California Housing Price Prediction

Problem Statement :

The US Census Bureau has published California Census Data which has 10 types of metrics such as the population, median income, median housing price, and so on for each block group in California. The dataset also serves as an input for project scoping and tries to specify the functional and nonfunctional requirements for it.

Objective :

The project aims at building a model of housing prices to predict median house values in California using the provided dataset. This model should learn from the data and be able to predict the median housing price in any district, given all the other metrics.
Districts or block groups are the smallest geographical units for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). There are 20,640 districts in the project dataset.

Domain: Finance and Housing

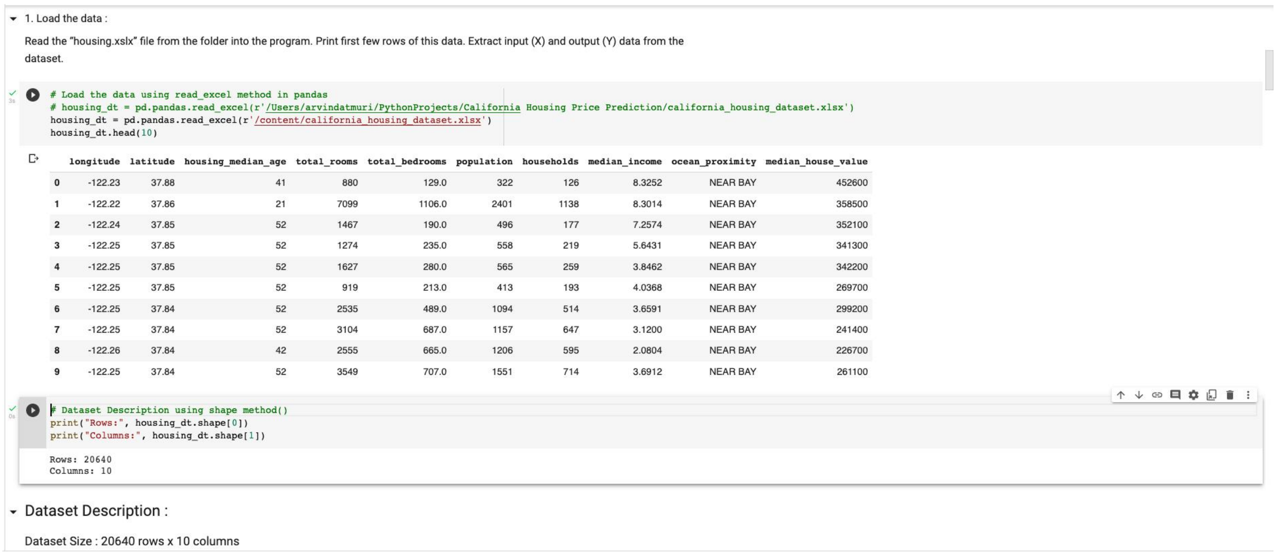
Analysis Tasks to be performed:

- Build a model of housing prices to predict median house values in California using the provided dataset.
- Train the model to learn from the data to predict the median housing price in any district, given all the other metrics.
- Predict housing prices based on median_income and plot the regression chart for it.

Task 1: Load the data

- Read the “housing.xslx” file from the folder into the program.
- Print first few rows of this data. Extract input (X) and output (Y) data from the dataset.
- So as to load the excel sheet we are using read_excel method from pandas library to create the dataframe for the current dataset.
- Now that we have created a pandas dataframe using given excel sheet, lets check the number of rows and columns provided so as to validate the dataset according to the problem statement provided in the assessment section.
- We will be Extracting X and Y values later once the dataset is preprocessed.

