DATA CLEANING OR FEATURE ENGINEERING

TASKS TO BE PERFORMED

1. DATA CLEANING

2. HANDLING DATASET

3. PROBLEM STATEMENT

importing the libraries for the dataset

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
import warnings
import scipy
import matplotlib.image as mpimg
from pandas.plotting import scatter_matrix
from scipy.stats import norm
from numpy.random import randn
from statsmodels.stats.proportion import proportions_ztest
import plotly.graph_objects as go
from scipy.stats import spearmanr
from scipy.stats import shapiro
from scipy.stats import chi2_contingency

warnings.filterwarnings("ignore")

%matplotlib inline
```

reading the dataset

In [2]:

```
df1 = pd.read_csv(r"C:\Users\Hp\Downloads\archive\housing.csv")
```

making a new copy of the dataset

In [3]:

```
df = df1.copy()
```

1. DATA CLEANING

In [4]:

```
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  float64
 3   total_rooms         20640 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  float64
 6   households          20640 non-null  float64
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  float64
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

1.1 missing value handling of total_bedrooms

In [5]:

```
# comment - checking the missing values or nan values in total_bedrooms are 207 entries.
# observation - data has missing values only in total_bedrooms and are 207 entries.
df.isnull().sum()
```

Out[5]:

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

1.1.1 we will fill the missing values by total_bedroom median values

In [6]:

```
# comment - median of total_bedroom feature
# observation - the missing value need to be filled by its median.
                    # I use median because it is less attractive to outliers and best way to
df['total_bedrooms'].median()
```

Out[6]:

```
435.0
```

In [7]:

```
# comment - using fillna inbuilt function of pandas
# observation - filling nan values by median.
df['total_bedrooms'] = df['total_bedrooms'].fillna(df['total_bedrooms'].median())
```

In [8]:

```
df['total_bedrooms'].shape
```

Out[8]:

```
(20640,)
```

In [9]:

```
# comment - after filling values checking is the total_bedrooms values are all filled or no
# observation - zero values interpret that now data is cleaned and no missing values are th
df.isnull().sum()
```

Out[9]:

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

```
# categorical features and numerical features segregating
```

In [10]:

```
categorical_features = [fea for fea in df.columns if df[fea].dtypes == 'O']
print(f'we have {categorical_features} as our categorical feature')
```

```
we have ['ocean_proximity'] as our categorical feature
```

In [11]:

```
numerical_features = [fea for fea in df.columns if df[fea].dtypes != 'O']
print(f'we have {numerical_features} as our numerical feature')
```

```
we have ['longitude', 'latitude', 'housing_median_age', 'total_rooms', 'tota
l_bedrooms', 'population', 'households', 'median_income', 'median_house_valu
e'] as our numerical feature
```

now converting the datatypes of 7 numeric features all together.

In [12]:

```
# comment - earlier column index of original data has float as its datatypes
# observation - now features of index 2 to 9 are all converted to int datatype.
for col in numerical_features[2:9]:
    df[col] = df[col].astype('int')
```

In [13]:

```
df
```

Out[13]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
0	-122.23	37.88	41	880	129	322	
1	-122.22	37.86	21	7099	1106	2401	
2	-122.24	37.85	52	1467	190	496	
3	-122.25	37.85	52	1274	235	558	
4	-122.25	37.85	52	1627	280	565	
...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374	845	
20636	-121.21	39.49	18	697	150	356	
20637	-121.22	39.43	17	2254	485	1007	
20638	-121.32	39.43	18	1860	409	741	
20639	-121.24	39.37	16	2785	616	1387	

20640 rows × 10 columns

In [14]:

```
# comment - now segregating data with only int and float datatypes.
# observation - first two index of data are of float datatypes.
                    # rest index 2 to 8 are of int datatypes.
df[df.dtypes[(df.dtypes == 'float64' )| (df.dtypes == 'int')].index]
```

Out[14]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
0	-122.23	37.88	41	880	129	322	
1	-122.22	37.86	21	7099	1106	2401	
2	-122.24	37.85	52	1467	190	496	
3	-122.25	37.85	52	1274	235	558	
4	-122.25	37.85	52	1627	280	565	
...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374	845	
20636	-121.21	39.49	18	697	150	356	
20637	-121.22	39.43	17	2254	485	1007	
20638	-121.32	39.43	18	1860	409	741	
20639	-121.24	39.37	16	2785	616	1387	

20640 rows × 9 columns

new info of the data and its types

In [15]:

```
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  int32  
 3   total_rooms         20640 non-null  int32  
 4   total_bedrooms      20640 non-null  int32  
 5   population          20640 non-null  int32  
 6   households          20640 non-null  int32  
 7   median_income       20640 non-null  int32  
 8   median_house_value  20640 non-null  int32  
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(2), int32(7), object(1)
memory usage: 1.0+ MB
```

2. handling data

2.1 handling scaling

In [16]:

```
# comment - scaling or normalizing datatypes of categorical features.
# observation - clear picture that 44% of the people are preferring <1H ocean location.
                        ## less preferrence at ISLAND location.
for col in categorical_features:
    print(df[col].value_counts(normalize=True) * 100)
```

```
<1H OCEAN     44.263566
INLAND        31.739341
NEAR OCEAN    12.877907
NEAR BAY      11.094961
ISLAND         0.024225
Name: ocean_proximity, dtype: float64
```

In [17]:

```
# comment -  scaling or normalizing datatypes of numerical features.
for col in numerical_features:
    print(df[col].value_counts(normalize=True) * 100)
```

```
9      0.474806
15     0.247093
11     0.218023
12     0.164729
13     0.106589
14     0.029070
Name: median_income, dtype: float64
500001    4.675388
137500    0.591085
162500    0.566860
112500    0.499031
187500    0.450581
            ...
359200    0.004845
54900     0.004845
377600    0.004845
81200     0.004845
47000     0.004845
Name: median_house_value, Length: 3842, dtype: float64
```

2.2 handling outliers

In [18]:

```
# comment - checking outliers in the data by using boxplot from seaborn .
# observation - numerical features index from 3 to 8 all have outliers in the data.
for i in numerical_features:
    sns.boxplot(df[i])
    plt.show()
```

handling outliers of features index [3:9]

while removing outliers we tried to show

1. outliers before

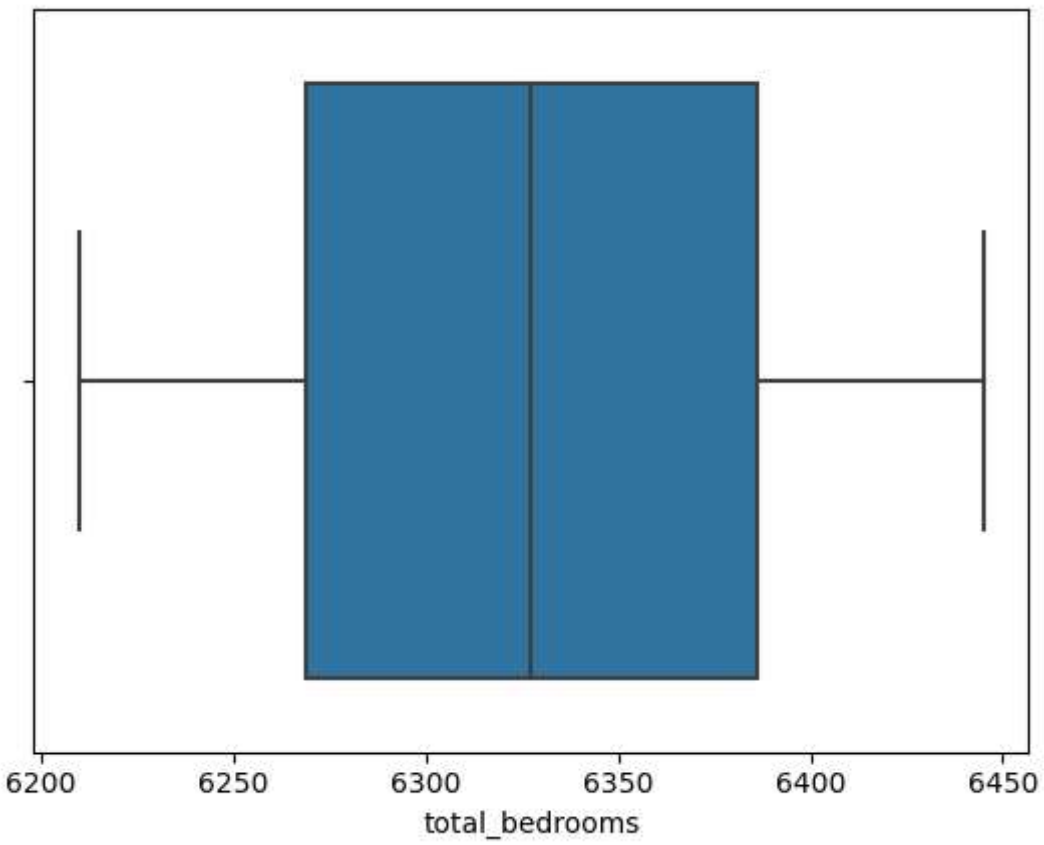
2. outliers after

3. and how many left to remove.

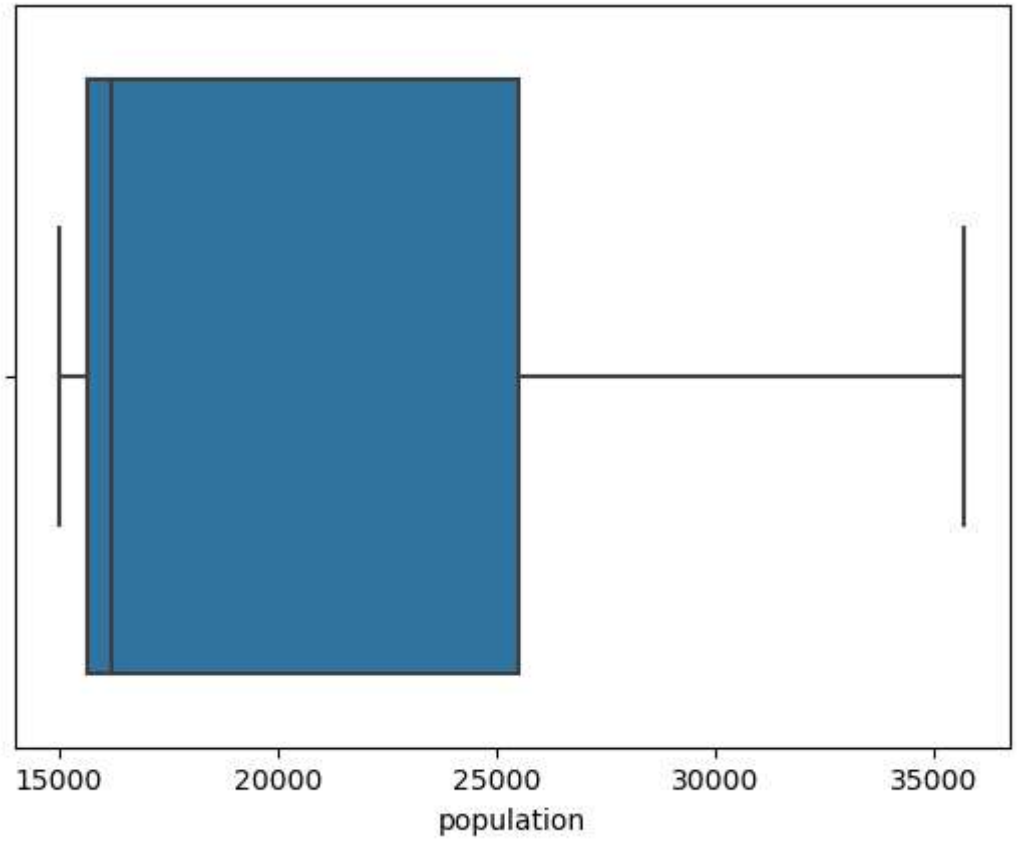
In [19]:

```
for col in numerical_features[4:6]:
    Q1 = np.percentile(df[col], 1,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df[col], 99,
                       interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR

    lower = Q1 - 1.5*IQR
    upper, lower
    new_df = df.loc[(df[col] > upper) | (df[col] < lower)]
    print(f'before removing outliers {len(df)}')
    print(f'after removing outliers {len(new_df)}')
    print('new outliers', len(df) - len(new_df))
    sns.boxplot(new_df[col])
    plt.show()
```

```
before removing outliers 20640
after removing outliers 2
new outliers 20638
```



