## Project Introduction:

The aim of this project is to thoroughly explore and analyze the California housing dataset, a widely used dataset for housing-related analysis. This dataset provides various features associated with housing in different regions of California. We will perform comprehensive data preprocessing, in-depth analysis, and visualizations to extract meaningful insights from the data.

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

```
df=pd.read_csv('/content/drive/MyDrive/odinschooldatasets/housing+(1).csv')
df.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | hou |
|---|---|---|---|---|---|---|---|
| 0 | -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | |
| 1 | -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | |
| 2 | -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | |
| 3 | -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | |
| 4 | -122.25 | 37.85 | 52 | 1627 | 280.0 | 565 | |

The features in the dataset can be classified into different types based on their characteristics:

**CATEGORICAL FEATURES**

(i) Nominal= The categories of nominal data are purely descriptive.They can never be quantified.In this dataset, the 'ocean_proximity' feature falls under this category as it represents different categories like 'NEAR BAY', 'INLAND', 'NEAR OCEAN', 'ISLAND', and 'NEAR OCEAN'.

(ii) Ordinal= These are categorical features with a specific order. There are no ordinal features in this dataset.

**NUMERICAL FEATURES**

(i) Discrete= These features have distinct, separate values that are usually integers. 'housing_median_age', 'total_rooms', 'total_bedrooms', 'population', and 'households' are discrete features in this dataset.

(ii) Continuous= These features can take any real value within a range. 'longitude', 'latitude', 'median_income', and 'median_house_value' are continuous features in this dataset.

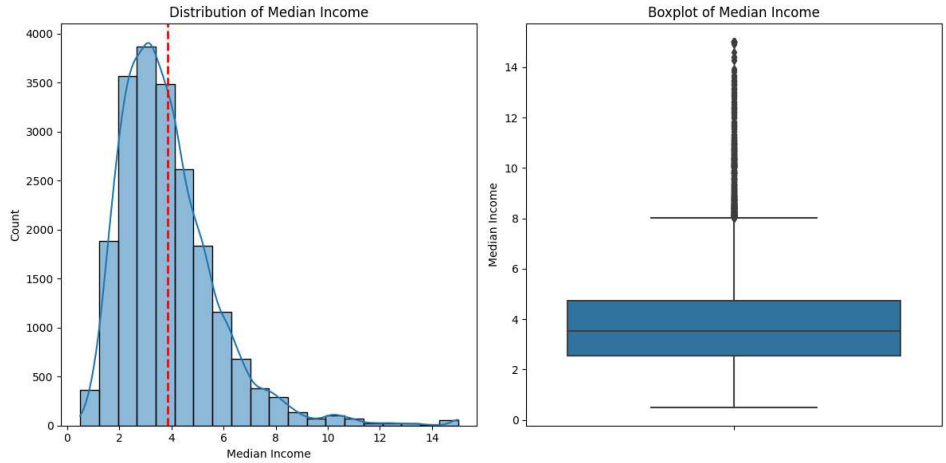
```
df.info()
```

```
<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  20640 non-null  int64  
 3   total_rooms         20640 non-null  int64  
 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  20640 non-null  int64  
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

**1. What is the average median income of the data set and check the distribution of data using appropriate plots. Please explain the distribution of the plot.**

```
average_median_income = df['median_income'].mean()

fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

sns.histplot(data=df, x='median_income', bins=20, kde=True, ax=ax1)
```

```
ax1.axvline(average_median_income, color='red', linestyle='dashed', linewidth=2, label='Average')
ax1.set_title("Distribution of Median Income")
ax1.set_xlabel("Median Income")

sns.boxplot(data=df, y='median_income', ax=ax2)
ax2.set_title("Boxplot of Median Income")
ax2.set_ylabel("Median Income")

plt.tight_layout()
plt.show()

print("Average Median Income:", average_median_income)
```




```
Average Median Income: 3.8706710029069766
```

The first line of code, average_median_income = df['median_income'].mean(), calculates the average median income of the data frame df. The mean() function calculates the average of all the values in a column.