Data Set Description as per the problem Statement: 20640 rows x 10 columns



1. Load the data :

Read the “housing.xslx” file from the folder into the program. Print first few rows of this data. Extract input (X) and output (Y) data from the dataset.

```python
# Load the data using read_excel method in pandas
# housing_dt = pd.pandas.read_excel(r'/Users/arvindatmuri/PythonProjects/California Housing Price Prediction/california_housing_dataset.xlsx')
housing_dt = pd.pandas.read_excel(r'/content/california_housing_dataset.xlsx')
housing_dt.head(10)
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
0	-122.23	37.88	41	880	129.0	322	126	8.3252	NEAR BAY	452600
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	NEAR BAY	358500
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	NEAR BAY	352100
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	NEAR BAY	341300
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	NEAR BAY	342200
5	-122.25	37.85	52	919	213.0	413	193	4.0368	NEAR BAY	269700
6	-122.25	37.84	52	2535	489.0	1094	514	3.6591	NEAR BAY	299200
7	-122.25	37.84	52	3104	687.0	1157	647	3.1200	NEAR BAY	241400
8	-122.26	37.84	42	2555	665.0	1206	595	2.0804	NEAR BAY	226700
9	-122.25	37.84	52	3549	707.0	1551	714	3.6912	NEAR BAY	261100

```python
# Dataset Description using shape method()
print("Rows:", housing_dt.shape[0])
print("Columns:", housing_dt.shape[1])
```

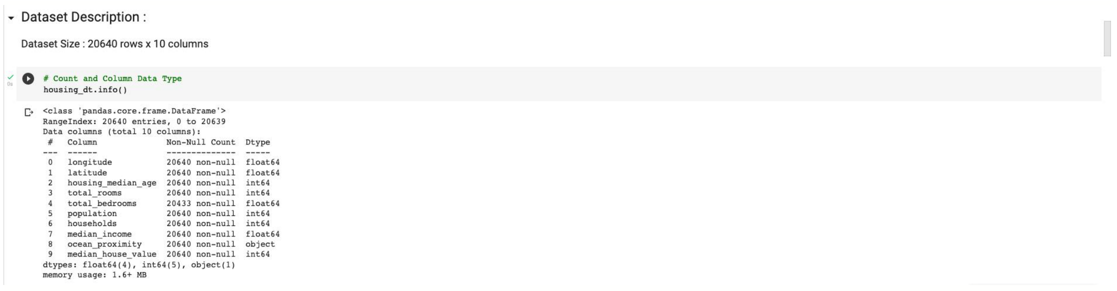
```
Rows: 20640
Columns: 10
```

Dataset Description :

Dataset Size : 20640 rows x 10 columns

Field Description

- longitude (signed numeric - float) : Longitude value for the block in California, USA
- latitude (numeric - float) : Latitude value for the block in California, USA
- housing_median_age (numeric - int) : Median age of the house in the block
- total_rooms (numeric - int) : Count of the total number of rooms (excluding bedrooms) in all houses in the block
- total_bedrooms (numeric - float) : Count of the total number of bedrooms in all houses in the block
- population (numeric - int) : Count of the total number of population in the block
- households (numeric - int) : Count of the total number of households in the block
- median_income (numeric - float) : Median of the total household income of all the houses in the block
- ocean_proximity (numeric - categorical) : Type of the landscape of the block [Unique Values : 'NEAR BAY', '<1H OCEAN', 'INLAND', 'NEAR OCEAN', 'ISLAND']
- median_house_value (numeric - int) : Median of the household prices of all the houses in the block


Dataset Description :

Dataset Size : 20640 rows x 10 columns

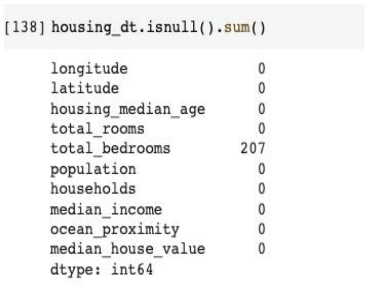
```
# Count and Column Data Type
housing_dt.info()
```

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

As per the description, there is only one Categorical/Object column which may have to be encoded later for better results. Apart from that, all the other columns looks good (since they are already in Numerical format)

Task 2: Handle missing values :

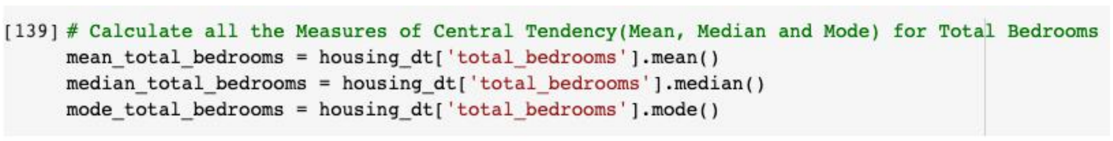
- To identify the null values in the dataset. We are using pandas internal methods to identify the number of null values in each and every column.