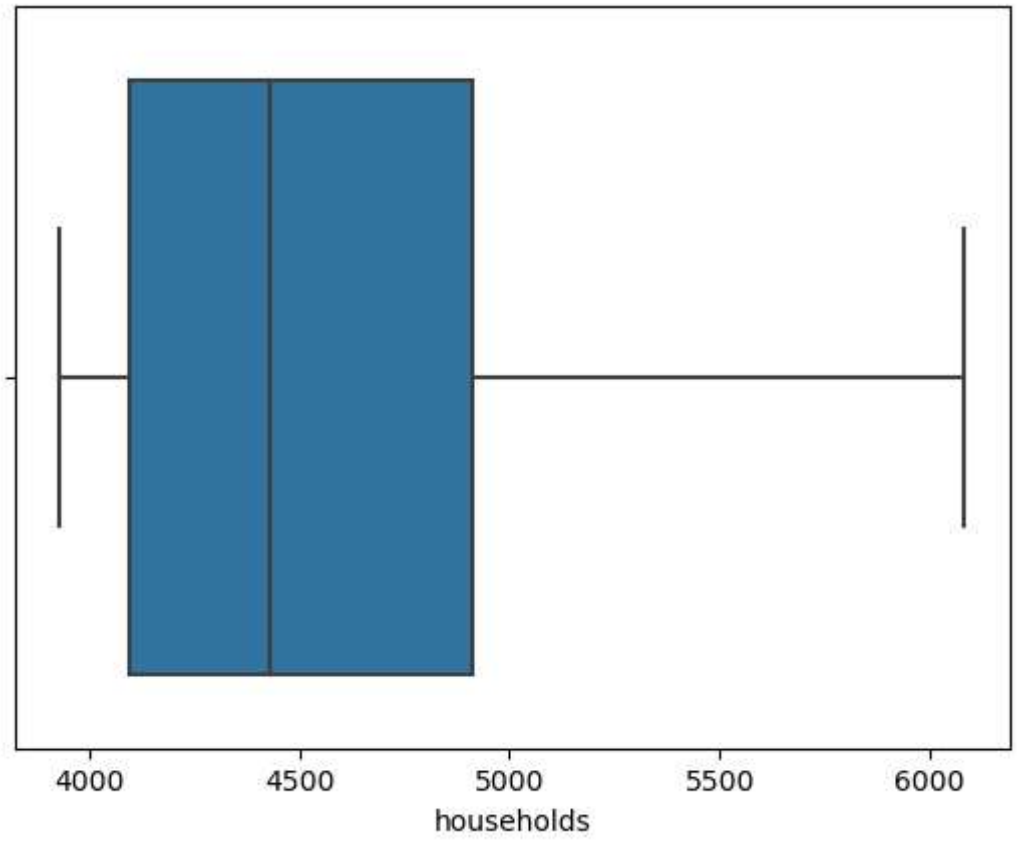
```
before removing outliers 20640
after removing outliers 6
new outliers 20634
```



In [20]:

```
for col in numerical_features[6:7]:
    Q1 = np.percentile(df[col], 1.95,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df[col], 98.05,
                       interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR

    lower = Q1 - 1.5*IQR
    upper, lower
    new_df = df.loc[(df[col] > upper) | (df[col] < lower)]
    print(f'before removing outliers {len(df)}')
    print(f'after removing outliers {len(new_df)}')
    print('new outliers', len(df) - len(new_df))
    sns.boxplot(new_df[col])
    plt.show()
```

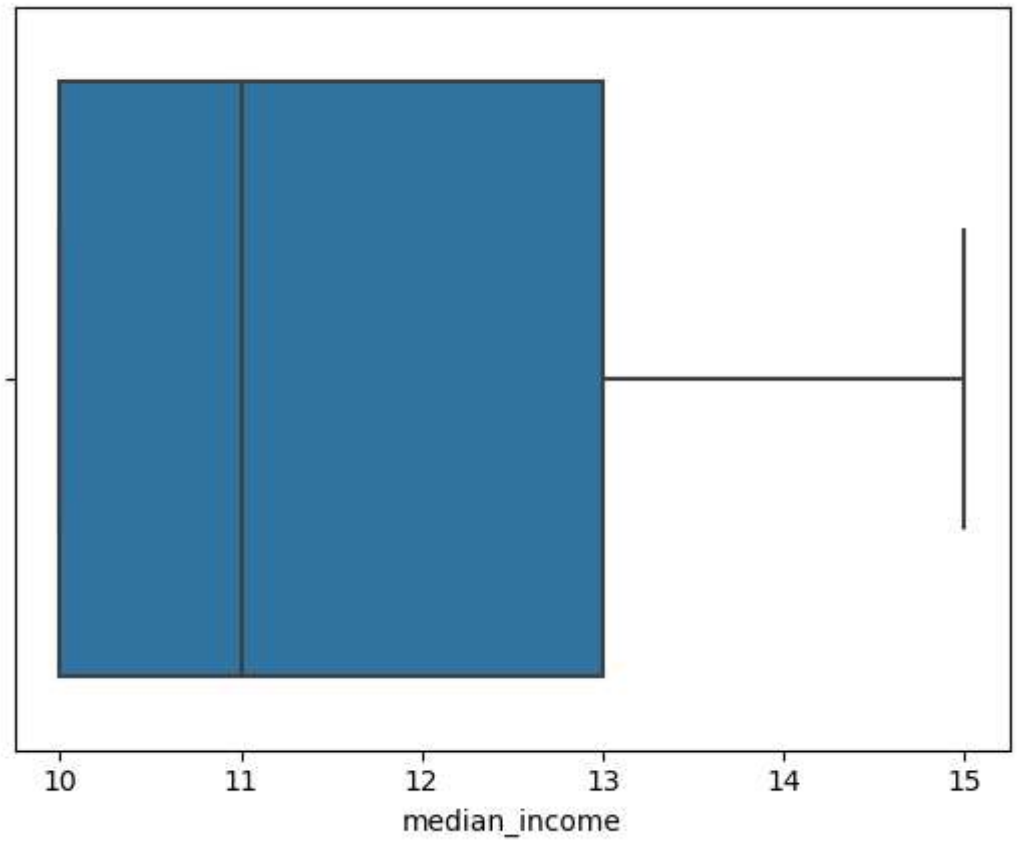
```
before removing outliers 20640
after removing outliers 18
new outliers 20622
```



In [21]:

```
for col in numerical_features[7:8]:
    Q1 = np.percentile(df[col], 15,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df[col], 85,
                       interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR

    lower = Q1 - 1.5*IQR
    upper, lower
    new_df = df.loc[(df[col] > upper) | (df[col] < lower)]
    print(f'before removing outliers {len(df)}')
    print(f'after removing outliers {len(new_df)}')
    print('new outliers', len(df) - len(new_df))
    sns.boxplot(new_df[col])
    plt.show()
```

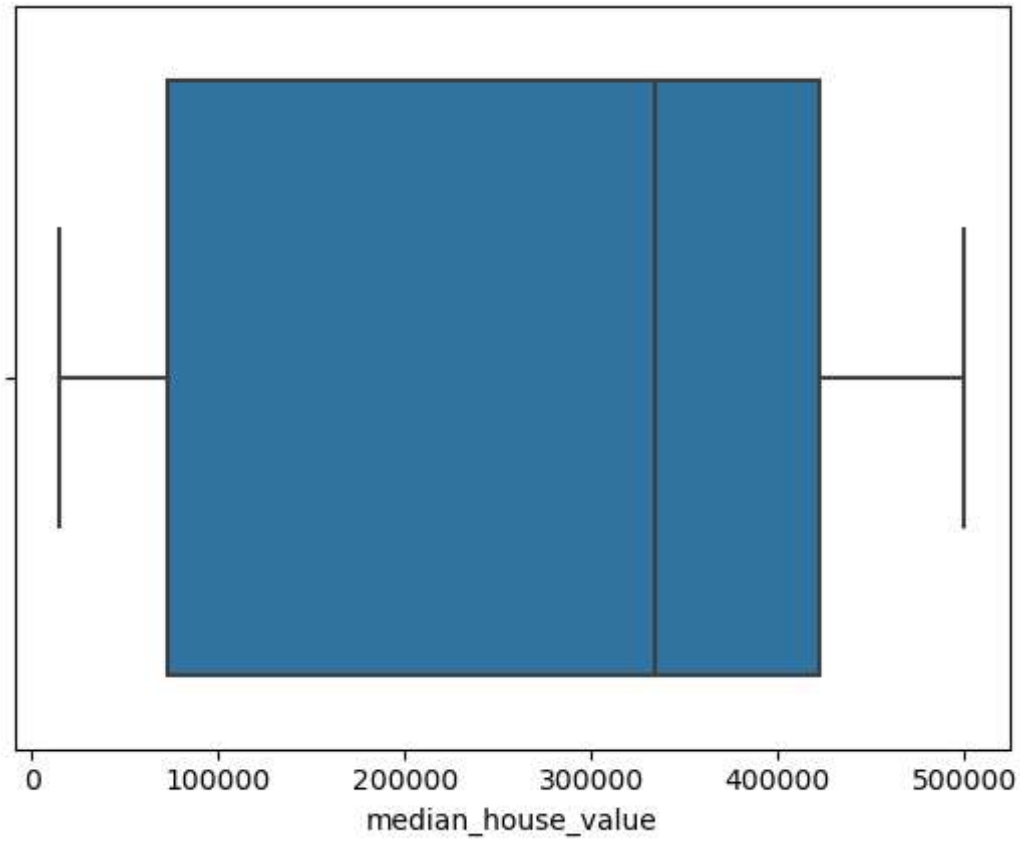
```
before removing outliers 20640
after removing outliers 309
new outliers 20331
```



In [22]:

```
for col in numerical_features[8:9]:
    Q1 = np.percentile(df[col], 40,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df[col], 60,
                       interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR

    lower = Q1 - 1.5*IQR
    upper, lower
    new_df = df.loc[(df[col] > upper) | (df[col] < lower)]
    print(f'before removing outliers {len(df)}')
    print(f'after removing outliers {len(new_df)}')
    print('new outliers', len(df) - len(new_df))
    sns.boxplot(new_df[col])
    plt.show()
```

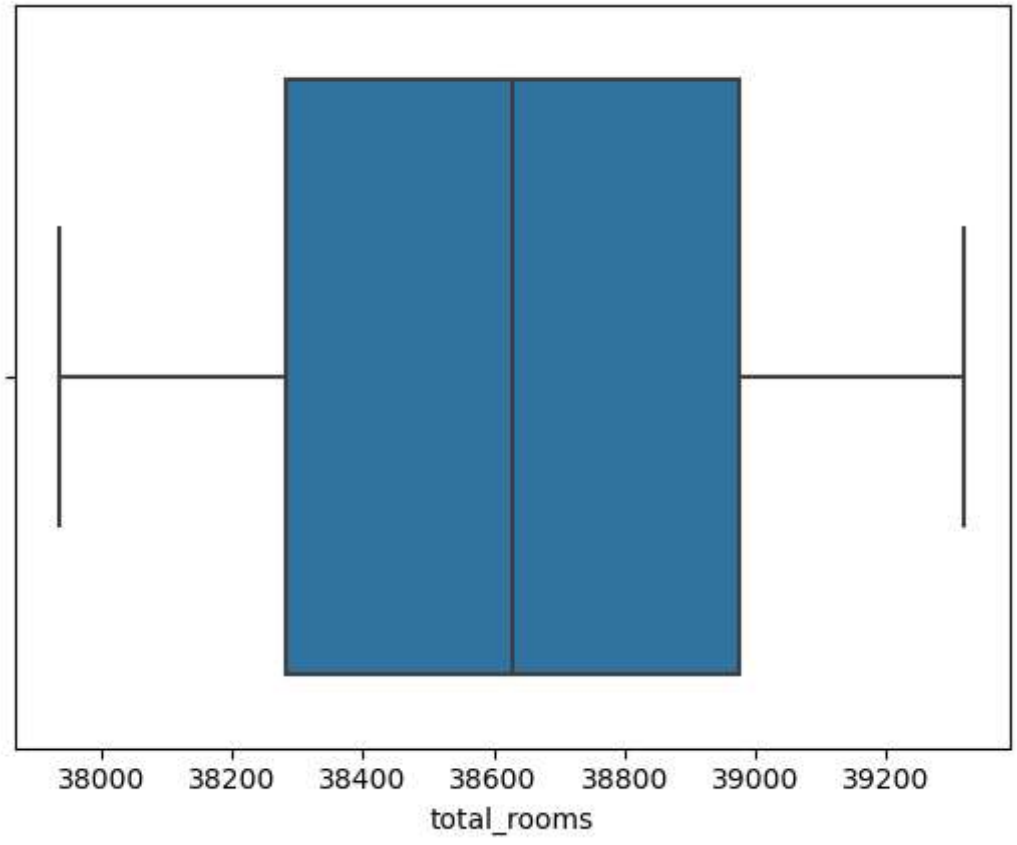
```
before removing outliers 20640
after removing outliers 6074
new outliers 14566
```



In [23]:

```
for col in numerical_features[3:4]:
    Q1 = np.percentile(df[col], .5,
                       interpolation = 'midpoint')

    Q3 = np.percentile(df[col], 99.5,
                       interpolation = 'midpoint')
    IQR = Q3 - Q1
    upper = Q3 +1.5*IQR

    lower = Q1 - 1.5*IQR
    upper, lower
    new_df = df.loc[(df[col] > upper) | (df[col] < lower)]
    print(f'before removing outliers {len(df)}')
    print(f'after removing outliers {len(new_df)}')
    print('new outliers', len(df) - len(new_df))
    sns.boxplot(new_df[col])
    plt.show()
```