The next four lines of code create a new figure with two subplots. The first subplot is a histogram of the median income, with 20 bins and a KDE. The second subplot is a boxplot of the median income.

The line ax1.axvline(average_median_income, color='red', linestyle='dashed', linewidth=2, label='Average') adds a red dashed line to the first subplot, indicating the average median income.

The last two lines of code set the titles and labels for the two subplots, and then show the figure.

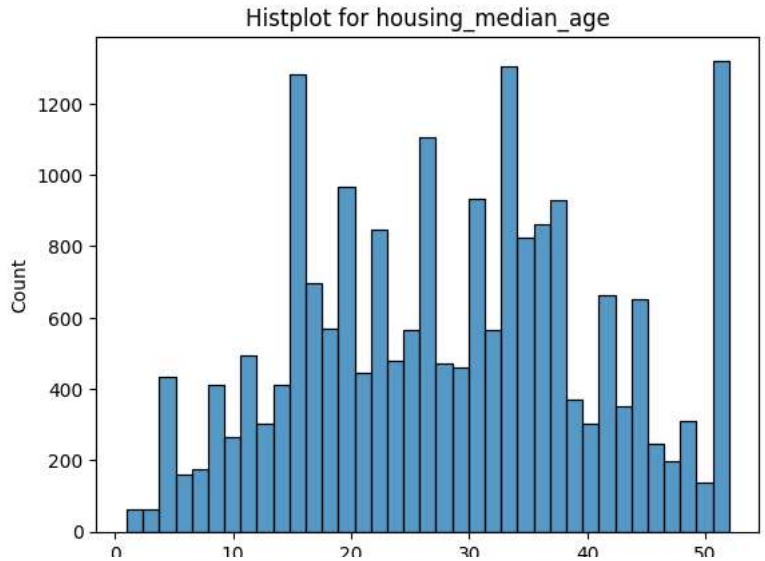
The code outputs the following:

Boxplot of Median Income Median Income The first plot shows the distribution of the median income. The distribution is somewhat right-skewed, meaning that there are more values towards the lower end of the income range.

The second plot shows the boxplot of the median income. The middle line of the box is median.It is not going to be exactly on the middle.The distribution is somewhat right-skewed.

**2. Draw an appropriate plot to see the distribution of housing_median_age and explain your observations.**

```
sns.histplot(df['housing_median_age'])
plt.title("Histplot for housing_median_age")
plt.xlabel("housing_median_age")
plt.show()
```

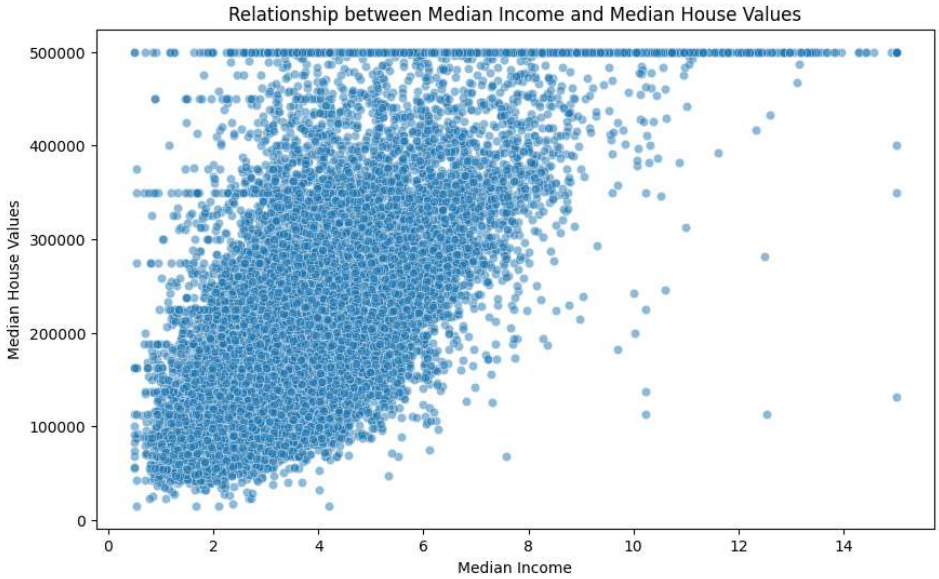



The histplot() function from the Seaborn library is used to create the histogram. The plt.title() and plt.xlabel() functions are used to set the title and label of the plot, respectively. The plt.show() function is used to display the plot.