```
[138] housing_dt.isnull().sum()
```

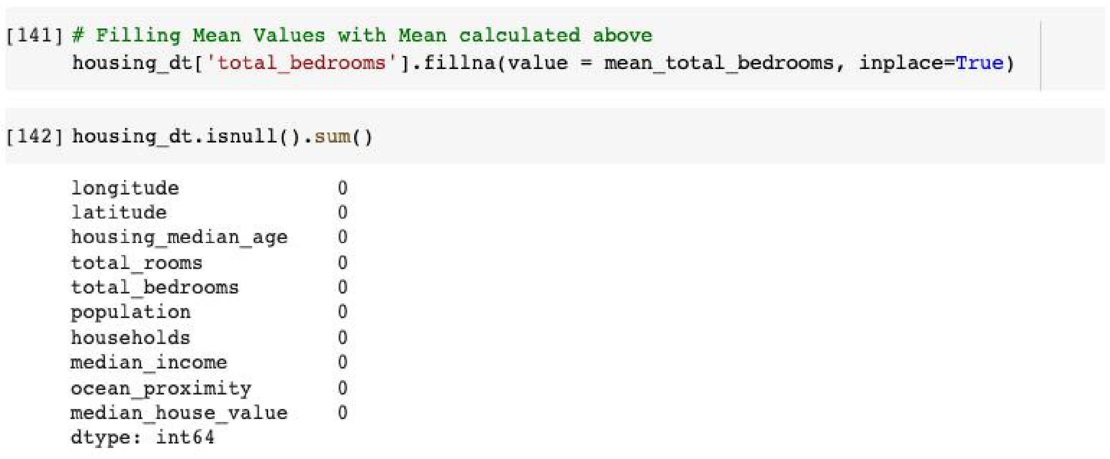
```
longitude               0
latitude                0
housing_median_age      0
total_rooms             0
total_bedrooms        207
population              0
households              0
median_income           0
ocean_proximity         0
median_house_value      0
dtype: int64
```

- To replace the null values, we need to calculate the measures of central tendency.
- There are in total 3 measures - Mean, Median, Mode
  - The mean is the same as the average value of a data set and is found using a calculation. Add up all of the numbers and divide by the number of numbers in the data set.
  - The median is the central number of a data set. Arrange data points from smallest to largest and locate the central number. This is the median. If there are 2 numbers in the middle, the median is the average of those 2 numbers.
  - The mode is the number in a data set that occurs most frequently. Count how many times each number occurs in the data set. The mode is the number with the highest tally. It's OK if there is more than one mode. And if all numbers occur the same number of times there is no mode.



```
[139] # Calculate all the Measures of Central Tendency(Mean, Median and Mode) for Total Bedrooms
      mean_total_bedrooms = housing_dt['total_bedrooms'].mean()
      median_total_bedrooms = housing_dt['total_bedrooms'].median()
      mode_total_bedrooms = housing_dt['total_bedrooms'].mode()
```

- Its always recommended to use any of these measures to fill the null values so as to give the optimal results in place of null values. So lets Calculate the three values using the predefined methods.
- As per the Task, Fill the missing values with the mean of the respective column and check accordingly.



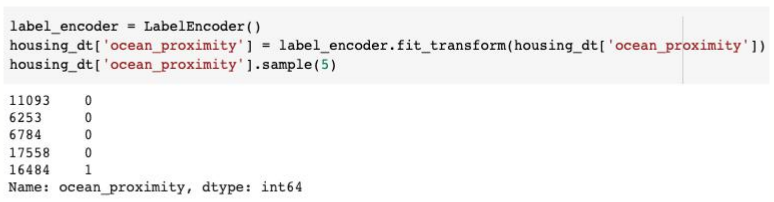
```
[141] # Filling Mean Values with Mean calculated above
      housing_dt['total_bedrooms'].fillna(value = mean_total_bedrooms, inplace=True)
```