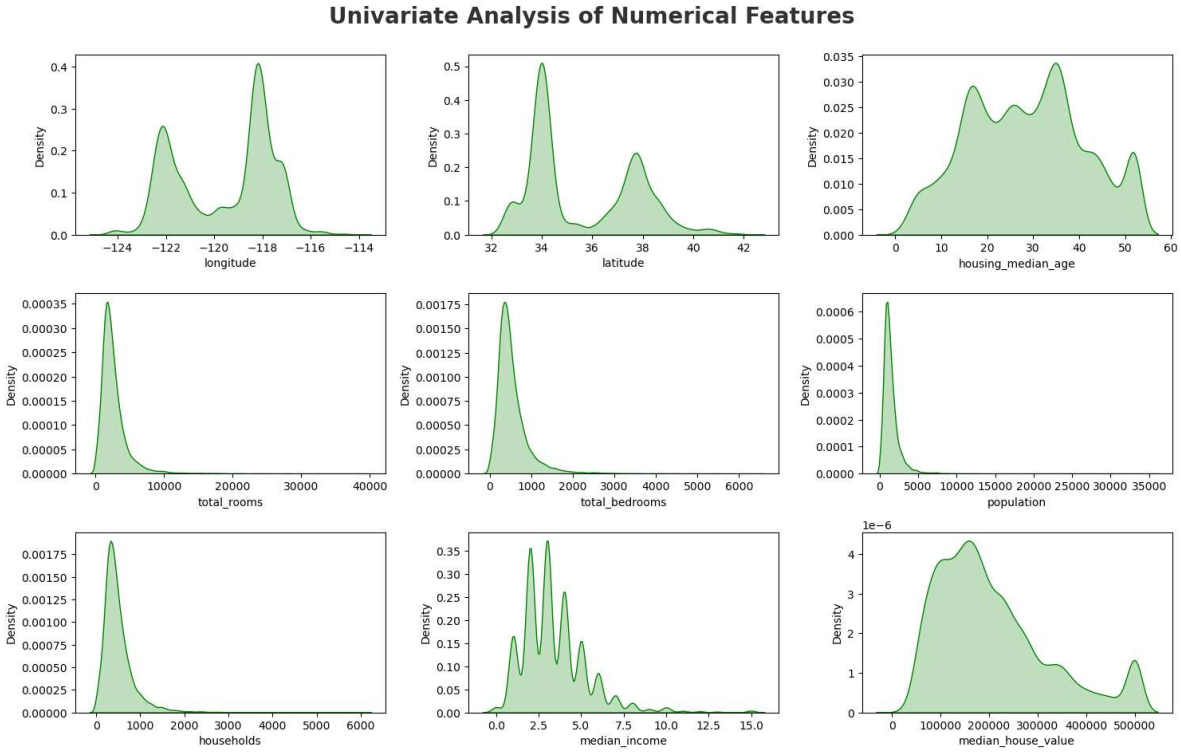
```
before removing outliers 20640
after removing outliers 2
new outliers 20638
```



2.3 transformation of skewed data distribution

In [24]:

```
# comment - separate distribution of each and every numerical features
# observation - the index of numerical features from 3 to 7 are right skewed also known as
              # the index 0,1 are bimodal distributed and 2 is multimodal disrtibuted
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight='bold', a

for i in range(0, len(numerical_features)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[numerical_features[i]],shade=True, color='g')
    plt.xlabel = (numerical_features[i])
    plt.tight_layout()
```



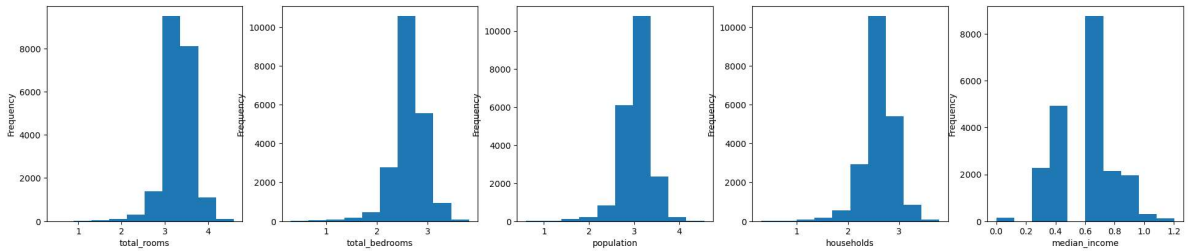
we need to transform the right skewed data into log transformation for normality of data.

In [25]:

```
# comment - selecting list of features which need to log transformed.
log_var = ['total_rooms', 'total_bedrooms', 'population', 'households', 'median_income']
```

In [26]:

```
# comment - log transformation of selected skewed data features.
# observation - the data of these features are now normally distributed.
fig = plt.figure(figsize = (24,10))
for j in range(len(log_var)):
    var = log_var[j]
    transformed = 'log_' + var
    df[transformed] = np.log10(df[var] + 1)
    sub = fig.add_subplot(2,5, j+1)
    sub.set_xlabel(var)
    df[transformed].plot(kind = 'hist')
```



2.4 scaling

importing libraries for scaling data.

In [27]:

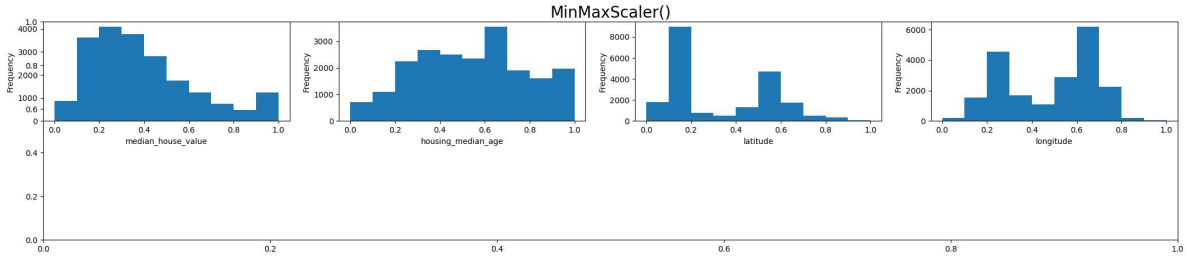
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

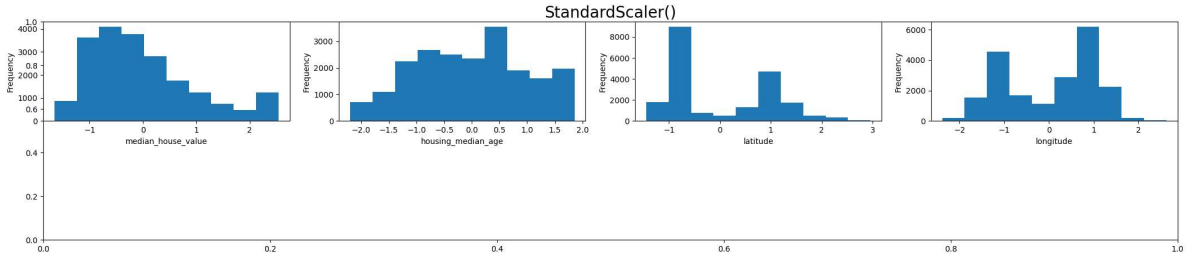
In [28]:

```
def log_features(scale_var):
    scale_var = ['median_house_value', 'housing_median_age', 'latitude', 'longitude']
    scalers_list = [MinMaxScaler(), StandardScaler()]
    for i in range(len(scalers_list)):
        scaler = scalers_list[i]
        fig = plt.figure(figsize = (26,5))
        plt.title(scaler, fontsize = 20)
        for j in range(len(scale_var)):
            var = scale_var[j]
            scaled_var = 'scaled_' + var
            model = scaler.fit(df[var].values.reshape(-1,1))
            df[scaled_var] = model.transform(df[var].values.reshape(-1,1))

            sub = fig.add_subplot(2,4, j+1)
            sub.set_xlabel(var)
            df[scaled_var].plot(kind = 'hist')
    return(log_features('median_house_value', 'housing_median_age', 'latitude', 'longitude'
```

In [29]:

```
# comment - we have used two scaling libraries MinMaxScaler has [0-1] scale.
# observation - by scaling we have transformed the features of remaining datatypes into nor
scale_var = ['median_house_value', 'housing_median_age', 'latitude', 'longitude']
scalers_list = [MinMaxScaler(), StandardScaler()]
for i in range(len(scalers_list)):
    scaler = scalers_list[i]
    fig = plt.figure(figsize = (26,5))
    plt.title(scaler, fontsize = 20)
    for j in range(len(scale_var)):
        var = scale_var[j]
        scaled_var = 'scaled_' + var
        model = scaler.fit(df[var].values.reshape(-1,1))
        df[scaled_var] = model.transform(df[var].values.reshape(-1,1))

        sub = fig.add_subplot(2,4, j+1)
        sub.set_xlabel(var)
        df[scaled_var].plot(kind = 'hist')
```

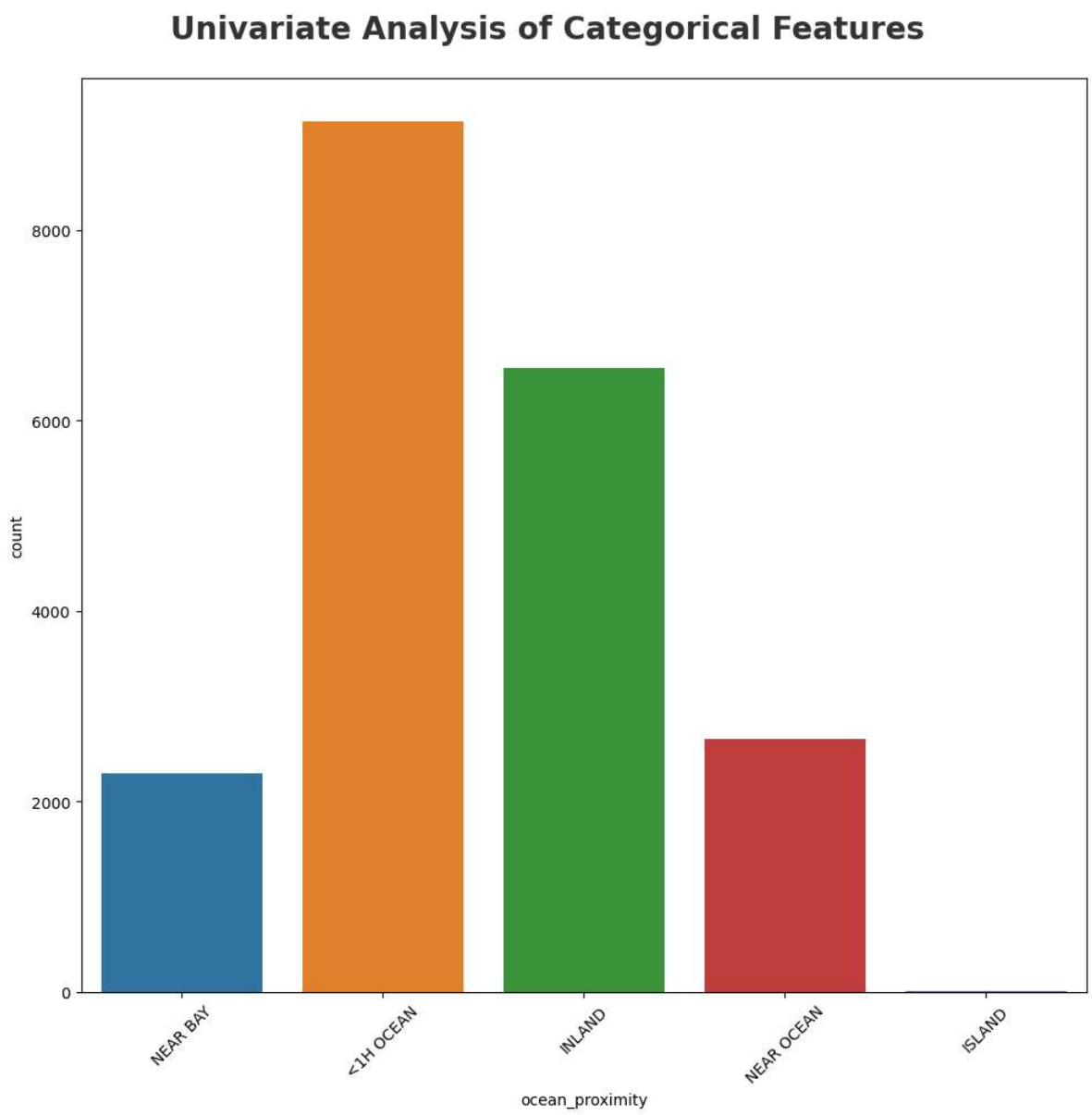




2.5 imbalance data

In [30]:

```
# comment - checking where the location of houses is highest near the ocean_proximity
# observation - the analysis shows that population or households prefers <1H ocean location
                # second preference of people of California will be INLAND.
                # less to NEAR OCEAN and than NEAR BAY respectively
                # shows the data is imbalanced as island as less or nrgligible preferences.
plt.figure(figsize=(10, 10))
plt.suptitle('Univariate Analysis of Categorical Features', fontsize=20, fontweight='bold',

for i in range(0, len(categorical_features)):
    plt.subplot(1, 1, i+1)
    sns.countplot(x = df[categorical_features[i]])
    plt.xticks(rotation=45)
    plt.xlabel = (categorical_features[i])
    plt.tight_layout()
```



creating dummies for having the imbalanced data to balanced data by giving equal numeric data presentation

In [31]:

```
# comment - one hot-encode all categorical features
ohe = pd.get_dummies(df[categorical_features].ocean_proximity)
```

In [32]:

```
ohe
```

Out[32]:

	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
0	0	0	0	1	0
1	0	0	0	1	0
2	0	0	0	1	0
3	0	0	0	1	0
4	0	0	0	1	0
...	...	...	...	...	...
20635	0	1	0	0	0
20636	0	1	0	0	0
20637	0	1	0	0	0
20638	0	1	0	0	0
20639	0	1	0	0	0