The histogram shows the distribution of the housing median age values. The distribution is somewhat skewed to the left, meaning that there are more values towards the lower end of the age range. The peak of the distribution is around 20 years, and there are a few values that are much older than the rest.

**3. Show with the help of visualization, how median_income and median_house_values are related?**

```
plt.figure(figsize=(10, 6))
sns.scatterplot(data=df, x='median_income', y='median_house_value', alpha=0.5)
plt.title("Relationship between Median Income and Median House Values")
plt.xlabel("Median Income")
plt.ylabel("Median House Values")
plt.show()
```




median_income and median_house_values both columns have continuous data.The scatterplot() function from the Seaborn library is used to create the scatterplot. The plt.title() and plt.xlabel() functions are used to set the title and label of the plot, respectively. The plt.show() function is used to display the plot.

The output of the code is a scatterplot that shows the relationship between the median income and median house values. The plot shows that there is a positive correlation between the two variables, meaning that as the median income increases, the median house values also tend to increase.

***4. Create a data set by deleting the corresponding examples from the data set for which total_bedrooms are not available.***

```
df.dropna(subset=['total_bedrooms']).reset_index(drop=True)
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | hou |
|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | |
| **1** | -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | |
| **2** | -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | |
| **3** | -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | |
| **4** | -122.25 | 37.85 | 52 | 1627 | 280.0 | 565 | |
| **...** | ... | ... | ... | ... | ... | ... | |
| **20428** | -121.09 | 39.48 | 25 | 1665 | 374.0 | 845 | |
| **20429** | -121.21 | 39.49 | 18 | 697 | 150.0 | 356 | |
| **20430** | -121.22 | 39.43 | 17 | 2254 | 485.0 | 1007 | |
| **20431** | -121.32 | 39.43 | 18 | 1860 | 409.0 | 741 | |
| **20432** | -121.24 | 39.37 | 16 | 2785 | 616.0 | 1387 | |

20433 rows × 10 columns

dropna() drops any rows in the DataFrame df where the total_bedrooms column has a missing value. Then reset_index resets the index of the DataFrame, dropping the old index column.

**5. Create a data set by filling the missing data with the mean value of the total_bedrooms in the original data set.**

```
mean_total_bedrooms =df['total_bedrooms'].mean()
filled_data = df.fillna({'total_bedrooms ': mean_total_bedrooms})
filled_data.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | household |
|---|---|---|---|---|---|---|---|
| **0** | -122.23 | 37.88 | 41 | 880 | 129.0 | 322 | |
| **1** | -122.22 | 37.86 | 21 | 7099 | 1106.0 | 2401 | 1 |
| **2** | -122.24 | 37.85 | 52 | 1467 | 190.0 | 496 | |
| **3** | -122.25 | 37.85 | 52 | 1274 | 235.0 | 558 | 2 |
| **4** | -122.25 | 37.85 | 52 | 1627 | 280.0 | 565 | 2 |

The code mean_total_bedrooms =df['total_bedrooms'].mean() calculates the mean of the total_bedrooms column in the DataFrame df. The mean() function calculates the average of all the values in a column.

The next line of code, filled_data = df.fillna({'total_bedrooms ': mean_total_bedrooms}), fills all the missing values in the total_bedrooms column with the mean value. The fillna() function takes a dictionary as its argument, where the keys are the column names and the values are the values to use to fill the missing values.

The last line of code, filled_data.head(), prints the first 5 rows of the DataFrame filled_data.