```
[142] housing_dt.isnull().sum()
```

```
      longitude             0
      latitude              0
      housing_median_age    0
      total_rooms           0
      total_bedrooms        0
      population            0
      households            0
      median_income         0
      ocean_proximity       0
      median_house_value    0
      dtype: int64
```

Task 3: Encode categorical data :

- Convert categorical column in the dataset to numerical data.
- Looking at the data, all the columns are numerical except to ocean_proximity Column. So lets convert the Categorical Data Column into Numerical Data.



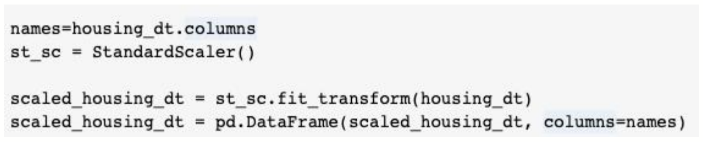
```
label_encoder = LabelEncoder()
housing_dt['ocean_proximity'] = label_encoder.fit_transform(housing_dt['ocean_proximity'])
housing_dt['ocean_proximity'].sample(5)
```

```
11093    0
6253     0
6784     0
17558    0
16484    1
Name: ocean_proximity, dtype: int64
```

- To map the values or to convert categorical values to Numerical values, we can use the LabelEncoder.
- LabelEncoder is used Encode target labels with value between 0 and n_classes-1.
- This transformer should be used to encode target values, i.e. y, and not the input X. It can also be used to transform non-numerical labels (as long as they are Hash-able and comparable) to numerical labels.

Task 4: Standardize data :

- Standardization of a dataset is a common requirement for many machine learning estimators: they might behave badly if the individual features do not more or less look like standard normally distributed data (e.g. Gaussian with 0 mean and unit variance).



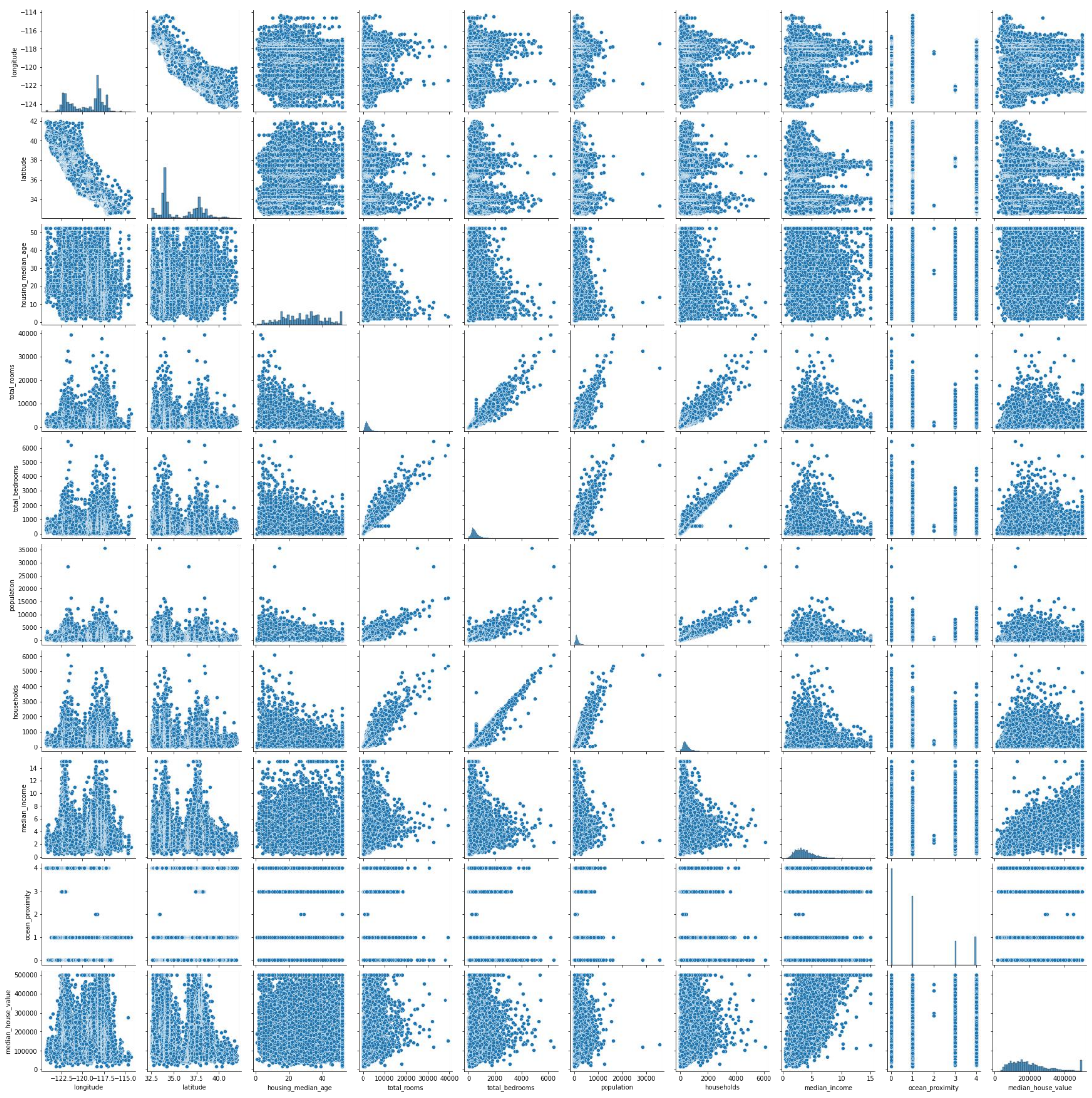
```
names=housing_dt.columns
st_sc = StandardScaler()

scaled_housing_dt = st_sc.fit_transform(housing_dt)
scaled_housing_dt = pd.DataFrame(scaled_housing_dt, columns=names)
```

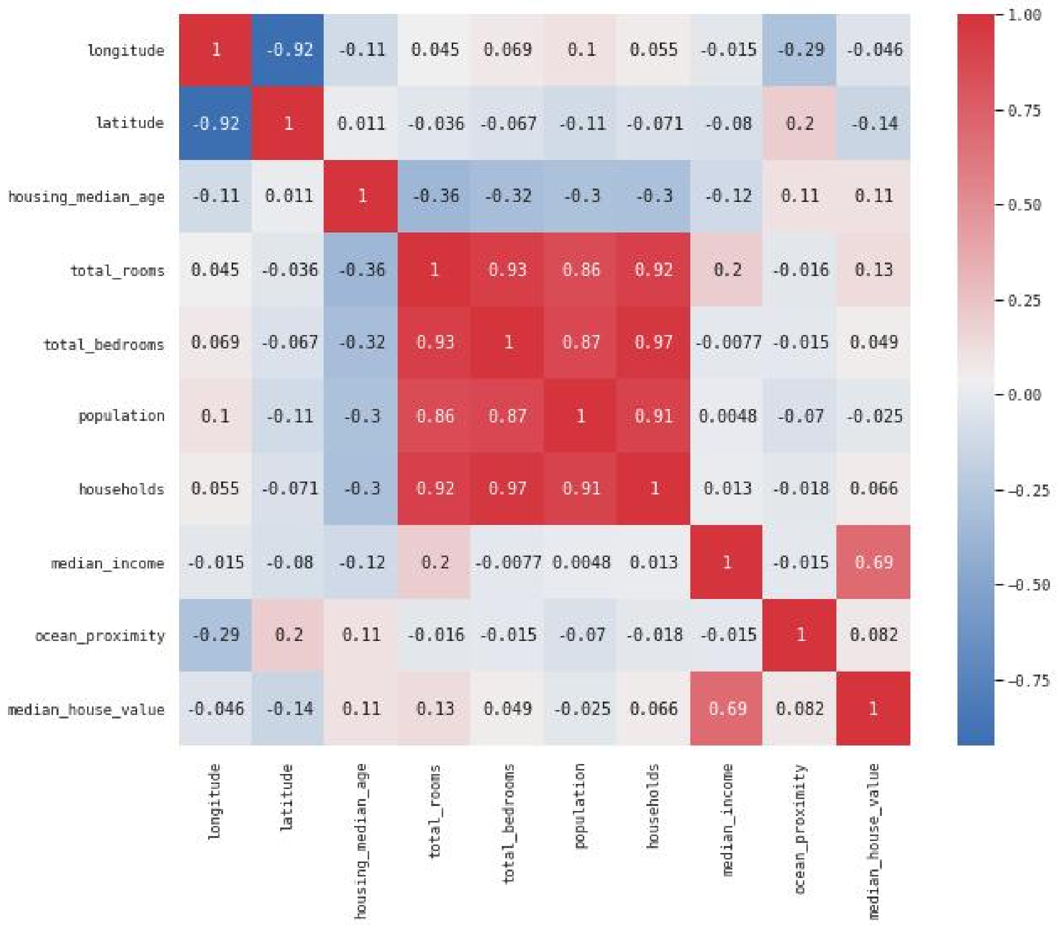
- So to Standardize the data, I have used the Standard Scaler.
- After looking at the data, the data is too large may impact the performance of the model.
- Centering and scaling happen independently on each feature by computing the relevant statistics on the samples in the training set. Mean and standard deviation are then stored to be used on later data using transform.

Task 5: Visualize the Data to Check the linearity among columns :

- First we will be using the scatter pairplot to check the linearity between columns.
- Lets put the preprocessed data into a graph to identify the linearity relationship between any two columns. To Visualize the Data Lets use the scaled data for better results.

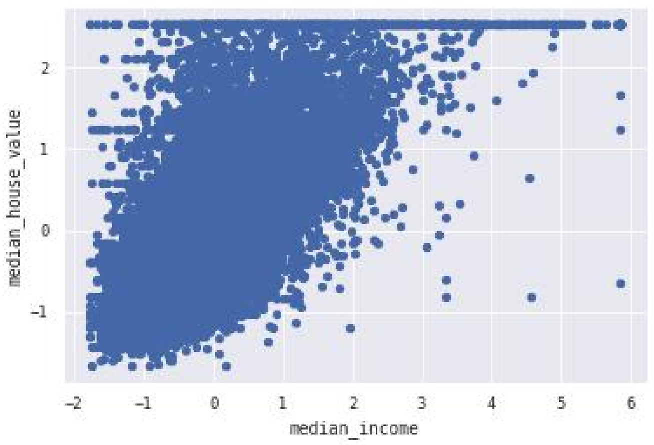


- Also we will be using the Correlation Matrix/Heat-map to look at the relation between the columns and based on that we can decide which independent variable can be chosen the best to check the linearity for our prediction.



	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	median_house_value
longitude	1	-0.92	-0.11	0.045	0.069	0.1	0.055	-0.015	-0.29	-0.046
latitude	-0.92	1	0.011	-0.036	-0.067	-0.11	-0.071	-0.08	0.2	-0.14
housing_median_age	-0.11	0.011	1	-0.36	-0.32	-0.3	-0.3	-0.12	0.11	0.11
total_rooms	0.045	-0.036	-0.36	1	0.93	0.86	0.92	0.2	-0.016	0.13
total_bedrooms	0.069	-0.067	-0.32	0.93	1	0.87	0.97	-0.0077	-0.015	0.049
population	0.1	-0.11	-0.3	0.86	0.87	1	0.91	0.0048	-0.07	-0.025
households	0.055	-0.071	-0.3	0.92	0.97	0.91	1	0.013	-0.018	0.066
median_income	-0.015	-0.08	-0.12	0.2	-0.0077	0.0048	0.013	1	-0.015	0.69
ocean_proximity	-0.29	0.2	0.11	-0.016	-0.015	-0.07	-0.018	-0.015	1	0.082
median_house_value	-0.046	-0.14	0.11	0.13	0.049	-0.025	0.066	0.69	0.082	1

- In the above diagram, Shades and Density of Red Color shows the stronger relation between features.
- From the above two diagrams, the below are our observations.
- Linearity can be observed among very less columns, Not all can be considered for our analysis.
- Latitude vs Longitude - Doesn't make much sense since data provided for these two columns are just Co-ordinates.
- Total_bedrooms vs Total Rooms - Analyzing a relationship between these columns also doesn't make much sense and are more obvious. More number of bedrooms then more will be the number of rooms.
- Households vs Total Bedrooms, Total rooms, Population - Also has linearity between them, but this may not not be useful for our problem statement.
- Last but not least, there is a linearity observed with the Median income and Median House Value - which has a strong relationship with income and house price and should be considered for our problem statement. Also as per the heat-map, Only column very much relatable to the output column is median_income).



- According to Dataset and Problem Statement, we have to find out the Median House Value so this column is considered as Dependent Column/Target Column and also can be classified as Output column. Rest of the columns can be classified as Independent Data.

Task 6: Split the dataset :

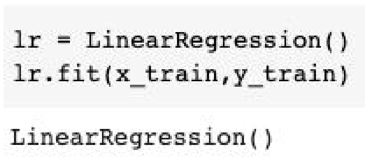
- Split the data into 80% training dataset and 20% test dataset. Using train_test_split to split the dataset into train and test methods.
- Splitting your dataset is essential for an unbiased evaluation of prediction performance. In most cases, it’s enough to split your dataset randomly into three subsets:
    - The training set is applied to train, or fit, your model. For example, you use the training set to find the optimal weights, or coefficients, for linear regression, logistic regression, or neural networks.
    - The validation set is used for unbiased model evaluation during hyper parameter tuning. For example, when you want to find the optimal number of neurons in a neural network or the best kernel for a support vector machine, you experiment with different values. For each considered setting of hyper parameters, you fit the model with the training set and assess its performance with the validation set.
    - The test set is needed for an unbiased evaluation of the final model. You shouldn’t use it for fitting or validation.
- In less complex cases, when you don’t have to tune hyper parameters, it’s okay to work with only the training and test sets.
- This is section we also calculate the X and Y columns in the first section.

```
X = scaled_housing_dt.drop('median_house_value',axis=1)
Y = scaled_housing_dt["median_house_value"]
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,random_state=2503)
x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

- I am using test_train_split for splitting the data. This method has options to split the methods by just passing the arguments.
- options are the optional keyword arguments that you can use to get desired behavior:
  - train_size is the number that defines the size of the training set. If you provide a float, then it must be between 0.0 and 1.0 and will define the share of the dataset used for testing. If you provide an int, then it will represent the total number of the training samples. The default value is None.
  - test_size is the number that defines the size of the test set. It’s very similar to train_size. You should provide either train_size or test_size. If neither is given, then the default share of the dataset that will be used for testing is 0.25, or 25 percent.
  - random_state is the object that controls randomization during splitting. It can be either an int or an instance of Random State. The default value is None.
  - shuffle is the Boolean object (True by default) that determines whether to shuffle the dataset before applying the split.
  - stratify is an array-like object that, if not None, determines how to use a stratified split.

Task 7: Perform Linear Regression :

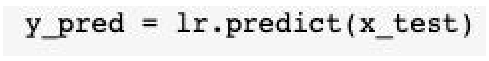
- Linear regression is a linear approach for modeling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).
- This is because models which depend linearly on their unknown parameters are easier to fit than models which are non-linearly related to their parameters and because the statistical properties of the resulting estimators are easier to determine.
- To use Linear Regression we just import the required library and directly use the LinearRegression methods available to create the model to fit our data and predict based on that.
- Under this task, we have 3 sub-tasks:
  - Perform Linear Regression on training data.



```
lr = LinearRegression()
lr.fit(x_train,y_train)
```

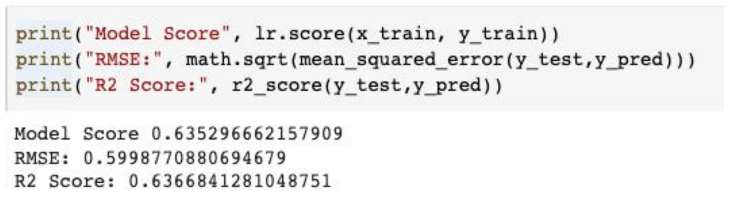
```
LinearRegression()
```

  - Predict output for test dataset using the fitted model.



```
y_pred = lr.predict(x_test)
```

  - Print root mean squared error (RMSE) from Linear Regression.



```
print("Model Score", lr.score(x_train, y_train))
print("RMSE:", math.sqrt(mean_squared_error(y_test,y_pred)))
print("R2 Score:", r2_score(y_test,y_pred))
```

```
Model Score 0.635296662157909
RMSE: 0.5998770880694679
R2 Score: 0.6366841281048751
```