20640 rows × 5 columns

2.7 handling multicollinearity

In [33]:

```
# comment - chi square test shows the relation between features and here we are checking fo
# observation - for index 0,1,2 and 8 we are able to reject H0 (which is good) as independe
                ## for index 3 to 7 we fail to reject as these factors might affect the rel
chi2_test = []
for feature in numerical_features:
    if chi2_contingency(pd.crosstab(df['ocean_proximity'], df[feature]))[1] < 0.05:
        chi2_test.append('Reject Null Hypothesis')
    else:
        chi2_test.append('Fail to Reject Null Hypothesis')
result = pd.DataFrame(data=[numerical_features, chi2_test]).T
result.columns = ['Column', 'Hypothesis Result']
result
```

Out[33]:

	Column	Hypothesis Result
0	longitude	Reject Null Hypothesis
1	latitude	Reject Null Hypothesis
2	housing_median_age	Reject Null Hypothesis
3	total_rooms	Fail to Reject Null Hypothesis
4	total_bedrooms	Fail to Reject Null Hypothesis
5	population	Fail to Reject Null Hypothesis
6	households	Fail to Reject Null Hypothesis
7	median_income	Reject Null Hypothesis
8	median_house_value	Reject Null Hypothesis

In [34]:

```
# comment - data has mulicollinaerity this means that some features are highly correlated t
# observation - we will handle by VIF and
                    ## if VIF > 5 means multicollinearity and need to be removed.
                    ## if VIF < 5 than no high correlation and good for the data.
# load statmodels functions
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

# compute the vif for all given features
def compute_vif(considered_features):

    X = df[considered_features]
    # the calculation of variance inflation requires a constant
    X['intercept'] = 1

    # create dataframe to store vif values
    vif = pd.DataFrame()
    vif["Variable"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i)
    for i in range(X.shape[1])]
    vif = vif[vif['Variable']!='intercept']
    return vif
```

In [35]:

```
# comment - the feature index from 3 to 6(including) have failed the test and shows that hi
# observation - households has high correlation and this means that we need to remove it fo
considered_features = ['total_rooms','total_bedrooms','population','households','median_inc
# compute vif
compute_vif(considered_features).sort_values('VIF', ascending=False)
```

Out[35]:

	Variable	VIF
3	households	27.005610
1	total_bedrooms	26.382505
0	total_rooms	10.972795
2	population	6.107050
4	median_income	1.446764

In [36]:

```
# comment - removed the housedhold feature
# observation - now total_rooms has high multicollinearity and need to be removed.

# compute vif values after removing a feature
considered_features.remove('households')
compute_vif(considered_features)
```

Out[36]:

	Variable	VIF
0	total_rooms	10.971980
1	total_bedrooms	10.606719
2	population	4.618403
3	median_income	1.439523

In [37]:

```
# comment - compute vif values after removing another feature total_rooms
# observation - now the multicollinearity is all under 5 and hence all errors are removed a
considered_features.remove('total_rooms')
compute_vif(considered_features)
```

Out[37]:

	Variable	VIF
0	total_bedrooms	4.223097
1	population	4.222945
2	median_income	1.000631

3. problem statement and conditional problems

3.1 answering various observation via equation and graphs

1. Where the maximum population have houses near the ocean location ?

In [38]:

```
# observation - max population resides or prefers ocean location as <1H OCEAN.
df[df['population'] == max(df['population'])]['ocean_proximity']
```

Out[38]:

```
15360    <1H OCEAN
Name: ocean_proximity, dtype: object
```

2. Among population and households who has the maximum median income ?

In [39]:

```
# observation - max income population earns is 2 ten thousands USD.
df[df['population'] == max(df['population'])]['median_income']
```

Out[39]:

```
15360    2
Name: median_income, dtype: int32
```

In [40]:

```
# observation - same result as population are households residing in a block and earns max
df[df['households'] == max(df['households'])]['median_income']
```

Out[40]:

```
9880    2
Name: median_income, dtype: int32
```

In [41]:

```
# observation - towards north population more resides in a block.
df[df['population'] == max(df['population'])]['latitude']
```

Out[41]:

```
15360    33.35
Name: latitude, dtype: float64
```

3. Which location around ocean has highest median house value and where the house is costlier?

In [42]:

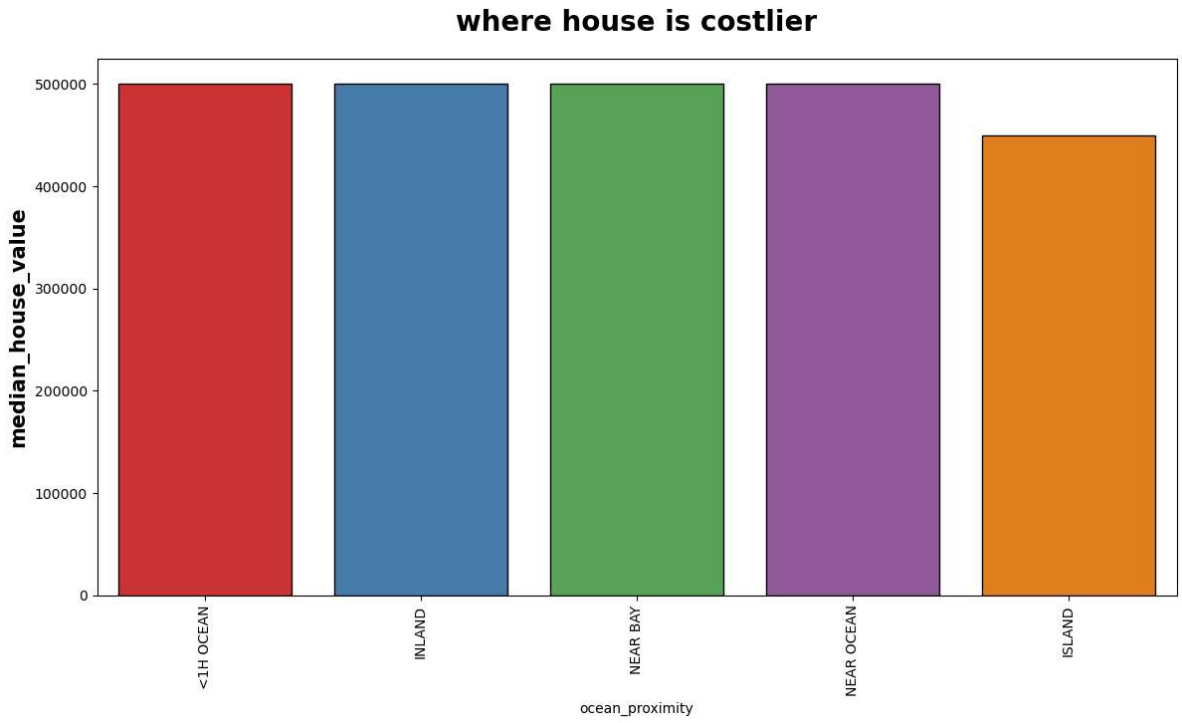
```
# observation - Except ISLAND the houses at all other locations has almost same cost.
                ## at first sight we might think that ISLAND has cheaper house, no, becau
                     ## population prefers less this location and still this price is hig
house= df.groupby('ocean_proximity').median_house_value.max()
house=house.to_frame().sort_values('median_house_value',ascending=False)
house
```

Out[42]:

ocean_proximity	median_house_value
<1H OCEAN	500001
INLAND	500001
NEAR BAY	500001
NEAR OCEAN	500001
ISLAND	450000

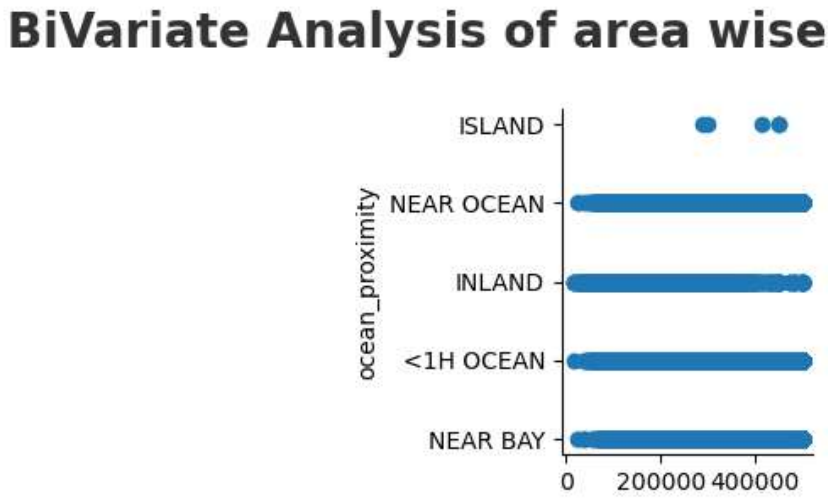
In [43]:

```
##  observation - graphical analysis for better understanding.
plt.subplots(figsize=(14,7))
sns.barplot(x=house.index, y= house.median_house_value,ec = "black",palette="Set1")
plt.title("where house is costlier", weight="bold",fontsize=20, pad=20)
plt.ylabel("median_house_value", weight="bold", fontsize=15)#plt.xlabel("ocean_proximity",
plt.xticks(rotation=90)
plt.show()
```



In [44]:

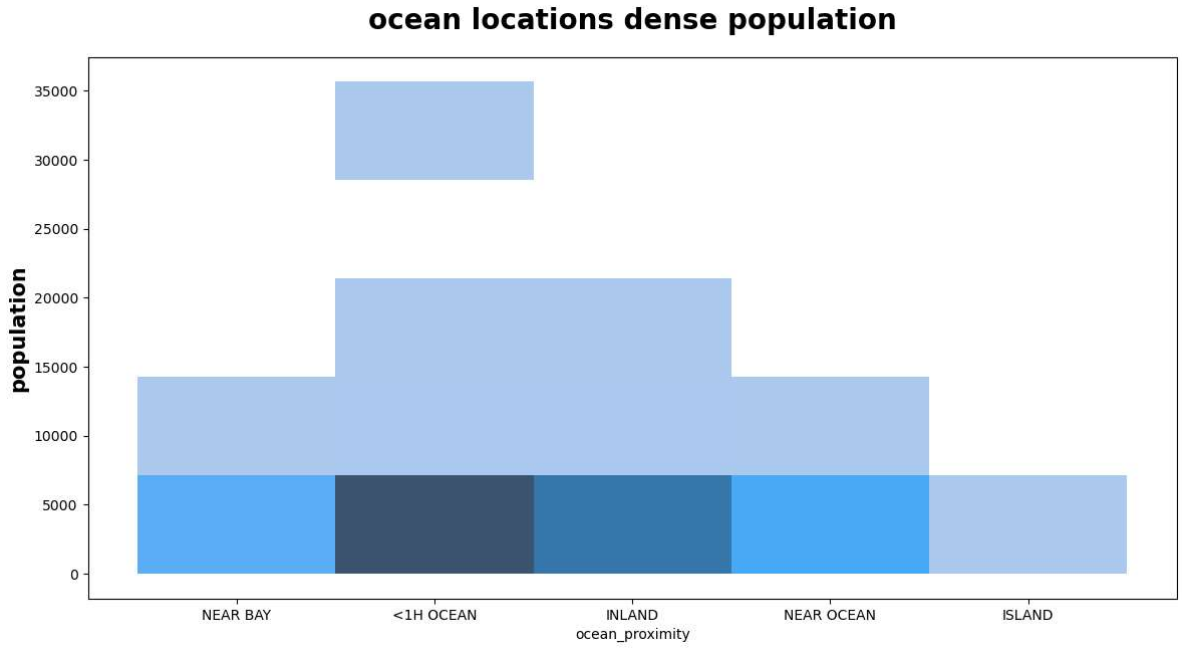
```
# observation - the ISLAND has highest house value ranges and all other have all types of r
sns.FacetGrid(df[numerical_features]).map(plt.scatter, x = df['median_house_value'], y = df
plt.suptitle('BiVariate Analysis of area wise house value', fontsize=20, fontweight='bold',
#plt.xlabel ('median_house_value')
plt.ylabel ('ocean_proximity')
plt.tight_layout()
```