**6. Write a programming construct (create a user defined function) to calculate the median value of the data set wherever required.**

```
def median(data):
  data.sort()
  if len(data) % 2 == 0:
    median = (data[len(data) // 2] + data[len(data) // 2 - 1]) / 2
  else:
    median = data[len(data) // 2]
```

```
  return median

median([10.0,20.0,30.5,40.0])

    25.25
```

The function median() takes a list of numbers as input and returns the median value of the data set. The median is the middle value of a sorted distribution. If the data set has an even number of elements, the median is the average of the two middle values.

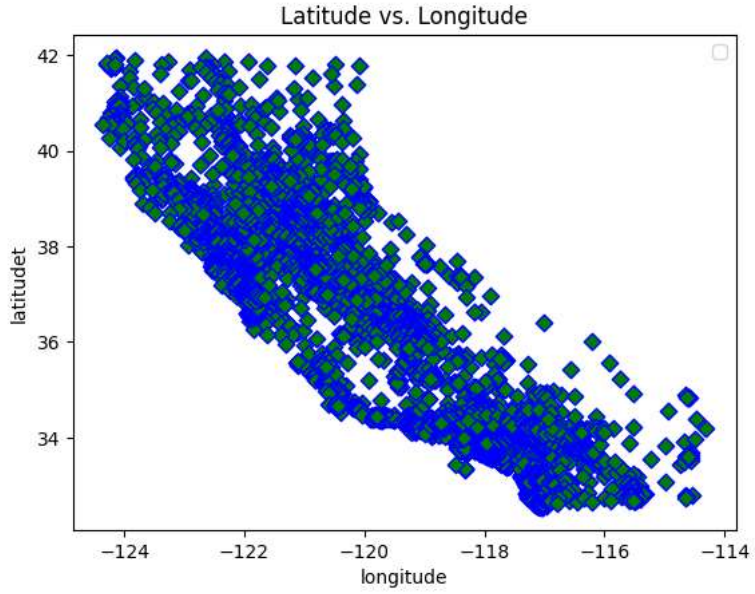
The function works as follows:

The data.sort() statement sorts the data in ascending order. The if len(data) % 2 == 0 statement checks if the length of the data set is even. If it is, the median is the average of the two middle values. The else statement executes if the length of the data set is odd. In this case, the median is the middle value. The return median statement returns the median value.

**7. Plot latitude versus longitude and explain your observations.**

```
plt.scatter(df['longitude'],df['latitude'], edgecolor = 'blue', facecolor = 'green', s = 40, marker = 'D')
plt.legend()
plt.xlabel("longitude")
plt.ylabel("latitudet")
plt.title('Latitude vs. Longitude')
plt.plot()
```

```
WARNING:matplotlib.legend:No artists with labels found to put in legend.  Note that artists
[]
```




The code plt.scatter(df['longitude'],df['latitude'], edgecolor = 'blue', facecolor = 'green', s = 40, marker = 'D') creates a scatter plot of the longitude and latitude columns in the DataFrame df.

The edgecolor parameter specifies the color of the edges of the points, the facecolor parameter specifies the color of the fill, the s parameter specifies the size of the points, and the marker parameter specifies the shape of the points.

The next line of code, plt.legend(), adds a legend to the plot. The legend identifies the different markers in the plot.

The following two lines of code, plt.xlabel("longitude") and plt.ylabel("latitudet"), set the labels for the x-axis and y-axis, respectively.

The last line of code, plt.title('Latitude vs. Longitude'), sets the title of the plot.

The output of the code is a scatter plot that shows the distribution of the longitude and latitude values. The plot shows that the points are concentrated around a certain region, which is likely the area where the data was collected. There are also some outliers, which are points that are far away from the main concentration of points.

