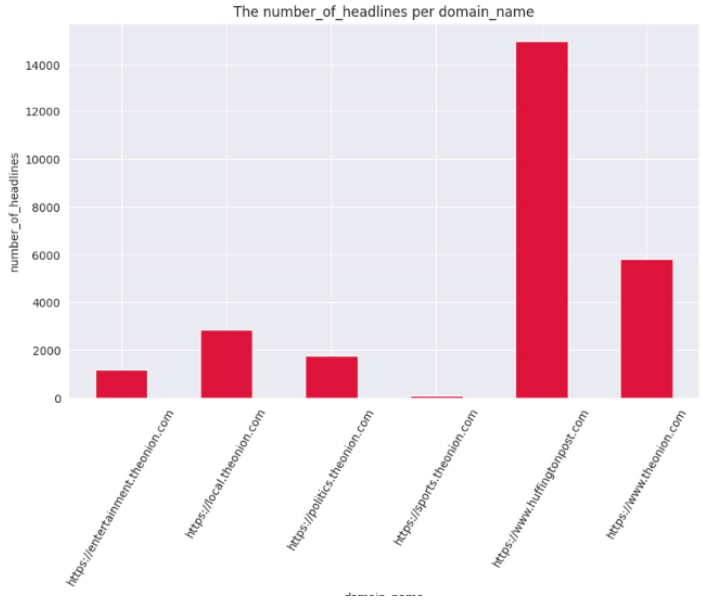
```
[76] plt.figure(figsize=(10, 6))
     # bar plot for showing the number of headlines per the domain name
     sns.barplot(x='domain_name', y='number_of_headlines', data=headlines , palette='deep', hue='domain_name')
     plt.xlabel('domain_name')
     plt.ylabel('number_of_headlines')
     plt.title('The number_of_headlines per domain_name')
     plt.xticks(rotation=60)
```

```
([0, 1, 2, 3, 4, 5],
 [Text(0, 0, 'https://entertainment.theonion.com'),
  Text(1, 0, 'https://local.theonion.com'),
  Text(2, 0, 'https://politics.theonion.com'),
  Text(3, 0, 'https://sports.theonion.com'),
  Text(4, 0, 'https://www.huffingtonpost.com'),
  Text(5, 0, 'https://www.theonion.com')])
```




```
[77] # other way
     # group by domain_nameand count number_of_headlines
     headlines = sarcasm.groupby('domain_name').size()
     plt.figure(figsize=(10, 6))
     headlines.plot(color='crimson' , kind='bar')
     plt.xlabel('domain_name')
     plt.ylabel('number_of_headlines')
     plt.title('The number_of_headlines per domain_name')
     plt.xticks(rotation=60)
```

```
(array([0, 1, 2, 3, 4, 5]),
 [Text(0, 0, 'https://entertainment.theonion.com'),
  Text(1, 0, 'https://local.theonion.com'),
  Text(2, 0, 'https://politics.theonion.com'),
  Text(3, 0, 'https://sports.theonion.com'),
  Text(4, 0, 'https://www.huffingtonpost.com'),
  Text(5, 0, 'https://www.theonion.com')])
```



domain_name

26. Apply the following preprocessing steps on the headlines in sequence:

```
[78] import nltk
```

- Tokenization.

في هاي الخطوه هنحول النصوص لكلمات

```
[79] from nltk.tokenize import wordpunct_tokenize
     nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
True
```

```
[80] headlines = sarcasm['headline'].values
     tokenize_phase =[wordpunct_tokenize(i.lower()) for i in headlines]
     headlines[0] , tokenize_phase[0]
```

```
("former versace store clerk sues over secret 'black code' for minority shoppers",
 ['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'over',
  'secret',
  "'",
  'black',
  'code',
  "'",
  'for',
  'minority',
  'shoppers'])
```

- Stopwords and punctuation removal

في هاي الخطوه هنعمل كلين للداتا ونقيم علامات الترقيم والكلمات اللي ما الها تاثير ولا دلالة في المعنى وممكن تعمل تحيز في الموديل اللي هنبني

```
[81] nltk.download('stopwords')
     en_stopwords = nltk.corpus.stopwords.words('english')
     import string
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

```
[82] cleaning_phase1 =[]
     for headline_tokens in tokenize_phase:
       cleaning_phase1.append([i for i in headline_tokens if i not in en_stopwords])

     tokenize_phase[0] , cleaning_phase1[0]
```

```
(['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'over',
  'secret',
  "'",
  'black',
  'code',
  "'",
  'for',
  'minority',
  'shoppers'],
 ['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'secret',
  "'",
  'black',
  'code',
  "'",
  'minority',
  'shoppers'])
```

```
[83] to_remove = string.punctuation+'0123456789'
     cleaning_phase2=[]
     for headline_tokens in cleaning_phase1:
       cleaning_phase2.append([i for i in headline_tokens if i not in to_remove])

     cleaning_phase1[0] , cleaning_phase2[0]
```

```
(['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'secret',
  "'",
  'black',
  'code',
  "'",
  'minority',
  'shoppers'],
 ['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'secret',
  'black',
  'code',
  'minority',
  'shoppers'])
```

- Stemming (Use Snowball stemmer)

في هاي الخطوه هنرجع كل كلمة لأصلها أو جذرها بحيث نجمع الكلمات المتشابهه وان كانت مش نتائج دقيقي لكن عملية التحويل واحده للكلمات المتشابهة

```
[84] from nltk.stem import SnowballStemmer
     stemmer = SnowballStemmer(language='english')
```

```
[85] stemming_phase=[]
     for headline_tokens in cleaning_phase2:
       stemming_phase.append([stemmer.stem(i) for i in headline_tokens])

     cleaning_phase2[0] , stemming_phase[0]
```

```
(['former',
  'versace',
  'store',
  'clerk',
  'sues',
  'secret',
  'black',
  'code',
  'minority',
  'shoppers'],
 ['former',
  'versac',
```

```
'store',
'clerk',
'sue',
'secret',
'black',
'code',
'minor',
'shopper'])
```

- Vectorization by using TF-IDF:

في هاي الخطوه هنحول النصوص لفيكتورز وارقام قادر المودل يتعامل معها

```
[86] from sklearn.feature_extraction.text import TfidfVectorizer
     vectorizer = TfidfVectorizer(min_df=2, max_features=500) #make vectorize for most frequent 500 word
```

```
[87] # make string for vectorization
     vectorize_phase=[' '.join(i) for i in stemming_phase]
     vectorize_phase[0]
```

```
'former versac store clerk sue secret black code minor shopper'
```

```
[88] # vectorization using TF-IDF
     tfidf_matrix = vectorizer.fit_transform(vectorize_phase) # ready to use in ML

     tfidf_matrix.toarray()
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])
```

```
[89] feature_names = vectorizer.get_feature_names_out()
```

```
# words weghit on first row
for i, weight in enumerate(tfidf_matrix.toarray()[0]):
  if weight != 0:
    print(f"{feature_names[i]}: {weight}")
```

```
black: 0.5253732681364958
former: 0.6135084708809126
secret: 0.5895678801333585
```