4. Where the population is dense near ocean locations ?

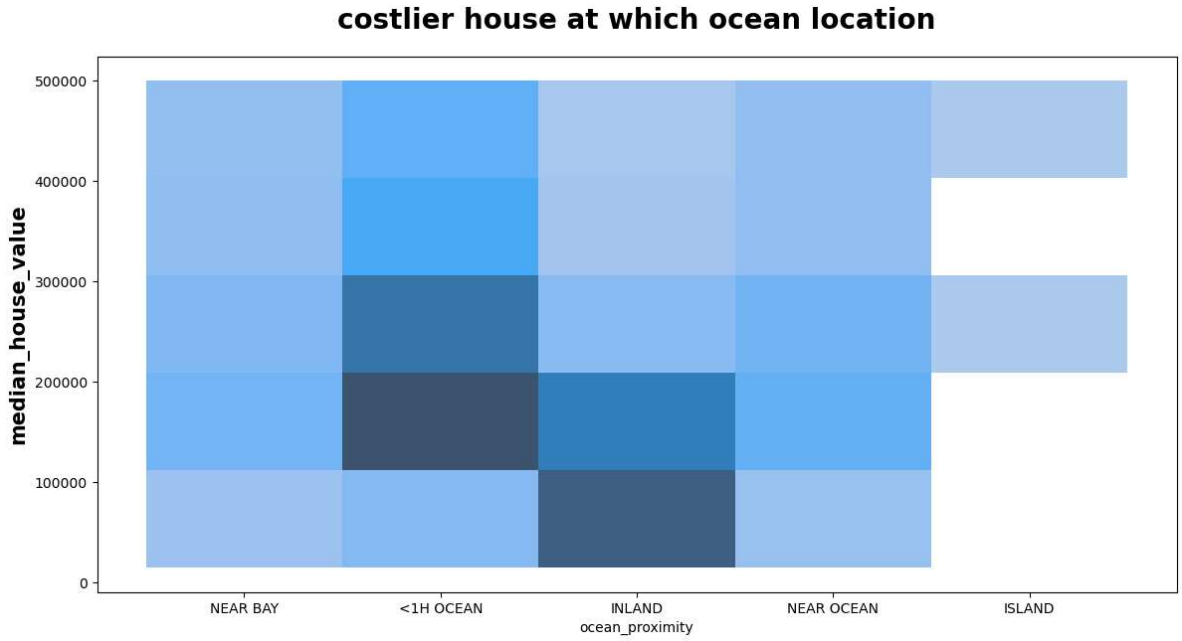
In [45]:

```
# observation - the population is densely residing more in the block of ocean location (tha
plt.subplots(figsize=(14,7))
sns.histplot(x='ocean_proximity', y='population', bins = 5,  data=df ,palette="Set1_r")
plt.title("ocean locations dense population", weight="bold",fontsize=20, pad=20)
plt.ylabel("population", weight="bold", fontsize=15)
#plt.xlabel("ocean_proximity", weight="bold", fontsize=12)
plt.show()
```



In [46]:

```
plt.subplots(figsize=(14,7))
sns.histplot(x='ocean_proximity', y='median_house_value', bins = 5, data=df ,palette="Set1_
plt.title("costlier house at which ocean location", weight="bold",fontsize=20, pad=20)
plt.ylabel("median_house_value", weight="bold", fontsize=15)
#plt.xlabel("ocean_proximity", weight="bold", fontsize=12)
plt.show()
```



5. How many peope have less than 10 (thousand USD) and still can afford the houses and where?

In [47]:

```
# observation - out of 20640 total 20331 population has less than 10 thousand USD.
df[(df['median_income']< 10)]
```

Out[47]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
0	-122.23	37.88	41	880	129	322	
1	-122.22	37.86	21	7099	1106	2401	
2	-122.24	37.85	52	1467	190	496	
3	-122.25	37.85	52	1274	235	558	
4	-122.25	37.85	52	1627	280	565	
...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374	845	
20636	-121.21	39.49	18	697	150	356	
20637	-121.22	39.43	17	2254	485	1007	
20638	-121.32	39.43	18	1860	409	741	
20639	-121.24	39.37	16	2785	616	1387	

20331 rows × 19 columns

In [48]:

```
df[(df['median_income'] < 10) & (df['median_house_value']< 20000)]
```

Out[48]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	house
2521	-122.74	39.71	16	255	73	85	
2799	-117.02	36.40	19	619	239	490	
5887	-118.33	34.15	39	493	168	259	
9188	-117.86	34.24	52	803	267	628	
19802	-123.17	40.31	36	98	28	18	

6. How population manages to purchase houses even at less income than house value ?

In [49]:

```
# observation - we can estimate new column from the data about other_income_source feature.
df['other_income_source'] = df['median_house_value'] - df['median_income']
```

In [50]:

```
# observation - datatype chnanged to int.
df['other_income_source'] = df['other_income_source'].astype('int')
```

In [51]:

```
df[['other_income_source']]
```

Out[51]:

	other_income_source
0	452592
1	358492
2	352093
3	341295
4	342197
...	...
20635	78099
20636	77098
20637	92299
20638	84699
20639	89398

20640 rows × 1 columns

In [52]:

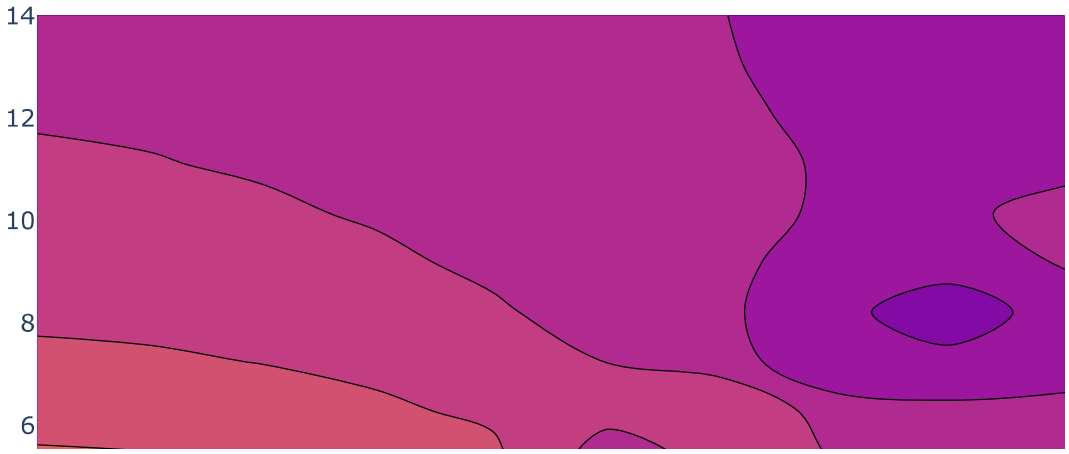
```
# comment - checking how this new is correlated to other features
# observation - obvious it will show the high positive correlation with median_house_value
corr_matrix = df.corr()
corr_matrix['other_income_source'].sort_values(ascending = False)
```

Out[52]:

```
other_income_source          1.000000
median_house_value           1.000000
scaled_median_house_value    1.000000
median_income                0.678393
log_median_income            0.645237
log_total_rooms              0.159420
total_rooms                  0.134151
housing_median_age           0.105627
scaled_housing_median_age    0.105627
log_households               0.073613
households                   0.065843
log_total_bedrooms           0.053060
total_bedrooms               0.049458
log_population              -0.021205
population                  -0.024650
scaled_longitude            -0.045967
longitude                   -0.045967
latitude                    -0.144161
scaled_latitude             -0.144161
Name: other_income_source, dtype: float64
```

In [53]:

```
# contour plot for seeing how the z variable is at each x and y variables by just moving ho
import plotly.graph_objects as go

fig = go.Figure(data =
    go.Contour(
        z= ((df['median_house_value'])- (df['median_income'])),
        x= np.linspace(14999, 500000, 500001),
        y = np.linspace(0.4999, 14, 15) # vertical axis
    ))
fig.show()
```



7. How many total rooms and bedrooms are in the blocks have under the every house value

In [54]:

```
# observation - the data shows the count of rooms and bedrooms in a block of that location.
                    ## so, we can see that at the highest house value (last index) has high
                              ### number of rooms and bedrooms, nearly positive realtion rel
df.groupby('median_house_value')['total_rooms','total_bedrooms'].count()
```

Out[54]:

median_house_value	total_rooms	total_bedrooms
14999	4	4
17500	1	1
22500	4	4
25000	1	1
26600	1	1
...	...	...
498800	1	1
499000	1	1
499100	1	1
500000	27	27
500001	965	965

3842 rows × 2 columns

8. Analysis of ecah population that where they reside more north or west along with how much they earn at that location?

In [55]:

```
# observation - latitude means north location and longitude is west location
   ## this shows that first index of below data shows less earning opportunity and that's w
     ## location plays important role along with population residing over there.
df.groupby('population')['longitude','latitude','median_income',].sum()
```

Out[55]:

population	longitude	latitude	median_income
3	-118.44	34.04	0
5	-114.62	33.62	0
6	-117.79	35.21	2
8	-473.89	139.13	6
9	-237.61	71.89	1
...	...	...	...
15507	-117.78	34.03	6
16122	-117.74	33.89	7
16305	-121.44	38.43	4
28566	-121.79	36.64	2
35682	-117.42	33.35	2

3888 rows × 3 columns

In [56]:

```
df.shape
```

Out[56]:

```
(20640, 20)
```

In [57]:

```
df.head()
```

Out[57]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
0	-122.23	37.88	41	880	129	322	12
1	-122.22	37.86	21	7099	1106	2401	113
2	-122.24	37.85	52	1467	190	496	17
3	-122.25	37.85	52	1274	235	558	21
4	-122.25	37.85	52	1627	280	565	25

storing our new and transformed dataset

droping the repeated and non-transformed data features

In [58]:

```
data_cleaned = df.drop(columns = ['longitude','latitude','housing_median_age','total_rooms'
```

In [59]:

```
data_cleaned
```

Out[59]:

	ocean_proximity	log_total_rooms	log_total_bedrooms	log_population	log_households
0	NEAR BAY	2.944976	2.113943	2.509203	2.103804
1	NEAR BAY	3.851258	3.044148	3.380573	3.056524
2	NEAR BAY	3.166726	2.281033	2.696356	2.250420
3	NEAR BAY	3.105510	2.372912	2.747412	2.342423
4	NEAR BAY	3.211654	2.448706	2.752816	2.414973
...	...	...	...	...	...
20635	INLAND	3.221675	2.574031	2.927370	2.519828
20636	INLAND	2.843855	2.178977	2.552668	2.060698
20637	INLAND	3.353147	2.686636	3.003461	2.637490
20638	INLAND	3.269746	2.612784	2.870404	2.544068
20639	INLAND	3.444981	2.790285	3.142389	2.725095

20640 rows × 11 columns

In [60]:

```
data_cleaned.shape
```

Out[60]:

```
(20640, 11)
```

our target variable is scaled_median_house_value

why? - as we have to see the value or house is dependent on which of the other features from dataset.

y is denoted for dependent variable and x is for independent variable.

In [61]:

```
### y is always in a series format
y = data_cleaned['scaled_median_house_value']
```

In [62]:

```
### x is in the form of DataFrame
X = data_cleaned[['log_total_rooms','log_total_bedrooms','log_population','log_households',
```

Model Training

importing library for training our dataset

In [63]:

```
from sklearn.model_selection import train_test_split
```

inbuilt function for training and testing the x and y variables

In [64]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

checking the shape of x and y variables , seeing how much data is collected and being test and trained

In [65]:

```
X_train.shape
```

Out[65]:

```
(13828, 9)
```

In [66]:

```
X_test.shape
```

Out[66]:

```
(6812, 9)
```

In [67]:

```
y_train.shape
```

Out[67]:

```
(13828,)
```

In [68]:

```
y_test.shape
```

Out[68]:

```
(6812,)
```

importing the standardscaler for transforming the train and test dataset.

```
### we donot perfom this transformation for y and error data
```

In [69]:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
```

In [70]:

```
X_train=scaler.fit_transform(X_train)
```

In [71]:

```
X_test=scaler.transform(X_test)
```

Model Testing

importing various models:

1. linear regression

2. ridge

3. lasso

4. elasticnet

In [72]:

```
from sklearn.linear_model import LinearRegression
```

assumptions of linear regression

1. linearity of y_test and reg_pred

2. uniform range of graph

3. normality for residuals

In [73]:

```
regression=LinearRegression()
```

we are fitting the data here only no execution of transformation

In [74]:

```
regression.fit(X_train,y_train)
```

Out[74]:

```
LinearRegression
LinearRegression()
```

checking the linear regression model analysis of y variable with x variables.

In [75]:

```
coeff_df = pd.DataFrame(regression.coef_,X.columns, columns = ['coefficients'])
coeff_df
```

Out[75]:

	coefficients
log_total_rooms	9.215659e-06
log_total_bedrooms	-1.081731e-05
log_population	5.571461e-06
log_households	-6.413457e-06
log_median_income	7.396618e-05
scaled_housing_median_age	-3.466561e-07
scaled_latitude	1.586218e-06
scaled_longitude	1.537459e-06
other_income_source	8.666076e-06

In [76]:

```
### we have total 9 x variables and the coefficient value is calculated
    ## log_total_rooms, log_population, log_median_income and other_income_source has posit
        ## log_total_bedrooms , log_households, sacled_housing_median_age has negative slop
            ### sacled_latitude and scaled_longitude has positive but small impact on log_

print(regression.coef_)
```

```
[ 9.21565926e-06 -1.08173090e-05  5.57146080e-06 -6.41345697e-06
  7.39661790e-05 -3.46656102e-07  1.58621769e-06  1.53745888e-06
  8.66607605e-06]
```

In [77]:

```
### when the all slopes value are zero the max value the model can take is told by intercep
print(regression.intercept_)
```

```
-1.792645987720523
```

the data above we obtained are from actual or truth data

model prediction value

In [78]:

```
reg_pred=regression.predict(X_test)
reg_pred
```

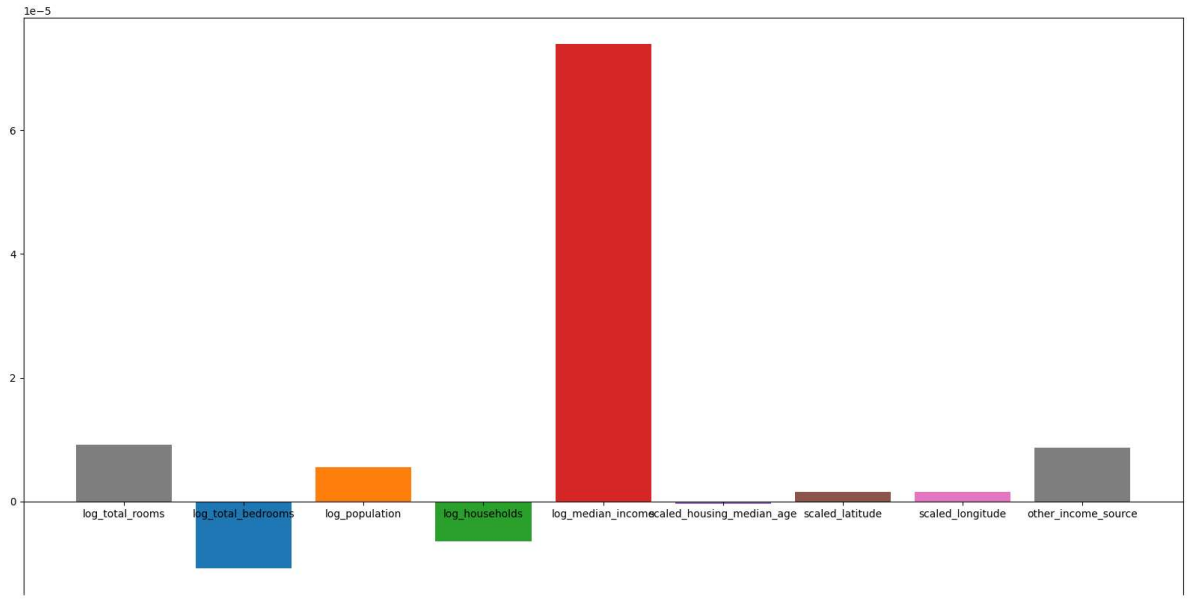
Out[78]:

```
array([-1.37925786, -1.3957186 ,  2.5404207 , ..., -0.8653527 ,
       -0.65996861,  0.86872374])
```

relation of independent variables with dependent variables via linear regression

In [79]:

```
### plotting regression coefficient showing relation between dependent and independent vari
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan',
'tab:orange', 'tab:green', 'tab:blue', 'tab:olive']

ax.bar(X.columns,
coeff_df['coefficients'],
color = color)

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```



linearity of y_test (truth data) feature with predicted data is exact staright line showing:

1. our best fit line has acurately minimize errors

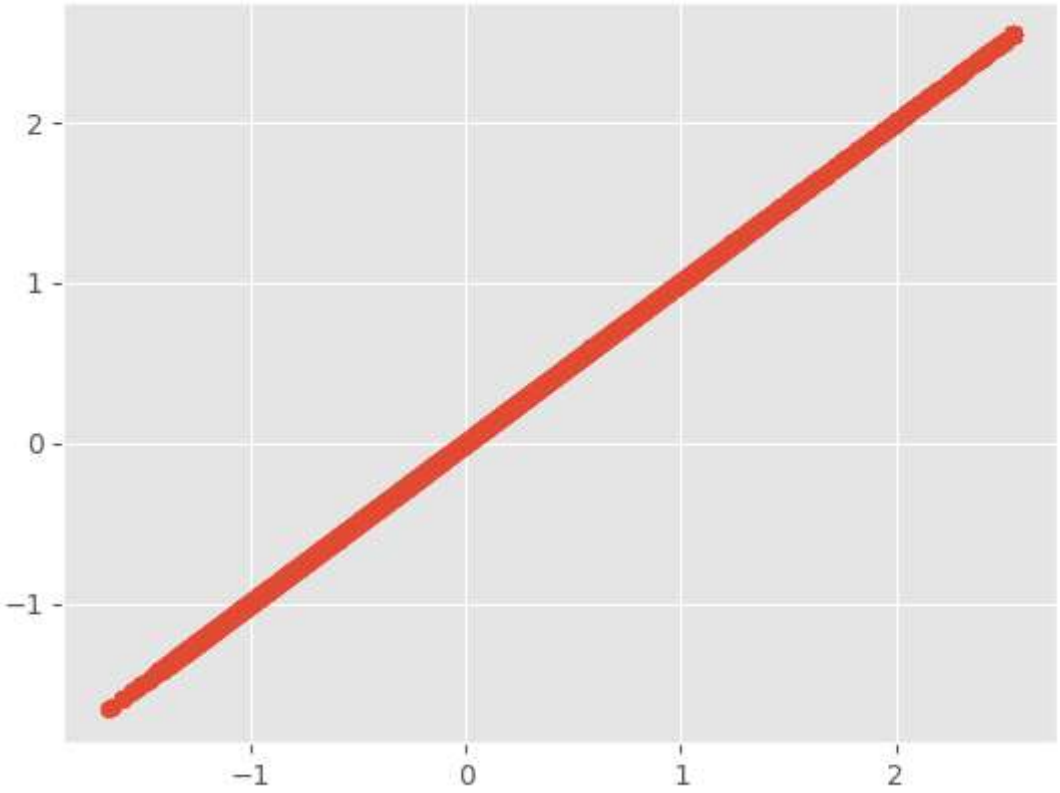
2. the actual and predicted data showing good accuracy of the model.

In [80]:

```
plt.scatter(y_test,reg_pred)
```

Out[80]:

```
<matplotlib.collections.PathCollection at 0x2afe7c4b6d0>
```



In [81]:

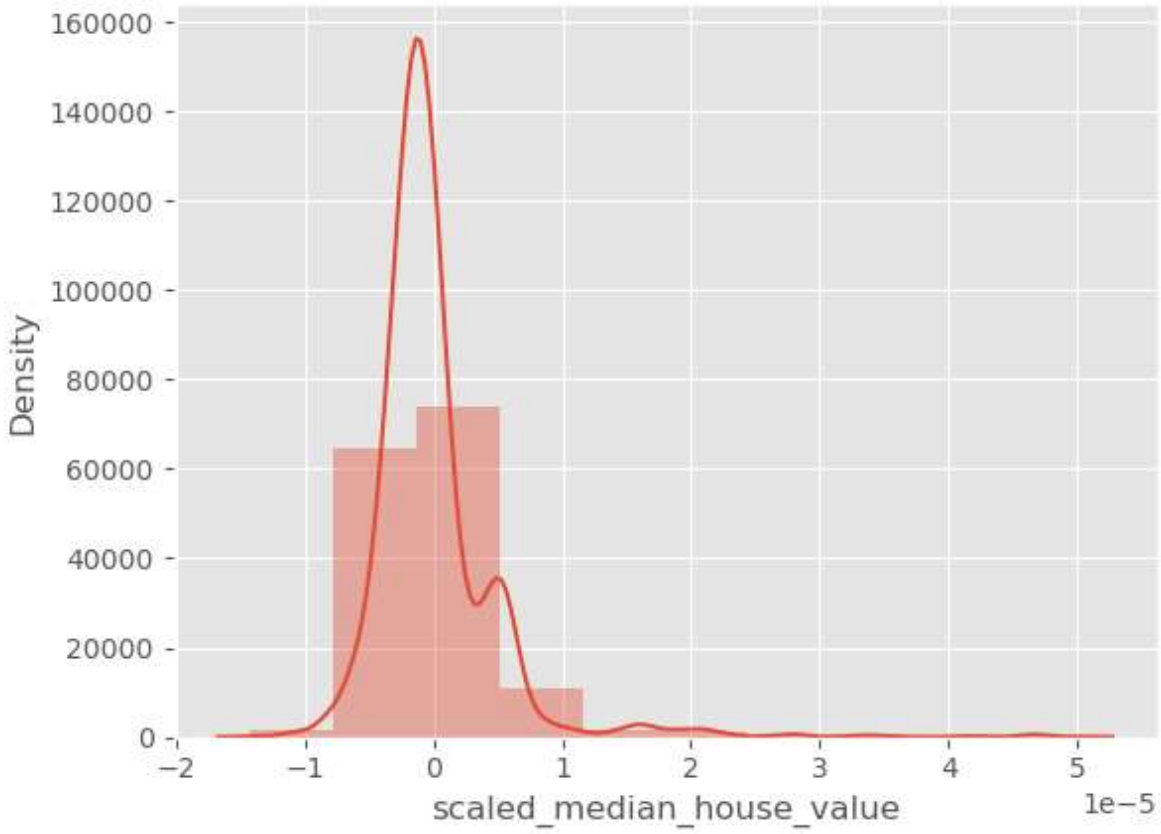
```
### residuals means errors in the model
   ### they aries due to actual data and predicted data test.
```

In [82]:

```
### it is somewhat normally distributed but still has some right skewness.

residulas = y_test - reg_pred
sns.distplot(residulas, bins = 10)
```

Out[82]:

```
<AxesSubplot:xlabel='scaled_median_house_value', ylabel='Density'>
```

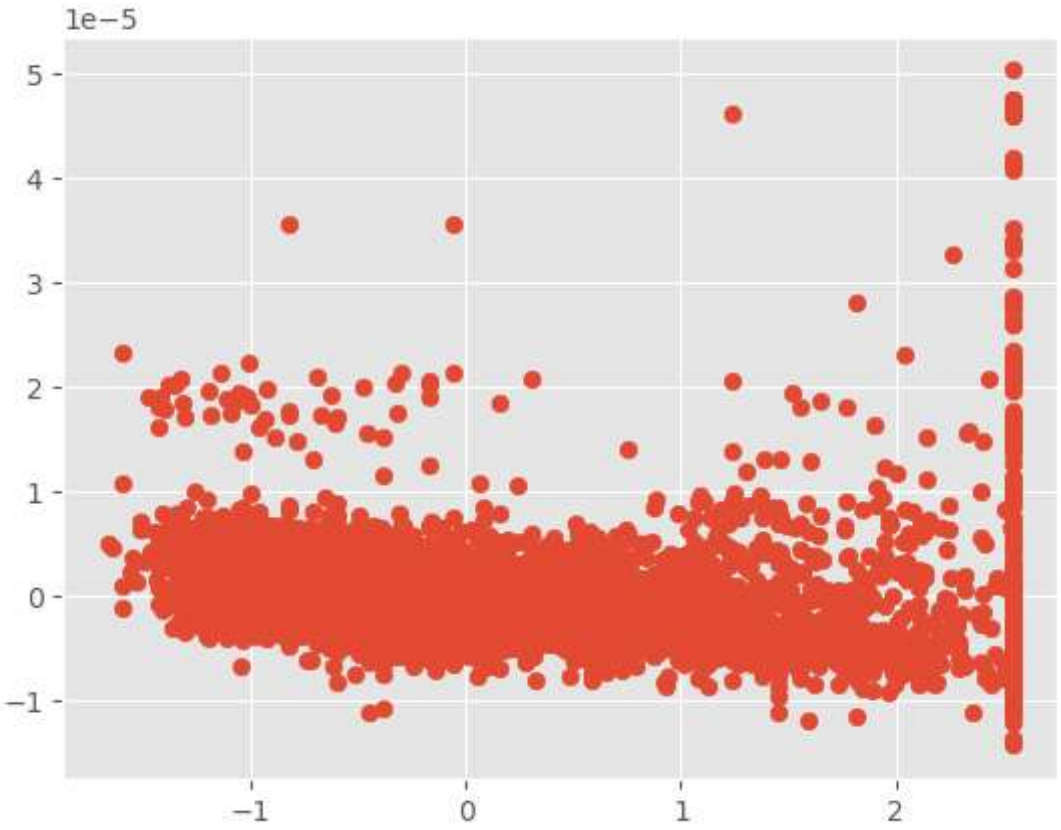


In [83]:

```
### uniformity range from 1 to 2 as most of the data lies here only
plt.scatter(reg_pred , residulas)
```

Out[83]:

```
<matplotlib.collections.PathCollection at 0x2afe7cf2850>
```



we have used all types of cost functions

1. MSE - as our model have outliers so we don't use it much.

2. MAE - robust to outliers but time consuming

3. RMSE - same not robust to outliers.

according to our model, we shall use MAE.