**8. Create a data set for which the ocean_proximity is 'Near ocean'.**

```
new_df = df[df['ocean_proximity'] == 'NEAR OCEAN']
new_df.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | house |
|---|---|---|---|---|---|---|---|
| **1850** | -124.17 | 41.80 | 16 | 2739 | 480.0 | 1259 | |
| **1851** | -124.30 | 41.80 | 19 | 2672 | 552.0 | 1298 | |
| **1852** | -124.23 | 41.75 | 11 | 3159 | 616.0 | 1343 | |
| **1853** | -124.21 | 41.77 | 17 | 3461 | 722.0 | 1947 | |
| **1854** | -124.19 | 41.78 | 15 | 3140 | 714.0 | 1645 | |

The code new_df = df[df['ocean_proximity'] == 'NEAR OCEAN'] creates a new DataFrame called new_df that only contains the rows where the ocean_proximity column is equal to 'NEAR OCEAN'. The df[ ] operator is used to filter the DataFrame df. The head() function prints the first 5 rows of the DataFrame.

The output of the code is the first 5 rows of the DataFrame new_df, which only contains the rows where the ocean_proximity column is equal to 'NEAR OCEAN'.

**9. Find the mean and median of the median income for the data set created in question 8.**

```
mean=new_df['median_income'].mean()
median=new_df['median_income'].median()

print('Mean of the median income =',mean)
print('Median of the median income =',median)
```

```
Mean of the median income = 4.0057848006019565
Median of the median income = 3.64705
```

The code mean=new_df['median_income'].mean() calculates the mean of the median_income column in the DataFrame new_df. The mean() function calculates the average of all the values in a column.

The next line of code, median=new_df['median_income'].median(), calculates the median of the median_income column in the DataFrame new_df. The median() function calculates the middle value of a sorted distribution.

The last line of code, print('Mean of the median income =',mean), prints the mean of the median income. The print() function prints the output of the expression to the console.

**10. Please create a new column named total_bedroom_size. If the total bedrooms is 10 or less, it should be quoted as small. If the total bedrooms is 11 or more but less than 1000, it should be medium, otherwise it should be considered large.**

```
new_df['total_bedroom_size']=None

for j, i in enumerate(df["total_bedrooms"]):
  if i <=10:
    new_df.loc[j, "total_bedroom_size"] = "small"
  elif 11<=i<1000:
    new_df.loc[j, "total_bedroom_size"] = "medium"
  else:
    new_df.loc[j, "total_bedroom_size"] = "large"

new_df.head()
```

| | longitude | latitude | housing_median_age | total_rooms | total_bedrooms | population | house |
|---|---|---|---|---|---|---|---|
| **1850** | -124.17 | 41.80 | 16.0 | 2739.0 | 480.0 | 1259.0 | |
| **1851** | -124.30 | 41.80 | 19.0 | 2672.0 | 552.0 | 1298.0 | |
| **1852** | -124.23 | 41.75 | 11.0 | 3159.0 | 616.0 | 1343.0 | |
| **1853** | -124.21 | 41.77 | 17.0 | 3461.0 | 722.0 | 1947.0 | |
| **1854** | -124.19 | 41.78 | 15.0 | 3140.0 | 714.0 | 1645.0 | |

The code new_df['total_bedroom_size']=None creates a new column in the DataFrame new_df called total_bedroom_size. The column is initially set to None.

The next line of code, for j, i in enumerate(df["total_bedrooms"]):, creates a loop that iterates over the rows of the DataFrame df. The enumerate() function returns a tuple of the row index and the row value for each row.

The next two lines of code, if i <=10: and new_df.loc[j, "total_bedroom_size"] = "small", check if the number of bedrooms in the current row is less than or equal to 10. If it is, the code sets the value of the total_bedroom_size column in the current row to "small".

The next two lines of code, elif 11<=i<1000: and new_df.loc[j, "total_bedroom_size"] = "medium", check if the number of bedrooms in the current row is greater than or equal to 11 and less than 1000. If it is, the code sets the value of the total_bedroom_size column in the current row to "medium".

The last two lines of code, else: and new_df.loc[j, "total_bedroom_size"] = "large", set the value of the total_bedroom_size column in the current row to "large" if the number of bedrooms is greater than or equal to 1000.

The last line of code, new_df.head(), prints the first 5 rows of the DataFrame new_df.