In [84]:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
print(mean_squared_error(y_test, reg_pred))
print(mean_absolute_error(y_test, reg_pred))
print(np.sqrt(mean_squared_error(y_test, reg_pred)))
```

```
2.3801195588358396e-11
2.9852313282494656e-06
4.878646901381406e-06
```

performance matrix

R-square and Adjusted R-square

R-square

In [85]:

```
### our model has very high accuracy by reducing a lot of errors
from sklearn.metrics import r2_score
score=r2_score(y_test,reg_pred)
print(score)
```

```
0.9999999999762174
```

Adjusted R-square

In [86]:

```
### this shows that whether we include any feature which is not much useful in the model.
 ### here, this shows that all the features in the model are useful showing huge significan
adj_R_score = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
adj_R_score
```

Out[86]:

```
0.9999999999761859
```

importing library for checking via Ridge Regression

we use ridge when the model is overfitting

but our model is best fitted hence, ridge does not help us much.

In [87]:

```
from sklearn.linear_model import Ridge
ridge=Ridge()
```

Train the model

In [88]:

```
ridge.fit(X_train,y_train)
```

Out[88]:

```
Ridge
Ridge()
```

In [89]:

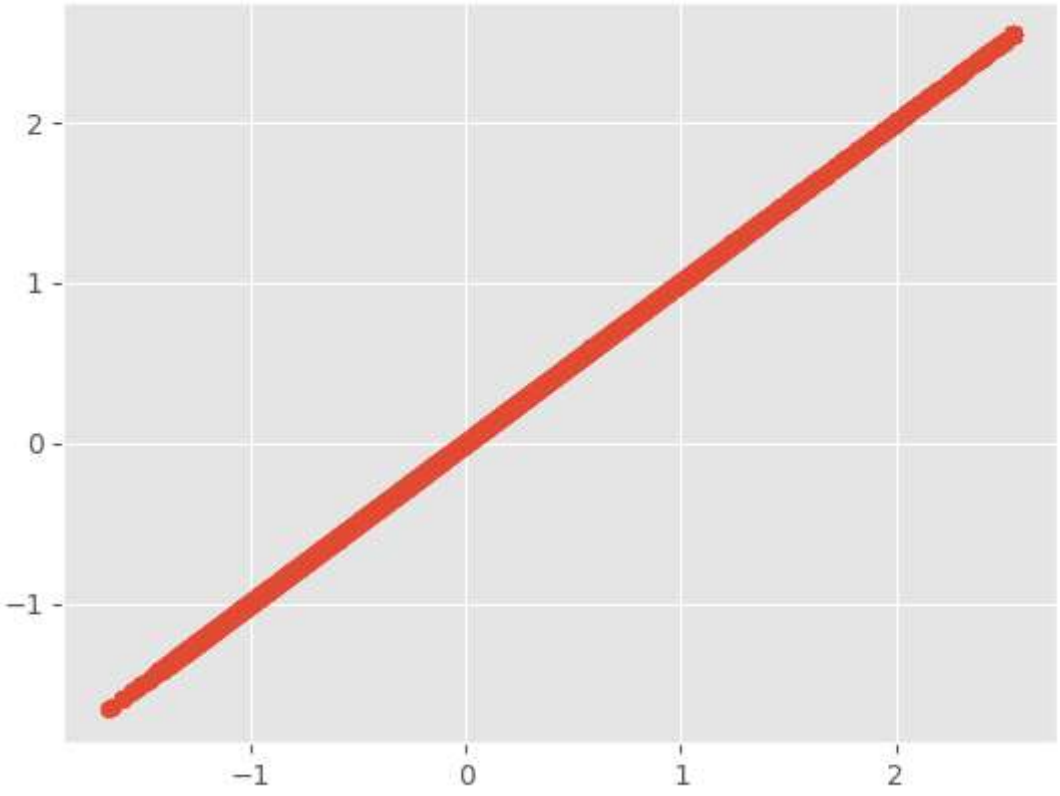
```
pred = ridge.predict(X_test)
```

In [90]:

```
# the model is accuarate in training and testing
plt.scatter(y_test, pred)
```

Out[90]:

```
<matplotlib.collections.PathCollection at 0x2afe7a5f490>
```



In [91]:

```
### ridge does not delete feature by making it 0, from the model directly.
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
# Calculate Mean Squared Error, mean absolute error and RMSE
print(mean_squared_error(y_test, pred))
print(mean_absolute_error(y_test, pred))
print(np.sqrt(mean_squared_error(y_test, pred)))

ridge_coefficient = pd.DataFrame()
ridge_coefficient["Columns"]= X_train.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridge.coef_)
print(ridge_coefficient)
```

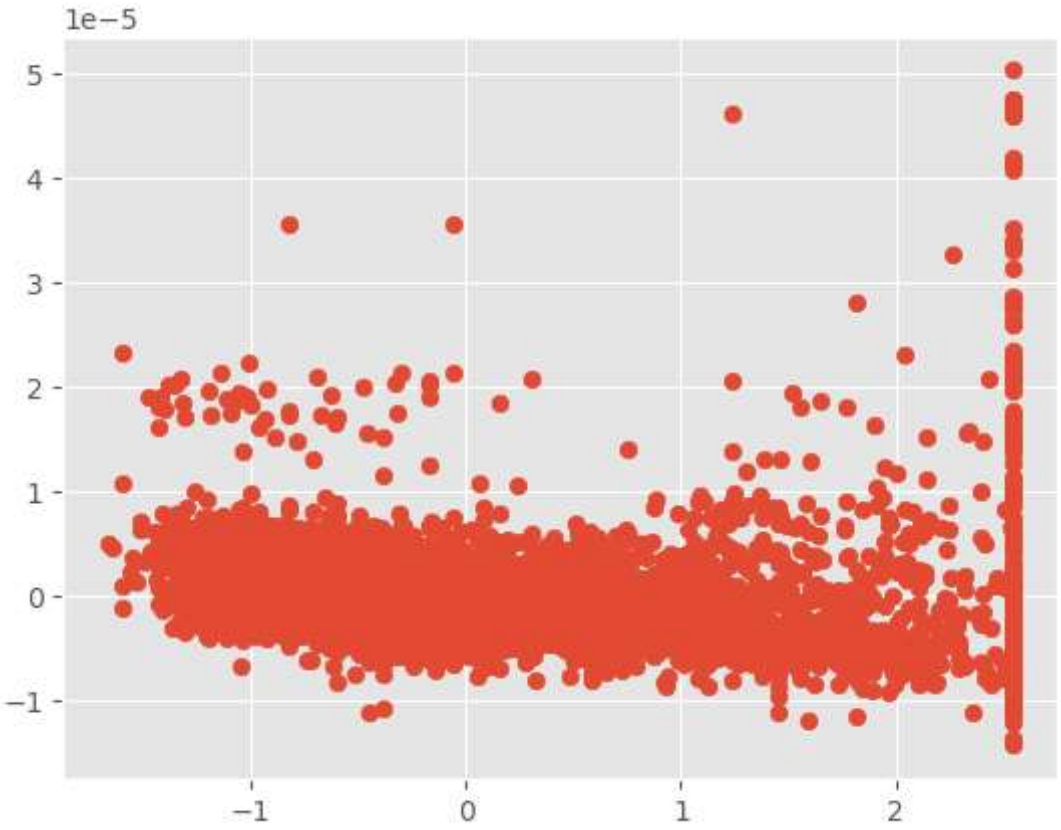
```
2.3825704208244534e-11
2.983722692413913e-06
4.881158080644852e-06
                     Columns  Coefficient Estimate
0           log_total_rooms          9.461744e-06
1        log_total_bedrooms         -1.105804e-05
2            log_population          5.553522e-06
3            log_households         -6.402752e-06
4         log_median_income          7.355184e-05
5  scaled_housing_median_age         -3.578161e-07
6           scaled_latitude          1.573237e-06
7          scaled_longitude          1.527744e-06
8       other_income_source          8.666076e-06
```

In [92]:

```
# has a uniformity in the data
plt.scatter(pred , residulas)
```

Out[92]:

```
<matplotlib.collections.PathCollection at 0x2afeae7a8b0>
```

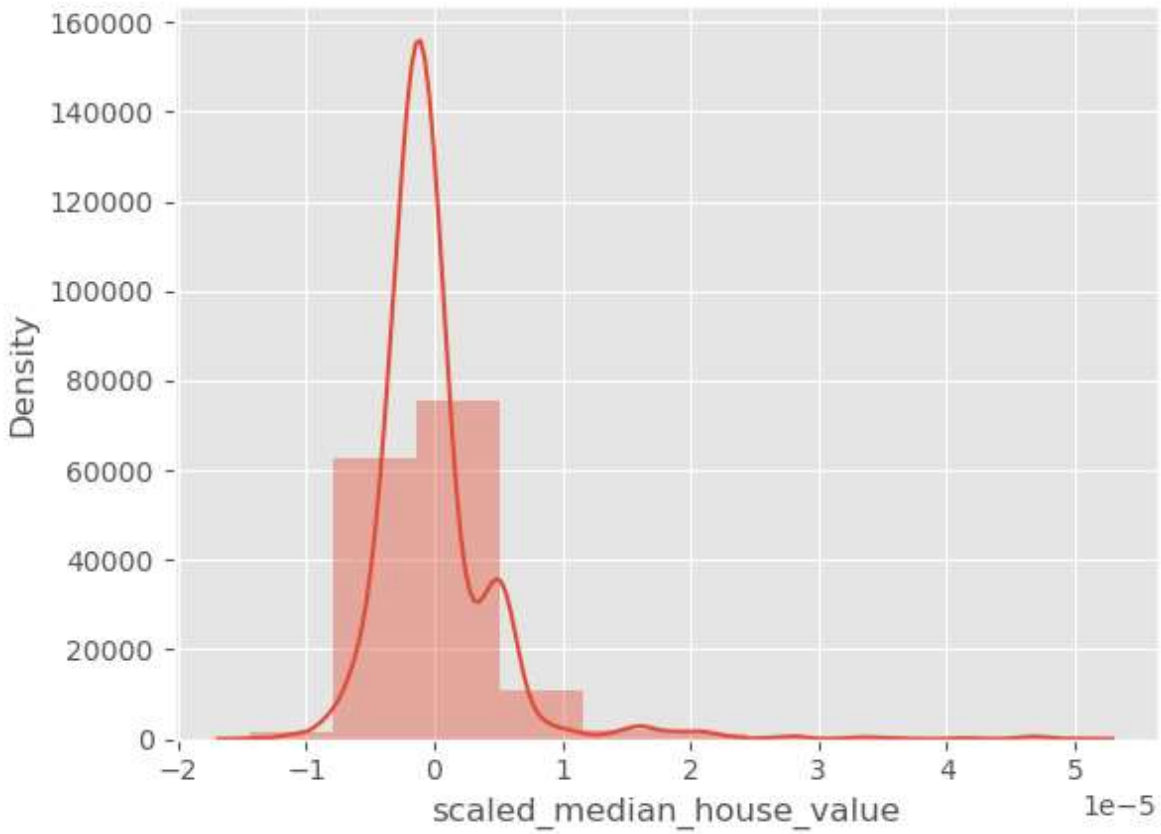


In [93]:

```
# it is only little bit skewed.
residulas = y_test - pred
sns.distplot(residulas, bins = 10)
```

Out[93]:

```
<AxesSubplot:xlabel='scaled_median_house_value', ylabel='Density'>
```



another way to calculate mean squared error via numpy library

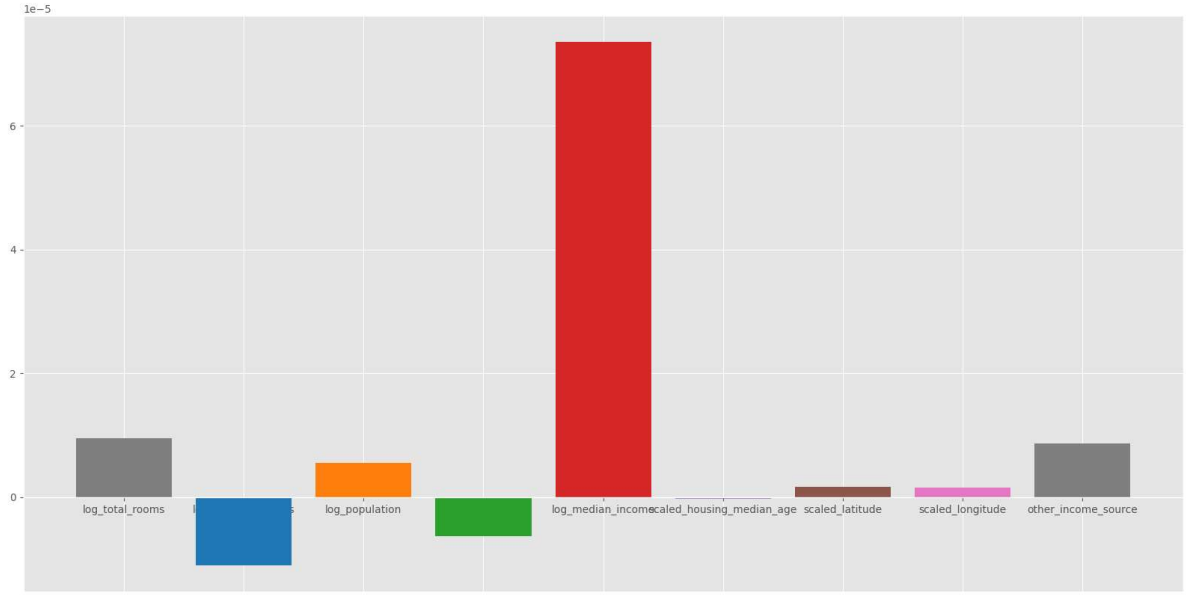
In [94]:

```
mean_squared_error_ridge = np.mean((pred - y_test)**2)
print(mean_squared_error_ridge)
```

```
2.3825704208244556e-11
```

In [95]:

```
### it is misleading that housing_median_age has removed actually it is so small that it ca
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan',
'tab:orange', 'tab:green', 'tab:blue', 'tab:olive']

ax.bar(ridge_coefficient["Columns"],
ridge_coefficient['Coefficient Estimate'],
color = color)

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```



Lasso Regression

In [96]:

```
### lasso removes the feature by making it zero which has no significance with the dependen
        ### it is showing zero as we have already removed multicollinearity between X feat
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
# Train the model
lasso = Lasso(alpha = 5)
lasso.fit(X_train, y_train)
pred1 = lasso.predict(X_test)

# Calculate Mean Squared Error, mean absolute error and RMSE
print(mean_squared_error(y_test, pred1))
print(mean_absolute_error(y_test,pred1))
print(np.sqrt(mean_squared_error(y_test, pred1)))
lasso_coeff = pd.DataFrame()
lasso_coeff["Columns"] = X_train.columns
lasso_coeff['Coefficient Estimate'] = pd.Series(lasso.coef_)

print(lasso_coeff)
```

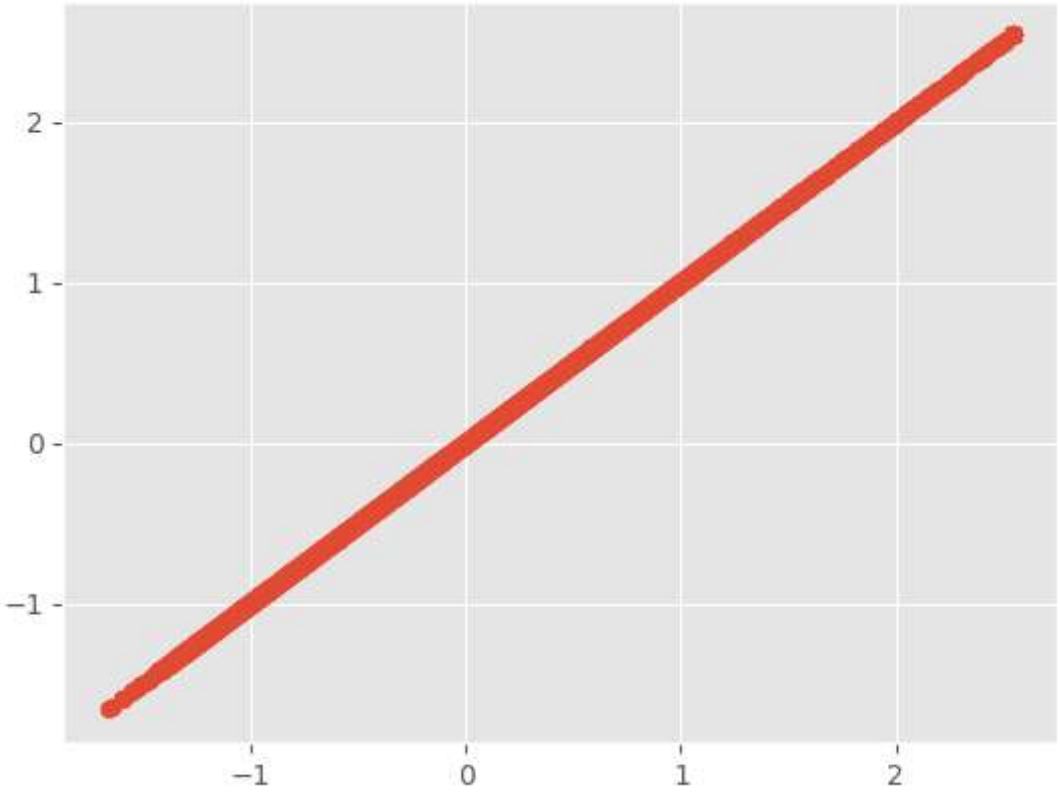
```
2.0235478335881914e-09
3.5053405452640325e-05
4.498386192389657e-05
                     Columns  Coefficient Estimate
0            log_total_rooms              0.000000
1         log_total_bedrooms              0.000000
2             log_population             -0.000000
3             log_households              0.000000
4          log_median_income              0.000000
5  scaled_housing_median_age              0.000000
6            scaled_latitude             -0.000000
7           scaled_longitude             -0.000000
8        other_income_source              0.000009
```

In [97]:

```
# the data has trained and tested most accurately.
plt.scatter(y_test, pred1)
```

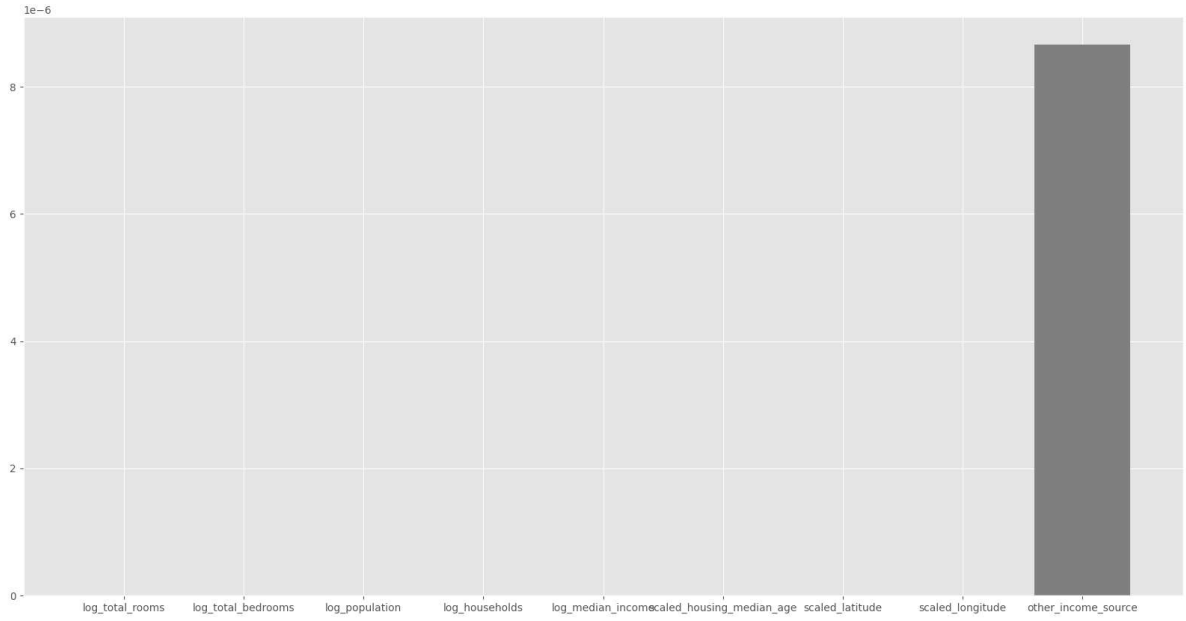
Out[97]:

```
<matplotlib.collections.PathCollection at 0x2afe9a97220>
```



In [98]:

```
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan',
'tab:orange', 'tab:green', 'tab:blue', 'tab:olive']

ax.bar(lasso_coeff["Columns"],
lasso_coeff['Coefficient Estimate'],
color = color)

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```



ElasticNet is a combination of both L1 and L2 normalization

In [99]:

```
## checking overfitting and removing features with the help of one function from sklearn is

from sklearn.linear_model import ElasticNet

# Train the model
e_net = ElasticNet(alpha = 1)
e_net.fit(X_train, y_train)
pred_enet = e_net.predict(X_test)
# imprting libraries
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error

# Calculate Mean Squared Error, mean absolute error and RMSE
print(mean_squared_error(y_test,pred_enet))
print(mean_absolute_error(y_test, pred_enet))
print(np.sqrt(mean_squared_error(y_test, pred_enet)))

e_net_coeff = pd.DataFrame()
e_net_coeff["Columns"] = X_train.columns
e_net_coeff['Coefficient Estimate'] = pd.Series(e_net.coef_)
e_net_coeff
```

```
1.6842369158948601e-10
9.622395774970751e-06
1.2977815362744456e-05
```

Out[99]:

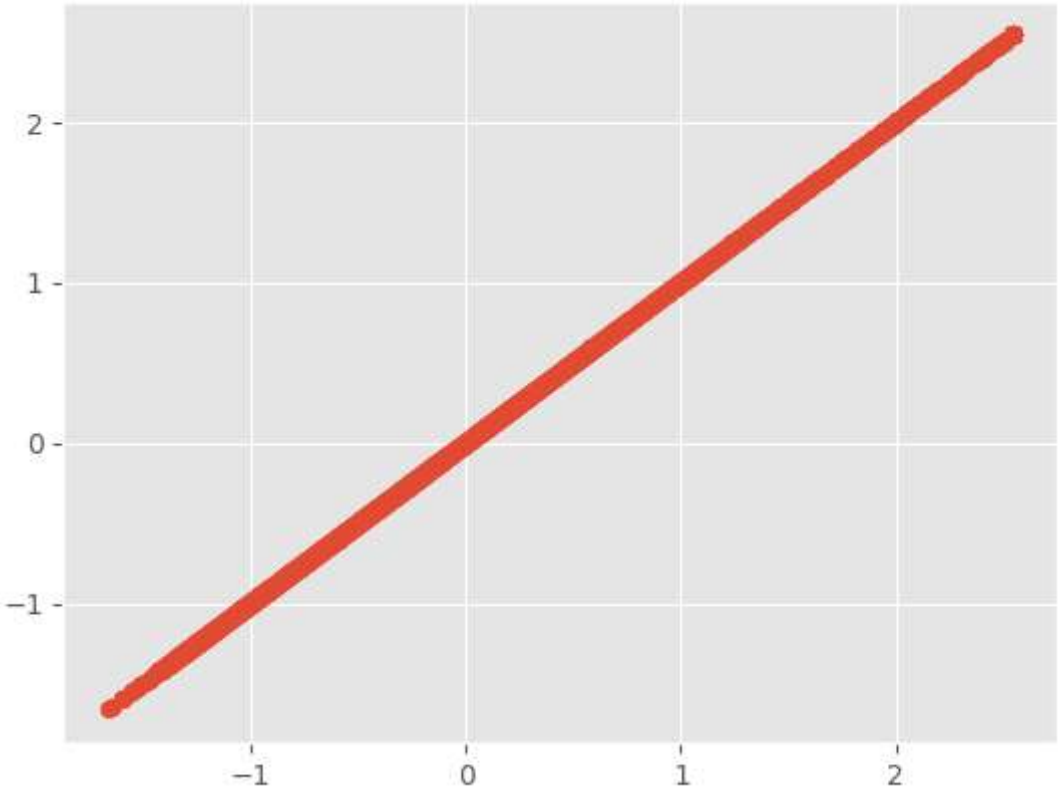
	Columns	Coefficient Estimate
0	log_total_rooms	0.000000
1	log_total_bedrooms	-0.000000
2	log_population	-0.000000
3	log_households	-0.000000
4	log_median_income	0.000000
5	scaled_housing_median_age	-0.000000
6	scaled_latitude	-0.000000
7	scaled_longitude	0.000000
8	other_income_source	0.000009

In [100]:

```
# the test by elaticnet is also accurately done.
plt.scatter(y_test, pred_enet)
```

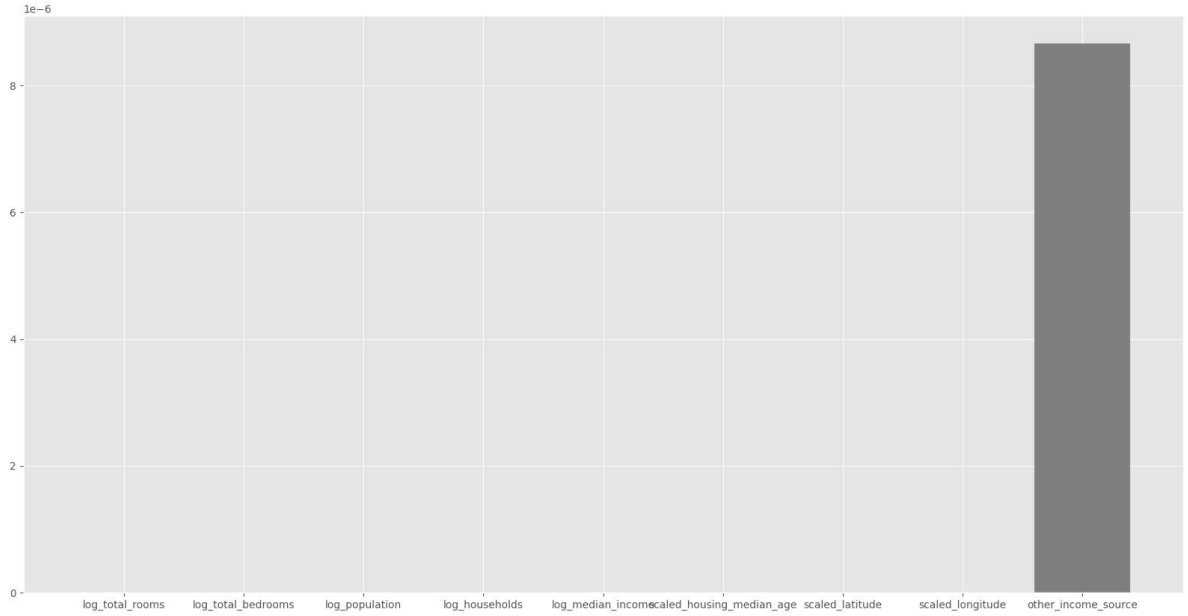
Out[100]:

```
<matplotlib.collections.PathCollection at 0x2afe9ba4ee0>
```



In [101]:

```
fig, ax = plt.subplots(figsize =(20, 10))

color =['tab:gray', 'tab:blue', 'tab:orange',
'tab:green', 'tab:red', 'tab:purple', 'tab:brown',
'tab:pink', 'tab:gray', 'tab:olive', 'tab:cyan',
'tab:orange', 'tab:green', 'tab:blue', 'tab:olive']

ax.bar(e_net_coeff["Columns"],
e_net_coeff['Coefficient Estimate'],
color = color)

ax.spines['bottom'].set_position('zero')

plt.style.use('ggplot')
plt.show()
```



simple linear regression is done.

In [ ]: