DATA CLEANING OR FEATURE ENGINEERING

TASKS TO BE PERFORMED

1. DATA CLEANING

2. HANDLING DATASET

3. PROBLEM STATEMENT

4. MODEL BUILDING

4.1 SIMPLE LINEAR REGRESSION (RIDGE,LASSO,ELARTIC NET)
4.2 LOGISTIC REGRESSION

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]:

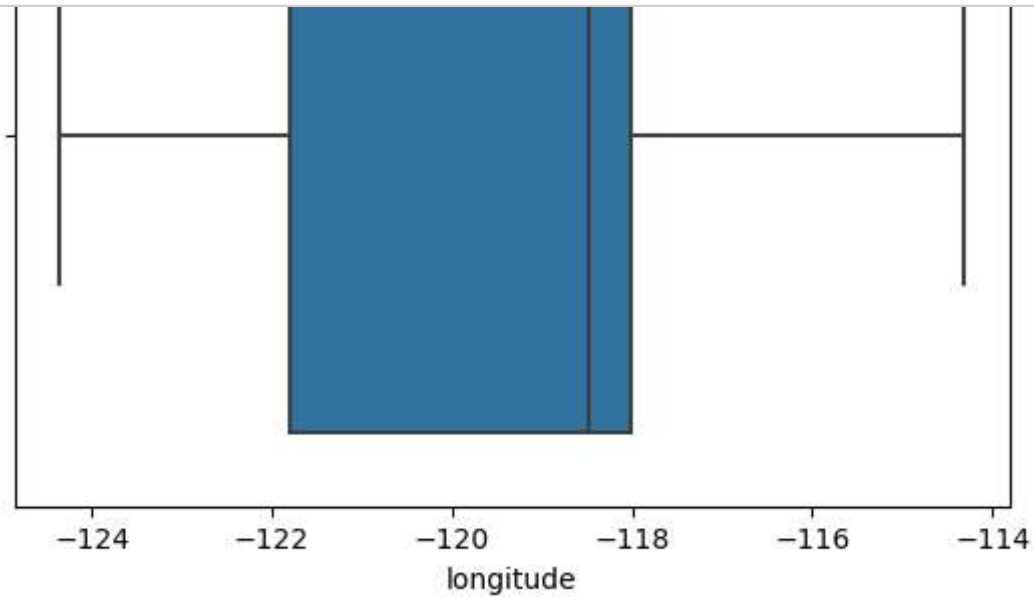
```
# comment -  scaling or normalizing datatypes of numerical features.
for col in numerical_features:
    print(df[col].value_counts(normalize=True) * 100)
```

```
-118.31    0.784884
-118.30    0.775194
-118.29    0.717054
-118.27    0.697674
-118.32    0.687984
             ...
-123.54    0.004845
-115.94    0.004845
-115.99    0.004845
-116.81    0.004845
-123.71    0.004845
Name: longitude, Length: 844, dtype: float64
34.06    1.182171
34.05    1.143411
34.08    1.133721
34.07    1.119186
34.04    1.070736
           ...
41.01    0.004845
```

2.2 handling outliers

In [19]:

```
# 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

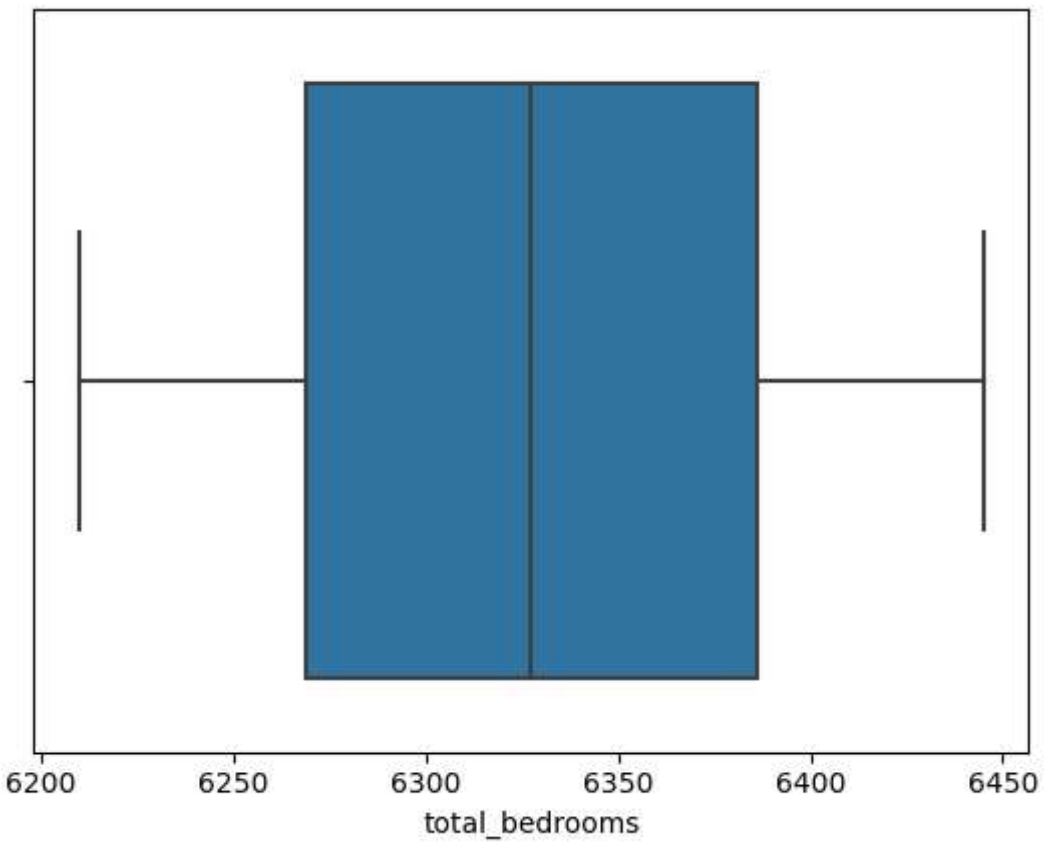
2. outliers after

3. and how many left to remove.

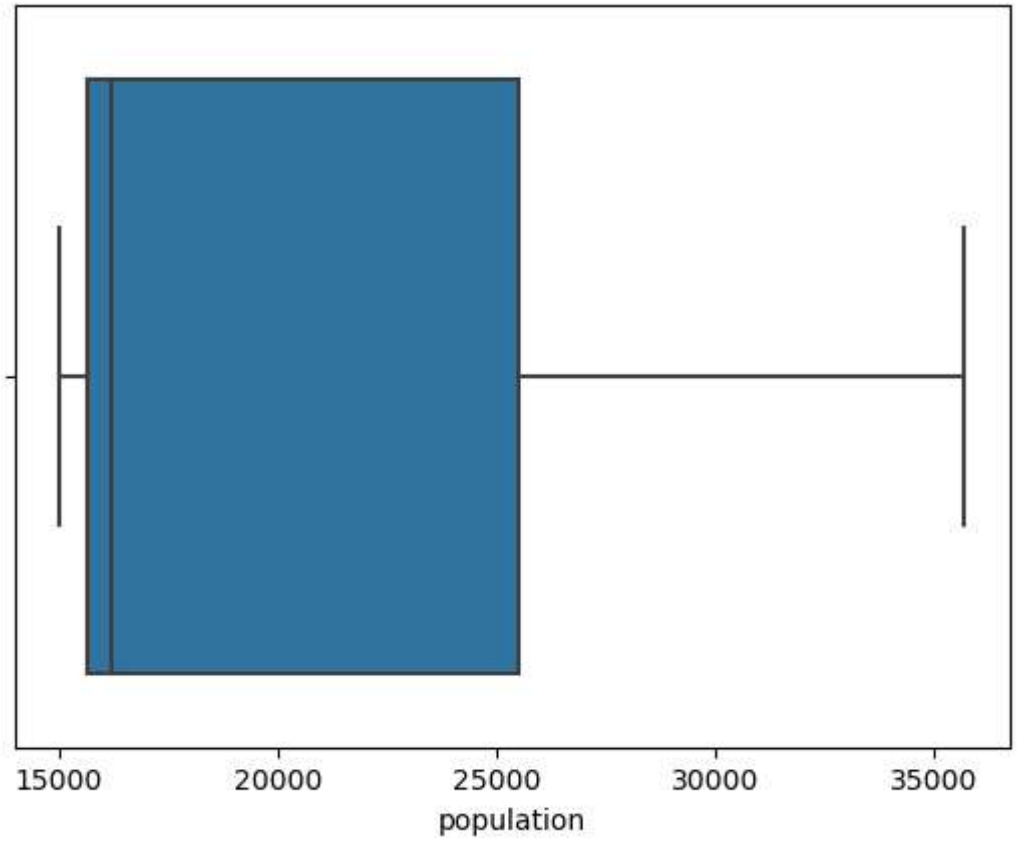
In [20]:

```
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
```



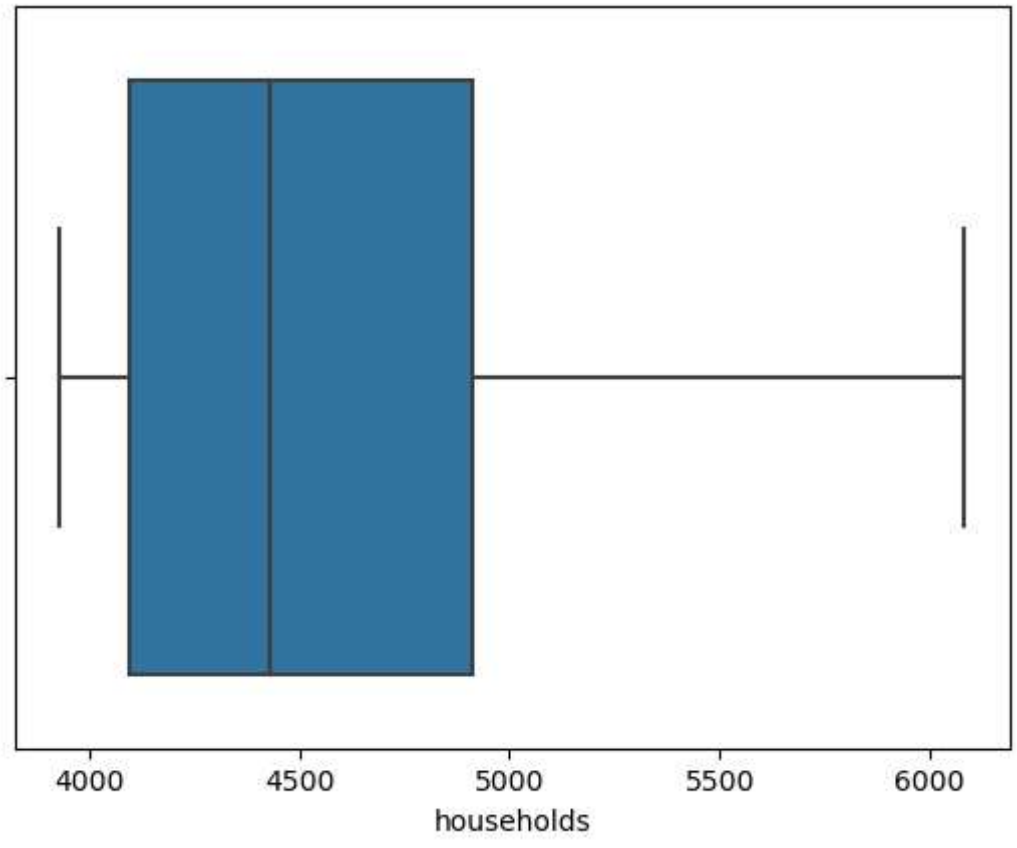
```
before removing outliers 20640
after removing outliers 6
new outliers 20634
```



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()
```

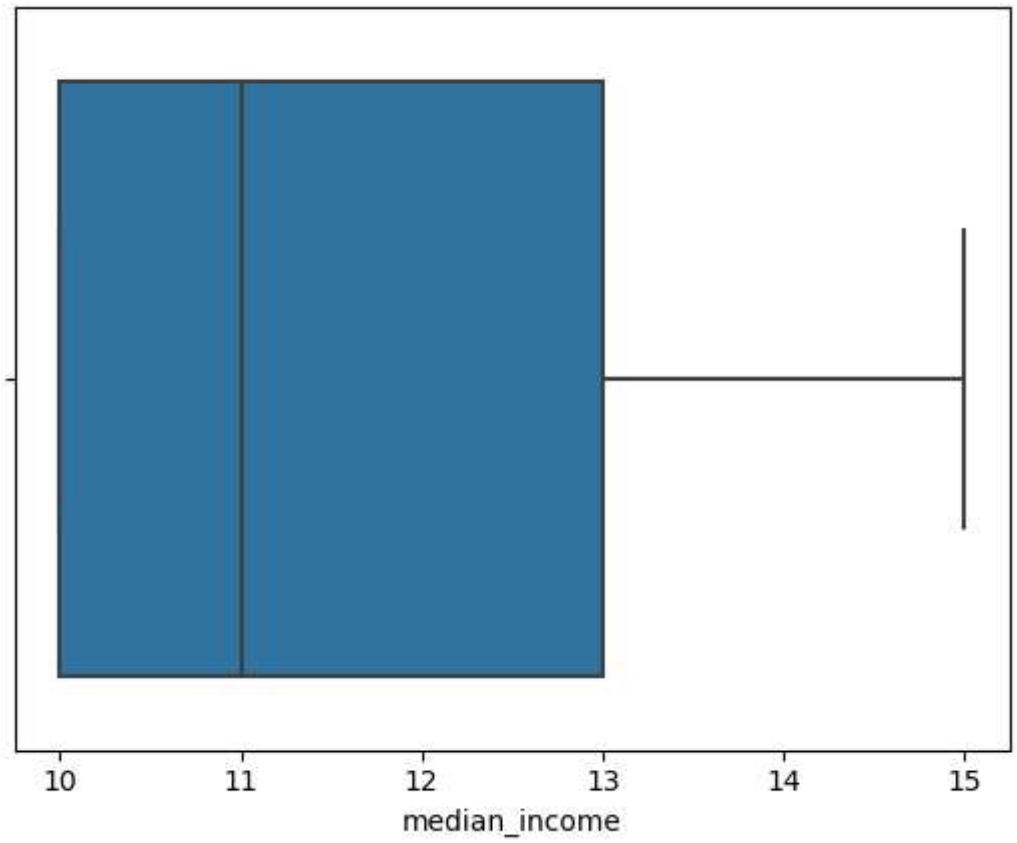
```
before removing outliers 20640
after removing outliers 18
new outliers 20622
```



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()
```

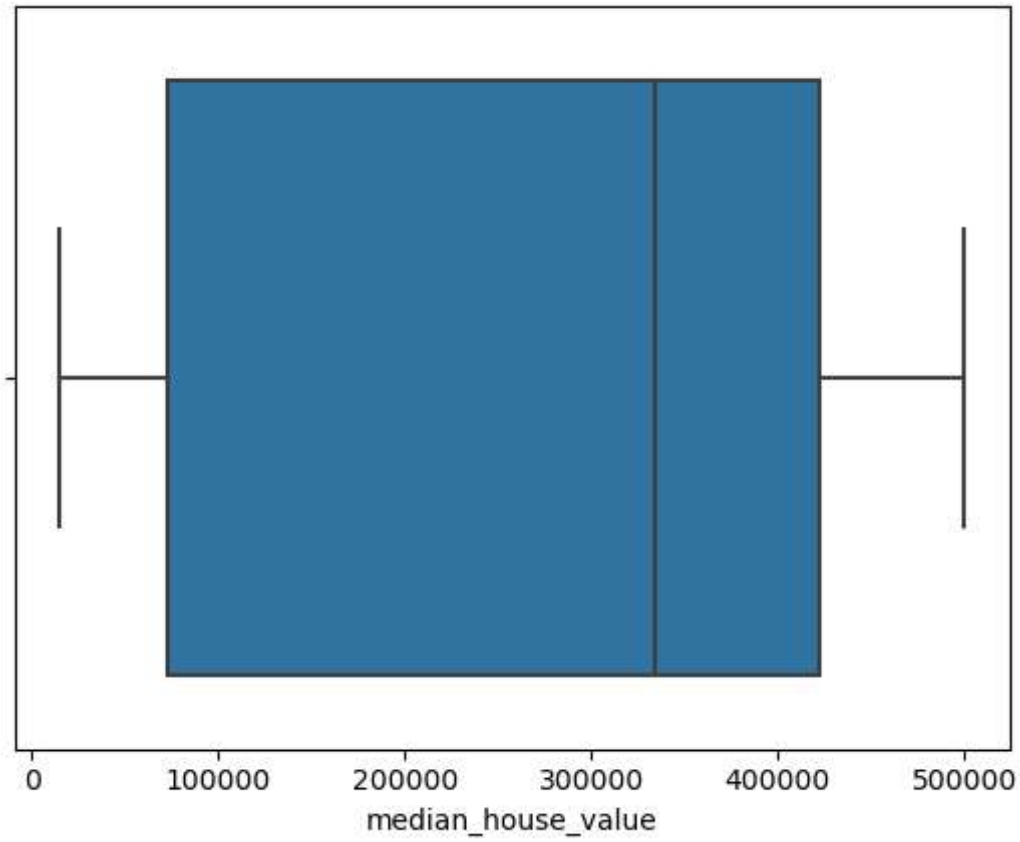
```
before removing outliers 20640
after removing outliers 309
new outliers 20331
```



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()
```

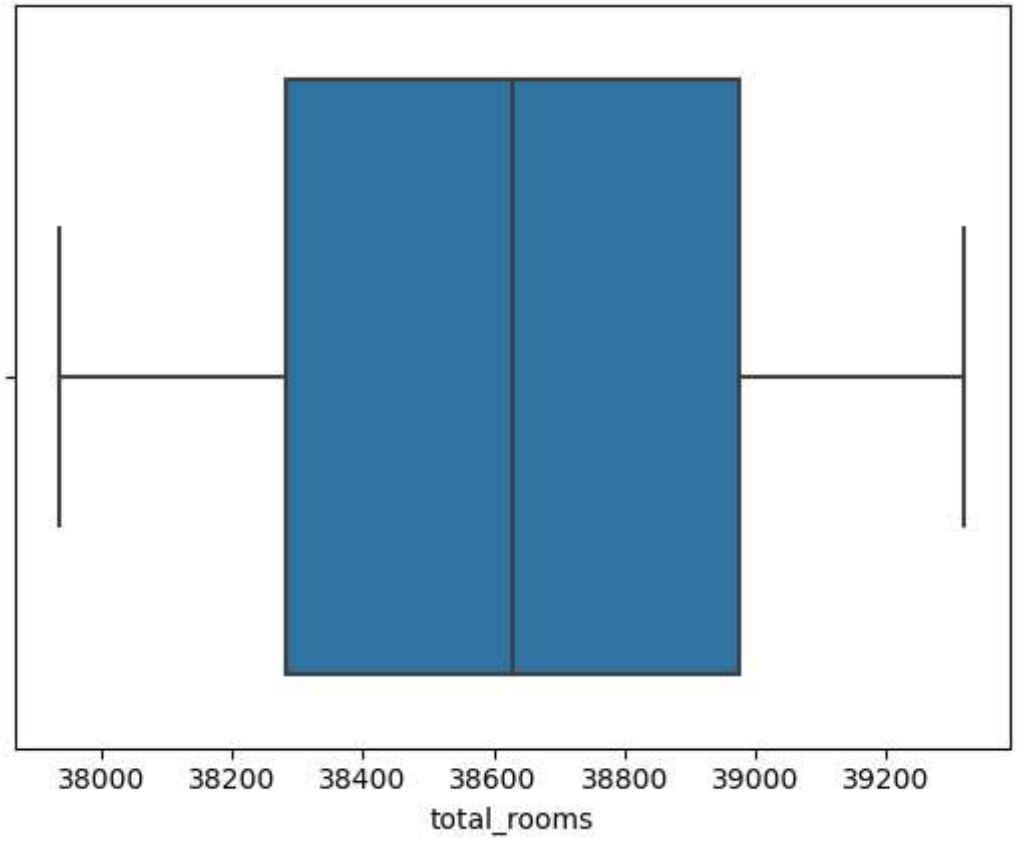
```
before removing outliers 20640
after removing outliers 6074
new outliers 14566
```



In [22]:

```
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
```



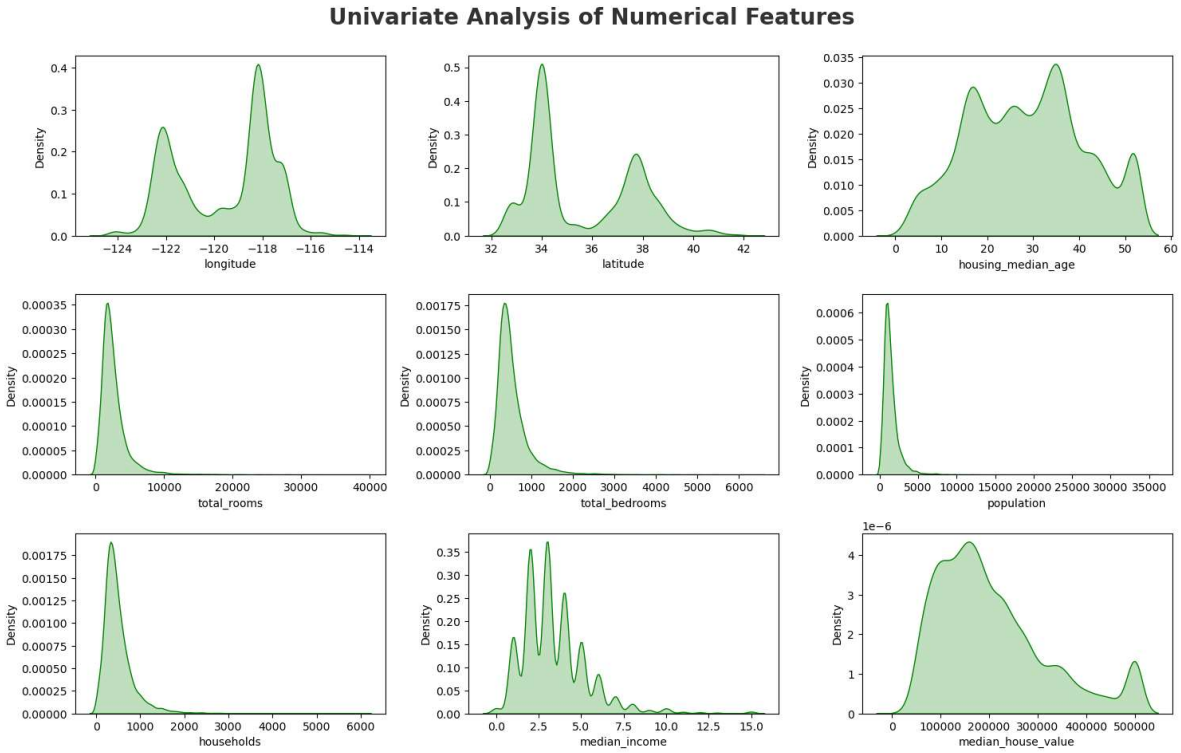
In [25]:

```
dff = df.to_csv('outliers.csv',index=False)
```

2.3 transformation of skewed data distribution

In [26]:

```
# 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()
```



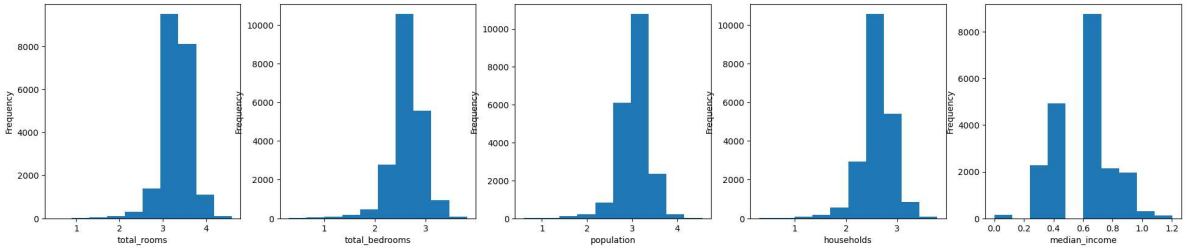
we need to transform the right skewed data into log transformation for normality of data.

In [27]:

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

In [28]:

```
# 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 [29]:

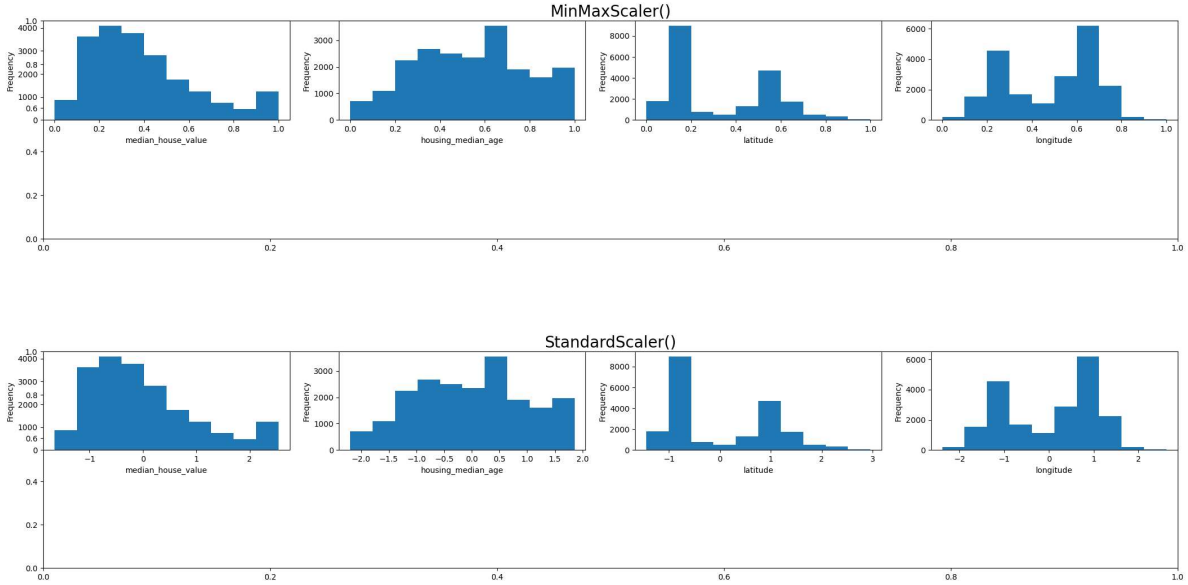
```
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import StandardScaler
```

In [30]:

```
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 [31]:

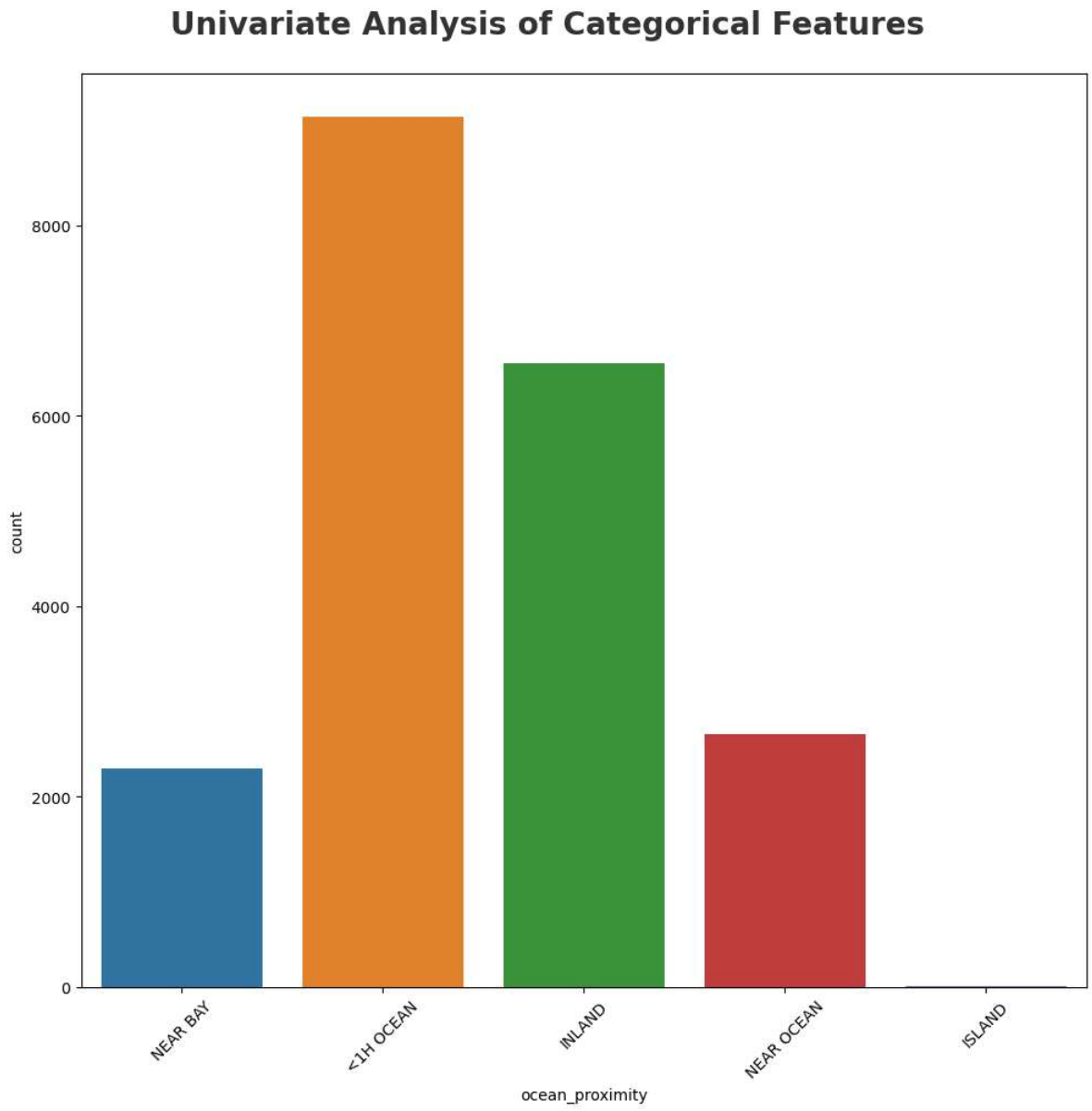
```
# comment - we have used two scaling libraries MinMaxScaler has [0-1] scale.
# observation - by scaling we have transformed the features of remaining datatypes into nor
scale_var = ['median_house_value', 'housing_median_age', 'latitude', 'longitude']
scalers_list = [MinMaxScaler(), StandardScaler()]
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 [32]:

```
# 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 [33]:

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

In [34]:

```
ohe
```

Out[34]:

	<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 [35]:

```
# 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[35]:

	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 [36]:

```
# 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 [37]:

```
# 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[37]:

	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 [38]:

```
# 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[38]:

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

In [39]:

```
# 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[39]:

	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 [40]:

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

Out[40]:

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

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

In [41]:

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

Out[41]:

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

In [42]:

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

Out[42]:

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

In [43]:

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

Out[43]:

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

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

In [44]:

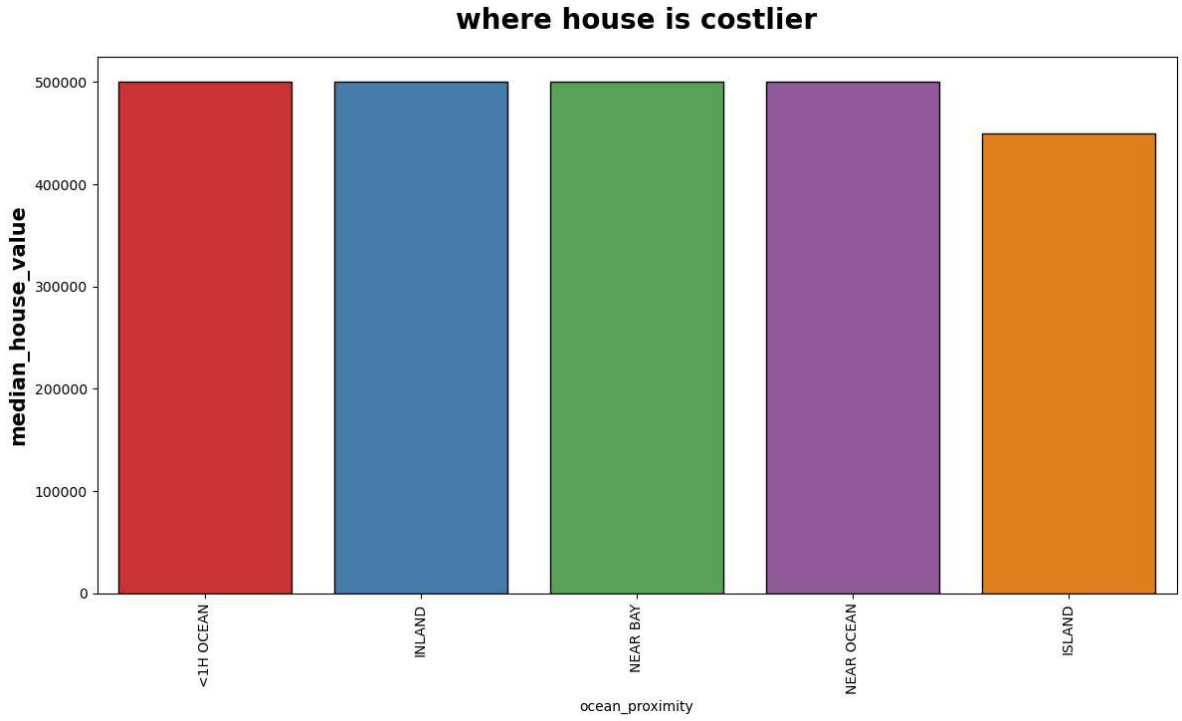
```
# observation - Except ISLAND the houses at all other locations has almost same cost.
                ## at first sight we might think that ISLAND has cheaper house, no, becau
                     ## population prefers less this location and still this price is hig
house= df.groupby('ocean_proximity').median_house_value.max()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

Out[44]:

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

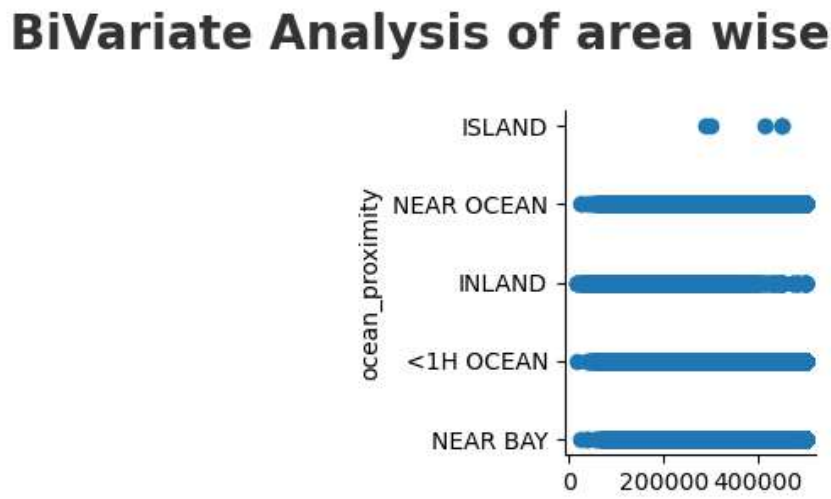
In [45]:

```
##  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 [46]:

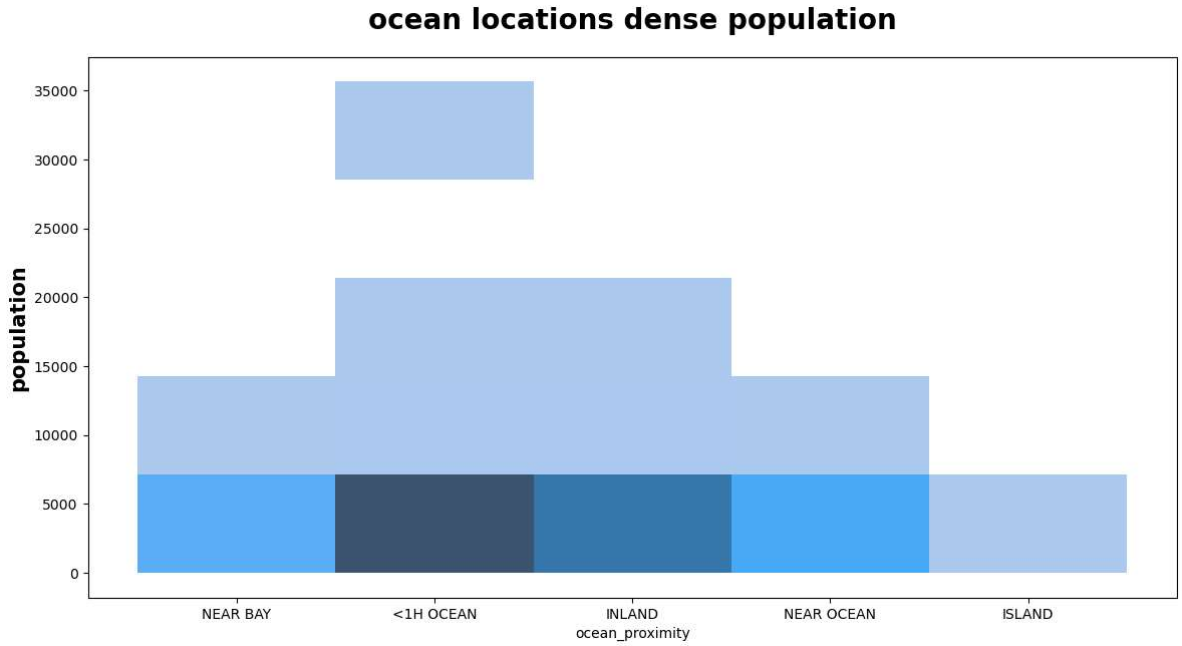
```
# observation - the ISLAND has highest house value ranges and all other have all types of r
sns.FacetGrid(df[numerical_features]).map(plt.scatter, x = df['median_house_value'], y = df
plt.suptitle('BiVariate Analysis of area wise house value', fontsize=20, fontweight='bold',
plt.xlabel ('median_house_value')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



4. Where the population is dense near ocean locations ?

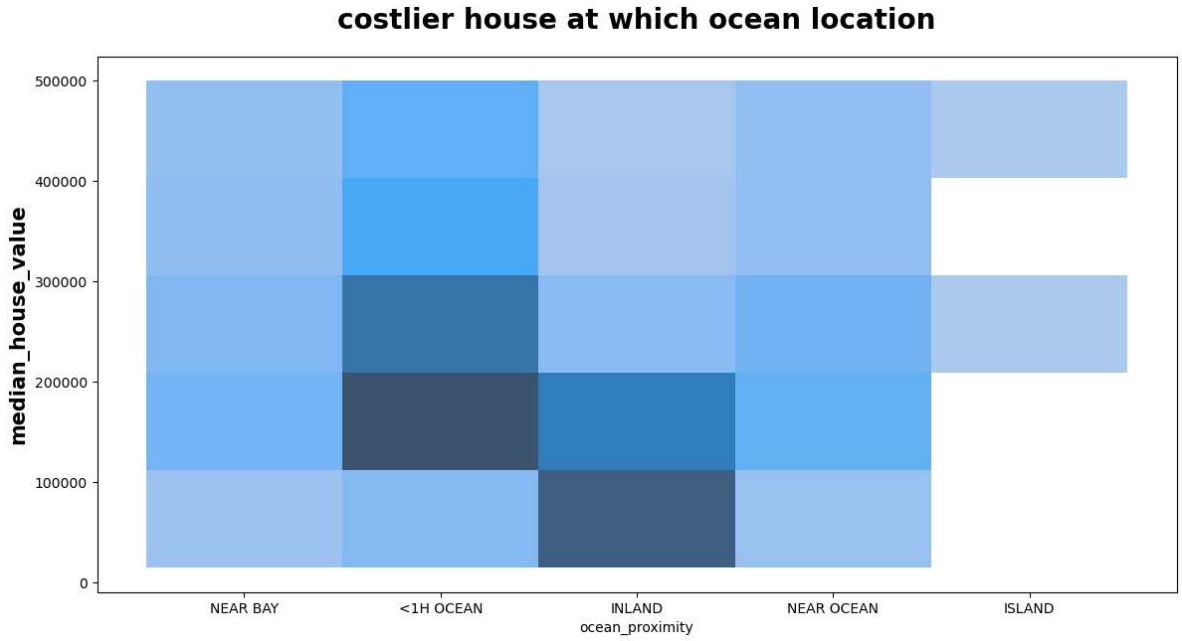
In [47]:

```
# 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 [48]:

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

Out[48]:

	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	

20331 rows × 19 columns

In [49]:

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

Out[49]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	hous
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 [50]:

```
# 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 [51]:

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

In [52]:

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

Out[52]:

	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 [53]:

```
# 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[53]:

```
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 [54]:

```
# 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 [55]:

```
# 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[55]:

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 [56]:

```
# 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[56]:

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 [57]:

```
df.shape
```

Out[57]:

```
(20640, 20)
```

In [58]:

```
df.head()
```

Out[58]:

	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 [59]:

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

In [60]:

```
data_cleaned
```

Out[60]:

	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 [61]:

```
data_cleaned.shape
```

Out[61]:

```
(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 [114]:

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

In [115]:

```
### 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 [116]:

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

inbuilt function for training and testing the x and y variables

In [117]:

```
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 [66]:

```
X_train.shape
```

Out[66]:

```
(13828, 9)
```

In [67]:

```
X_test.shape
```

Out[67]:

```
(6812, 9)
```

In [68]:

```
y_train.shape
```

Out[68]:

```
(13828,)
```

In [69]:

```
y_test.shape
```

Out[69]:

```
(6812,)
```

Model Testing

importing various models:

1. linear regression

2. ridge

3. lasso

4. elasticnet

In [119]:

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

assumptions of linear regression

1. linearity of y_test and reg_pred

2. uniform range of graph specific

3. normality for residuals

In [120]:

```
regression=LinearRegression()
```

we are fitting the data here only no execution of transformation

In [121]:

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

Out[121]:

```
LinearRegression()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

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

In [73]:

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

Out[73]:

	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 [74]:

```
### 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 [75]:

```
### 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

In [ ]:

```
```

In [76]:

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

In [77]:

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

In [78]:

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

model prediction value

In [122]:

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

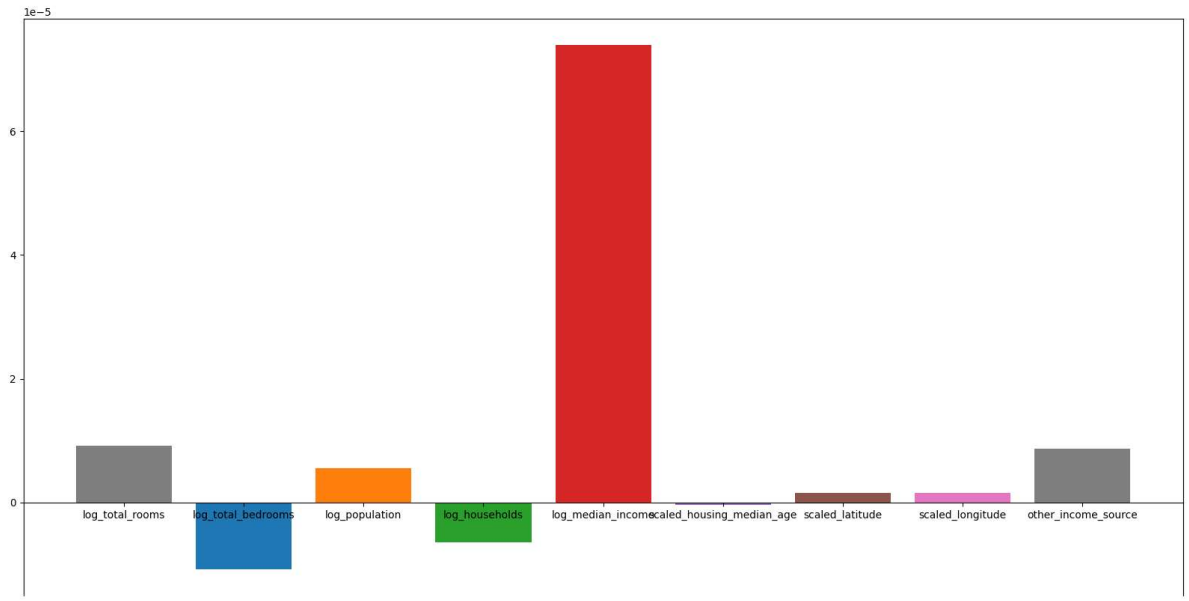
Out[122]:

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

relation of independent variables with dependent variables via linear regression

In [80]:

```
### 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:

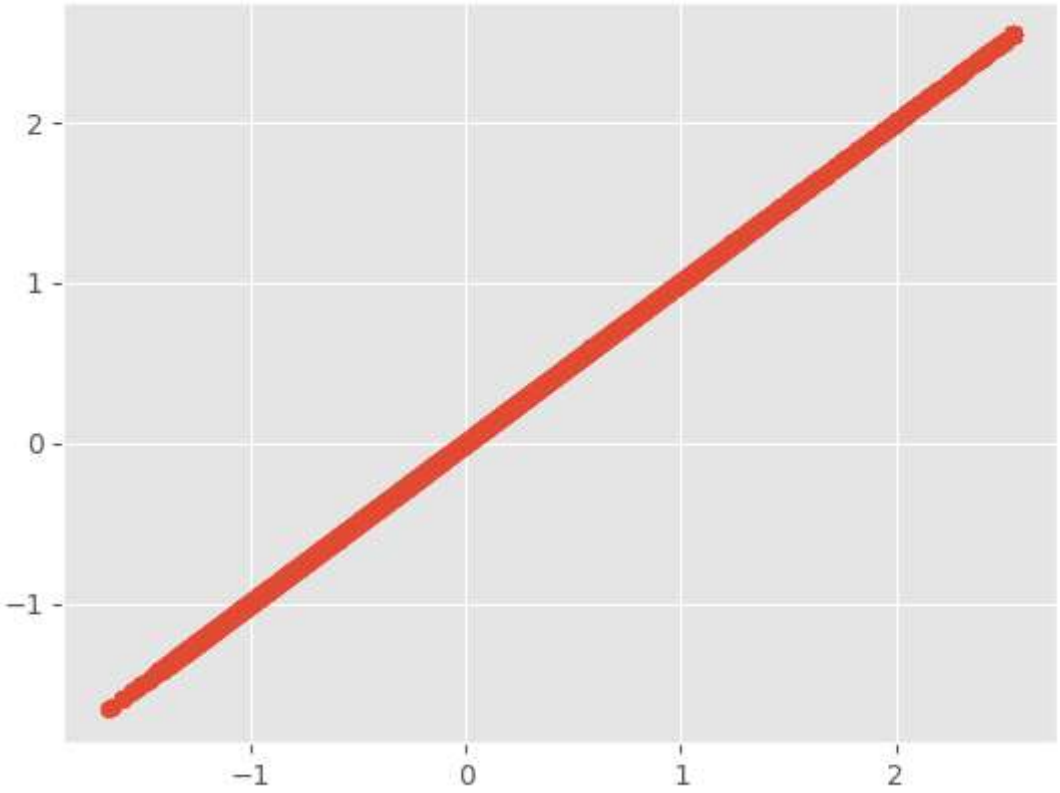
1. our best fit line has acurately minimize errors
2. the actual and predicted data showing good accuracy of the model.

In [123]:

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

Out[123]:

```
<matplotlib.collections.PathCollection at 0x229ebc22550>
```



In [81]:

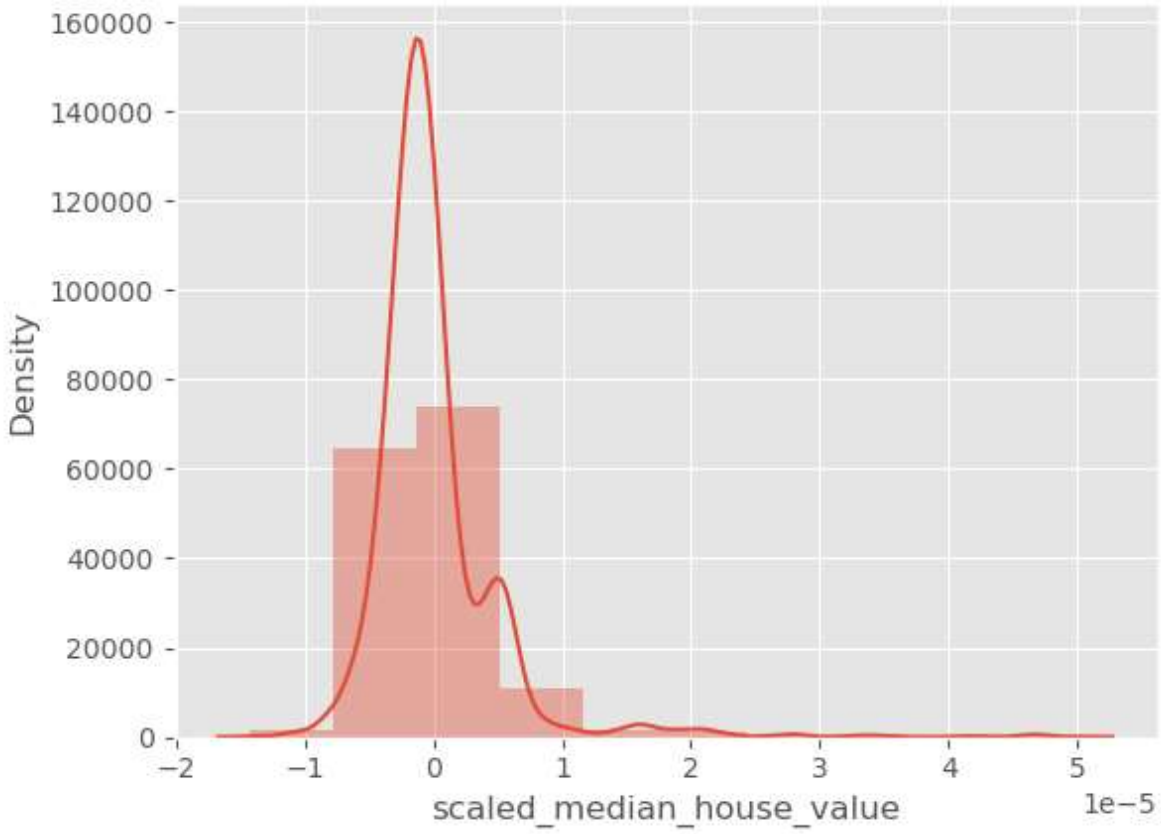
```
### residuals means errors in the model
   ### they aries due to actual data and predicted data test.
```

In [124]:

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

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

Out[124]:

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

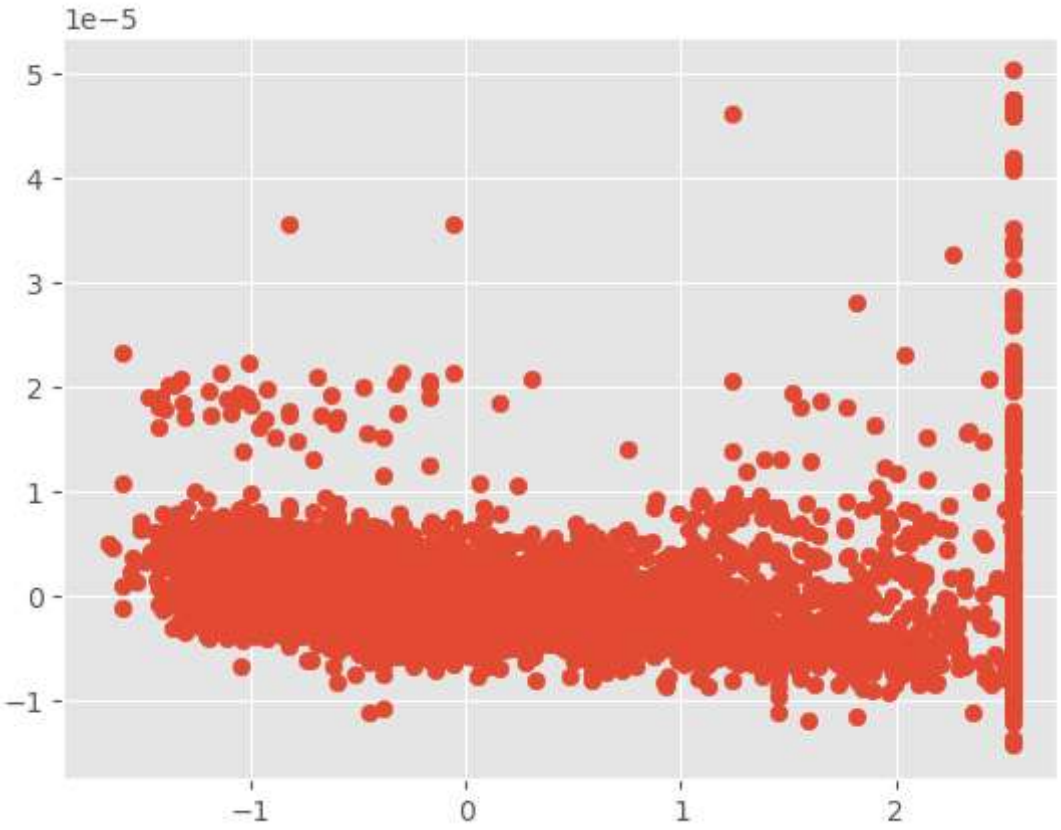


In [125]:

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

Out[125]:

```
<matplotlib.collections.PathCollection at 0x229eb281b20>
```



In [84]:

```
### 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 [126]:

```
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 [127]:

```
### 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 [128]:

```
### 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_square = 1 - (1-score)*(len(y_test)-1)/(len(y_test)-X_test.shape[1]-1)
Adj_R_square
```

Out[128]:

```
0.9999999999761859
```

importing library for checking via Ridge

we use ridge when the model is overfitting

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

In [97]:

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

Train the model

In [98]:

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

Out[98]:

```
Ridge()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [99]:

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

In [100]:

```
### 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
```

another way to calculate mean squared error via numpy library

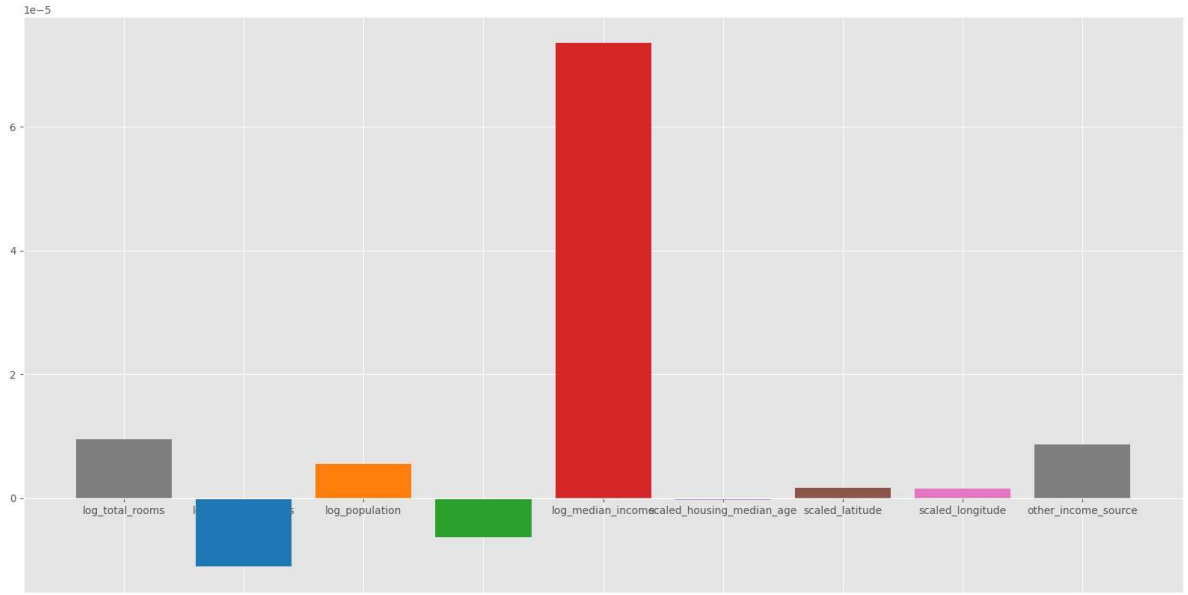
In [101]:

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

```
2.3825704208244556e-11
```

In [102]:

```
### 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()
```



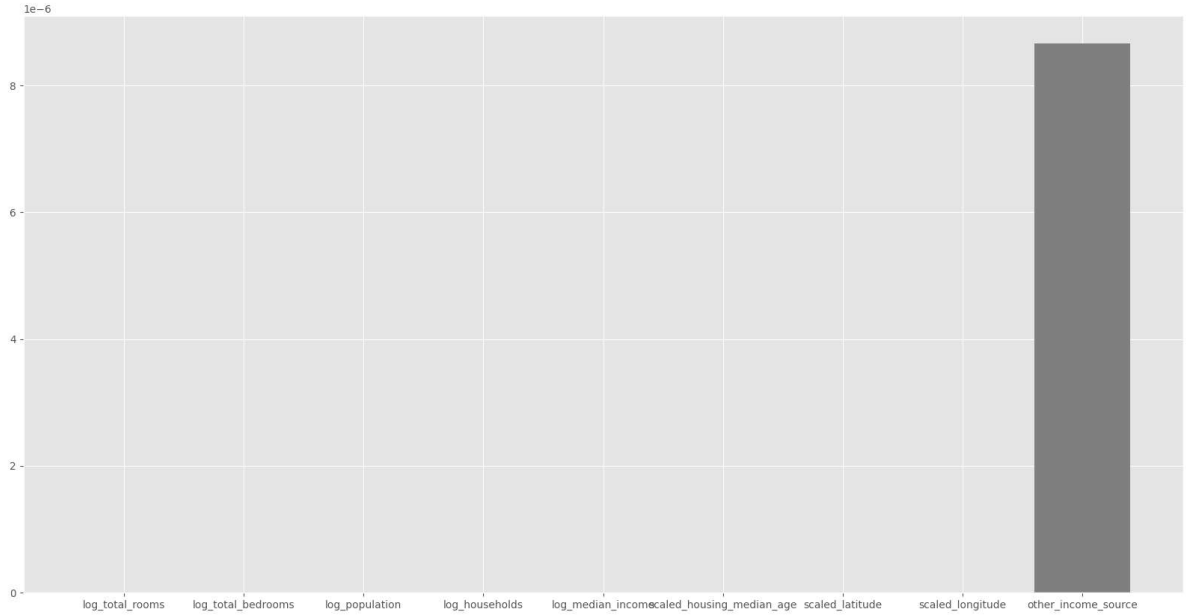
In [103]:

```
### 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 [104]:

```
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 [105]:

```
## 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)

 #importing 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, reg_pred))
print(mean_absolute_error(y_test, reg_pred))
print(np.sqrt(mean_squared_error(y_test, reg_pred)))

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
```

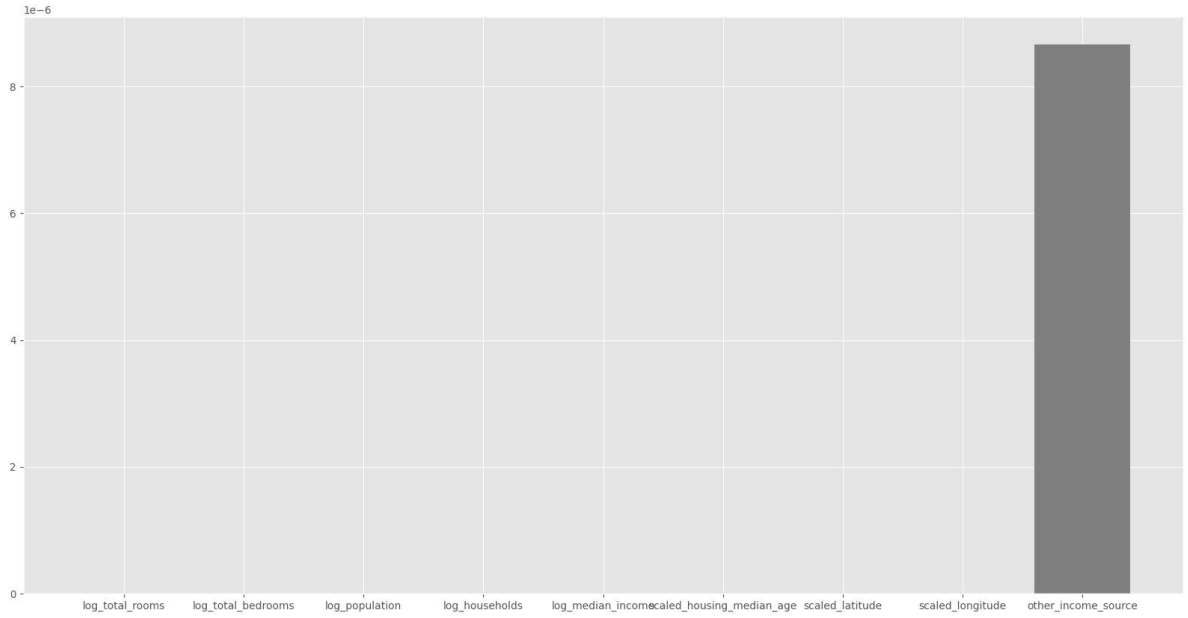
```
4.226384063071317
1.7960275486287254
2.0558171278280852
```

Out[105]:

	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 [106]:

```
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()
```



In [ ]:

Logistic Regression

task performed

our dataset has not any classification feature, so we created classification threshold for solving LogisticReg problem.

the classification is binary.

training and testing model for imbalanced classification model.

handling imbalanced feature.

again testing and training model for balanced data.

```
### understanding the new data
```

graphical analysis.

comparison

In [98]:

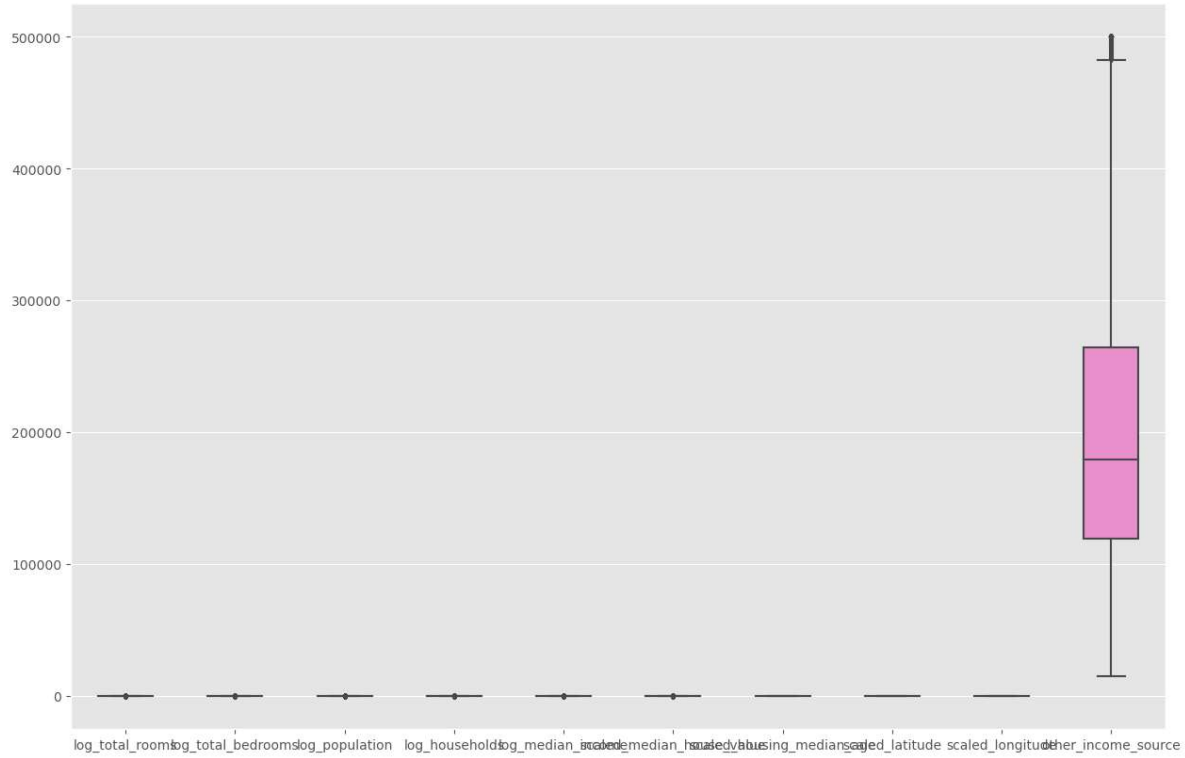
```
# checking the cleaned data whether it is cleaned or need further cleaning.
```

In [109]:

```
### the data is cleaned, but we can not see the size may be small to plot over here.
fig, ax = plt.subplots(figsize = (15,10))
sns.boxplot(data = data_cleaned, width = 0.5 , ax = ax, fliersize = 3)
```

Out[109]:

<AxesSubplot:>



In [112]:

```
data_cleaned
```

Out[112]:

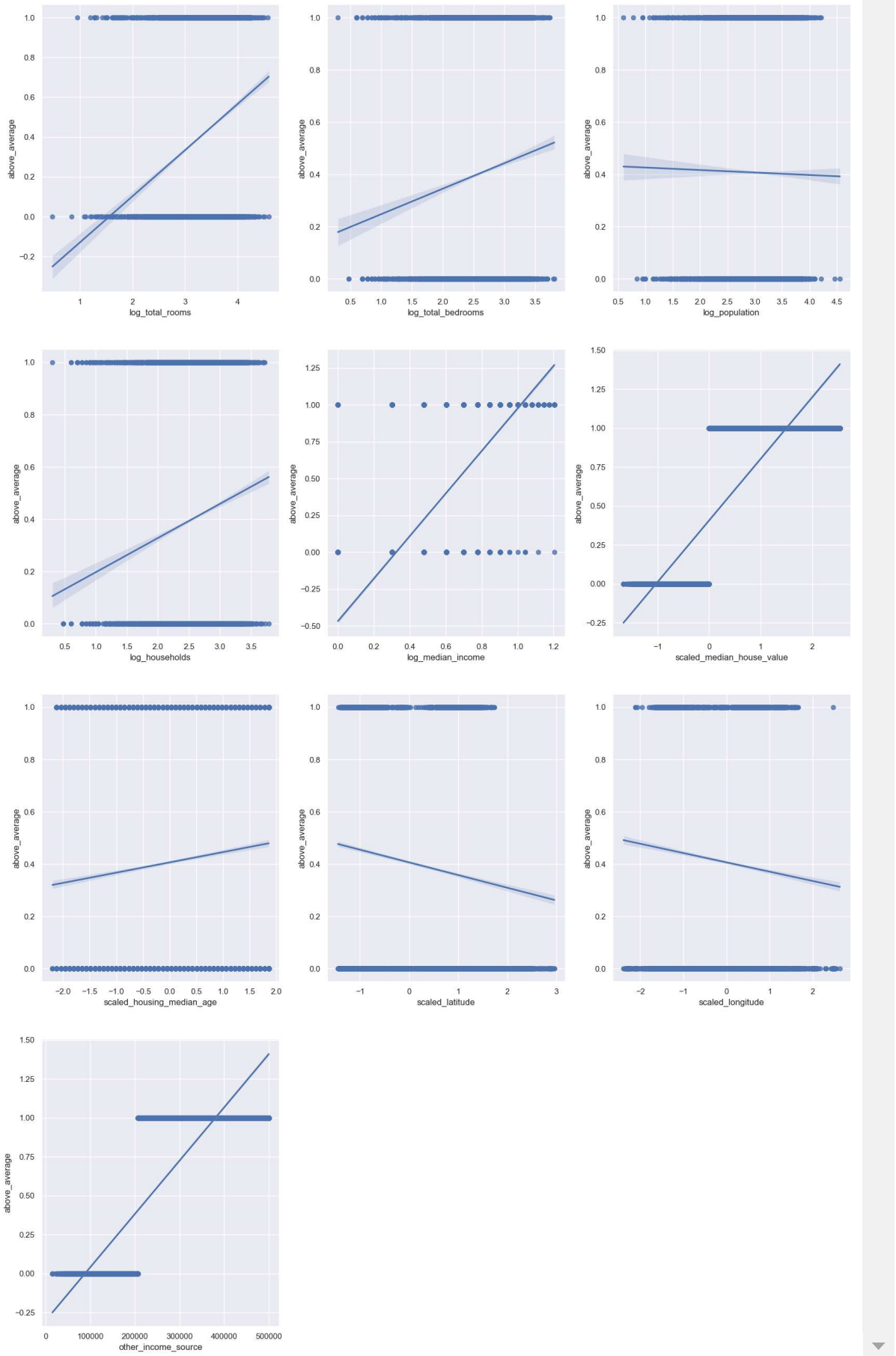
	ocean_proximity	log_total_rooms	log_total_bedrooms	log_population	log_households
0	NEAR BAY	2.944976	2.113943	2.509203	2.103804
1	NEAR BAY	3.851258	3.044148	3.380573	3.056524
2	NEAR BAY	3.166726	2.281033	2.696356	2.250420
3	NEAR BAY	3.105510	2.372912	2.747412	2.342423
4	NEAR BAY	3.211654	2.448706	2.752816	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

regplot enables us to analyse that how best fit line is able to anlyse the how model able to minimise the errors.

In [106]:

```
#plt.figure(figsize = (20,50))
#for i in enumerate(numerical_features):
    #plt.subplot(6,3,i[0]+1)
    #sns.set(rc={'figure.figsize':(8,10)})
    #sns.regplot(y = 'above_average' , x= i[1], data = data_cleaned)
#plt.show()
```



only analysing the dataset

creating more columns only for analysis that how much rooms are for each household in a block

how much population is living in a block in a house.

In [107]:

```
 data_cleaned['rooms_per_household'] = data_cleaned['log_total_rooms'] / data_cleaned['log_
data_cleaned['bedrooms_per_room'] = data_cleaned['log_total_bedrooms'] / data_cleaned['log_
data_cleaned['population_per_household'] = data_cleaned['log_population'] / data_cleaned['l
```

In [108]:

```
data_cleaned
```

Out[108]:

	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 × 15 columns

logistic classification creation

creating a classification threshold as:

```
### if mean of data for scaled_median_house_value has x >1 then above average has
 a value denoted by 1.
### if mean of data for scaled_median_house_value has x <1 then above average has
 a value denoted by 0.
```

binary classification

In [232]:

```
data_cleaned['above_average'] = data_cleaned['scaled_median_house_value'].apply(lambda x: 1
```

In [233]:

```
data_cleaned
```

Out[233]:

	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 × 15 columns

segregating feature in X and y as independent and dependent variables.

In [110]:

```
x = data_cleaned.iloc[:,:-1]
```

In [111]:

```
x
```

Out[111]:

	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 × 14 columns

In [112]:

```
## it is always in a series form.
y = data_cleaned.iloc[:,-1]
```

In [113]:

```
y
```

Out[113]:

```
0        1.192698
1        1.106019
2        1.198157
3        1.172893
4        1.139895
           ...
20635    1.161734
20636    1.238740
20637    1.138757
20638    1.128273
20639    1.153130
Name: population_per_household, Length: 20640, dtype: float64
```

importing sklearn for training and testing the model

In [134]:

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

testing separately for x

In [179]:

```
data_cleaned_full_train, data_cleaned_test = train_test_split(
    data_cleaned[(data_cleaned.columns.difference(['scaled_median_house_value']))], test_si
```

In [180]:

```
data_cleaned_train, data_cleaned_val = train_test_split(data_cleaned_full_train, test_size=
```

In [117]:

```
len(data_cleaned_train), len(data_cleaned_val), len(data_cleaned_test)
```

Out[117]:

```
(12384, 4128, 4128)
```

In [118]:

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

In [387]:

```
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.fit(X_test)
```

checking for above average classification 1 and 0 counts but in percentage format.

In [120]:

```
data_cleaned['above_average'].value_counts(normalize=True)
```

Out[120]:

```
0    0.59375
1    0.40625
Name: above_average, dtype: float64
```

testing for y

In [121]:

```
y_train, y_val, y_test = data_cleaned_train['above_average'], data_cleaned_val['above_avera
for data_cleaned_ in [data_cleaned_train, data_cleaned_val,data_cleaned_test]:
    del data_cleaned_['above_average']
```

In [122]:

```
y_train
```

Out[122]:

```
17244    1
8817     1
19686    0
3545     1
17019    1
        ..
5606     0
16339    0
14965    1
11117    0
8472     0
Name: above_average, Length: 12384, dtype: int64
```

In [123]:

```
y_val
```

Out[123]:

```
2071     0
2612     0
10838    1
4061     1
10767    1
        ..
2285     0
16904    1
18139    1
11471    1
788      0
Name: above_average, Length: 4128, dtype: int64
```

In [124]:

```
y_test
```

Out[124]:

```
20046    0
3024     0
15663    1
20484    1
9814     1
        ..
15362    1
16623    1
18086    1
2144     0
3665     0
Name: above_average, Length: 4128, dtype: int64
```

In [125]:

```
data_cleaned_
```

Out[125]:

	bedrooms_per_room	log_households	log_median_income	log_population	log_total_bedr
20046	0.830595	2.556303	0.301030	3.143951	2.6
3024	0.760892	2.767156	0.477121	3.194792	2.6
15663	0.736605	2.984077	0.602060	3.117603	2.6
20484	0.757475	2.695482	0.778151	3.231979	2.6
9814	0.782897	2.632457	0.602060	3.026942	2.6
...	...	...	...	...	
15362	0.766730	2.656098	0.698970	3.130977	2.6
16623	0.810786	2.848805	0.477121	3.217747	2.9
18086	0.755824	2.755112	1.000000	3.200303	2.7
2144	0.785249	2.676694	0.477121	3.089198	2.6
3665	0.819870	2.652246	0.602060	3.223236	2.6

4128 rows × 13 columns

In [126]:

```
data_cleaned
```

Out[126]:

	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 × 15 columns

In [127]:

```
len(data_cleaned_train), len(data_cleaned_val), len(data_cleaned_test)
```

Out[127]:

```
(12384, 4128, 4128)
```

checking for each records of the dataset for our training and testing model.

In [181]:

```
train_dict = data_cleaned_train.to_dict(orient='records')
val_dict = data_cleaned_val.to_dict(orient='records')
```

In [138]:

```
train_dict
```

Out[138]:


```
[{'above_average': 1,
  'bedrooms_per_room': 0.8153731385643177,
  'log_households': 2.574031267727719,
  'log_median_income': 0.47712125471966244,
  'log_population': 3.1476763242410986,
  'log_total_bedrooms': 2.582063362911709,
  'log_total_rooms': 3.166726055580052,
  'ocean_proximity': '<1H OCEAN',
  'other_income_source': 241398,
  'population_per_household': 1.222858620136257,
  'rooms_per_household': 1.2302593582616217,
  'scaled_housing_median_age': 0.8232265010232409,
  'scaled_latitude': -0.5626953819091274,
  'scaled_longitude': -0.050060586107770366},
 {'above_average': 1,
  'bedrooms_per_room': 0.766241352623425,
  'log_households': 2.90687353472207,
  'log_median_income': 1.0413926851582251,
```


observation:

can have all data be easily checked for each specific information.

In [139]:

```
val_dict
```

Out[139]:


```
[{'above_average': 0,
  'bedrooms_per_room': 0.7414917185153788,
  'log_households': 2.2624510897304293,
  'log_median_income': 0.6020599913279624,
  'log_population': 2.7307822756663893,
  'log_total_bedrooms': 2.2810333672477277,
  'log_total_rooms': 3.076276255404218,
  'ocean_proximity': 'INLAND',
  'other_income_source': 96697,
  'population_per_household': 1.207001684174203,
  'rooms_per_household': 1.359709506812258,
  'scaled_housing_median_age': 0.5053941867127075,
  'scaled_latitude': 0.5188155937502427,
  'scaled_longitude': -0.13990411516798287},
 {'above_average': 0,
  'bedrooms_per_room': 0.8024168021441694,
  'log_households': 2.5921767573958667,
  'log_median_income': 0.47712125471966244,
```


importing libraries

In [174]:

```
from sklearn.feature_extraction import DictVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

the DictVectorizer can be followed by OneHotEncoder to complete binary one-hot encoding.

In [182]:

```
dv = DictVectorizer(sparse=False)
dv.fit(train_dict)
```

Out[182]:

```
DictVectorizer(sparse=False)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [183]:

```
X_train = dv.transform(train_dict)
X_val = dv.transform(val_dict)
```

In [142]:

```
X_train
```

Out[142]:

```
array([[ 1.        ,  0.81537314,  2.57403127, ...,  0.8232265 ,
        -0.56269538, -0.05006059],
       [ 1.        ,  0.76624135,  2.90687353, ..., -0.36864468,
        -0.88574411,  0.62376588],
       [ 0.        ,  0.79853501,  2.5289167 , ...,  0.98214266,
         1.63778149, -1.02336548],
       ...,
       [ 1.        ,  0.79079067,  2.78031731, ..., -0.84539315,
        -1.35393068,  1.28760974],
       [ 0.        ,  0.79964278,  2.54530712, ..., -1.00430931,
        -0.83892546,  0.8483747 ],
       [ 0.        ,  0.77235445,  2.33445375, ...,  0.50539419,
        -0.8061524 ,  0.62376588]])
```

In [143]:

```
X_val.shape
```

Out[143]:

```
(4128, 18)
```

log_reg.coef_ specifies the features slope that indep impact dep feature

positively and negatively.

log_reg.intercept_ specifies the feature intercept (the max value the model can take when coef_ are zeros).

In [332]:

```
model = LogisticRegression(solver="liblinear", C=1.0, max_iter=1000, random_state=42)
```

In [340]:

```
print(model.intercept_)
```

```
[-0.0994195]
```

In [283]:

```
a = print(model.coef_)
```

```
[[ 8.67013702e-02 -7.98954962e-02 -2.49163686e-01 -3.84077157e-02
  -3.00758400e-01 -2.53547625e-01 -3.14633920e-01 -1.88479311e-02
  -7.56545515e-02  2.09383324e-05 -2.02067933e-03 -2.91727445e-03
   1.82912282e-05 -1.20486629e-01 -1.26256331e-01  1.44108419e-02
  -1.96690925e-02 -1.93994221e-02]]
```

our model has 97 % accuracy of the model which means it 97% times correctly interpret the actual and predicted values.

In [149]:

```
model = LogisticRegression(solver="liblinear", C=1.0, max_iter=1000, random_state=42)
model.fit(X_train, y_train)

y_pred = model.predict(X_val)

accuracy = accuracy_score(y_val, y_pred)
print(np.round(accuracy, 2))
```

```
0.97
```

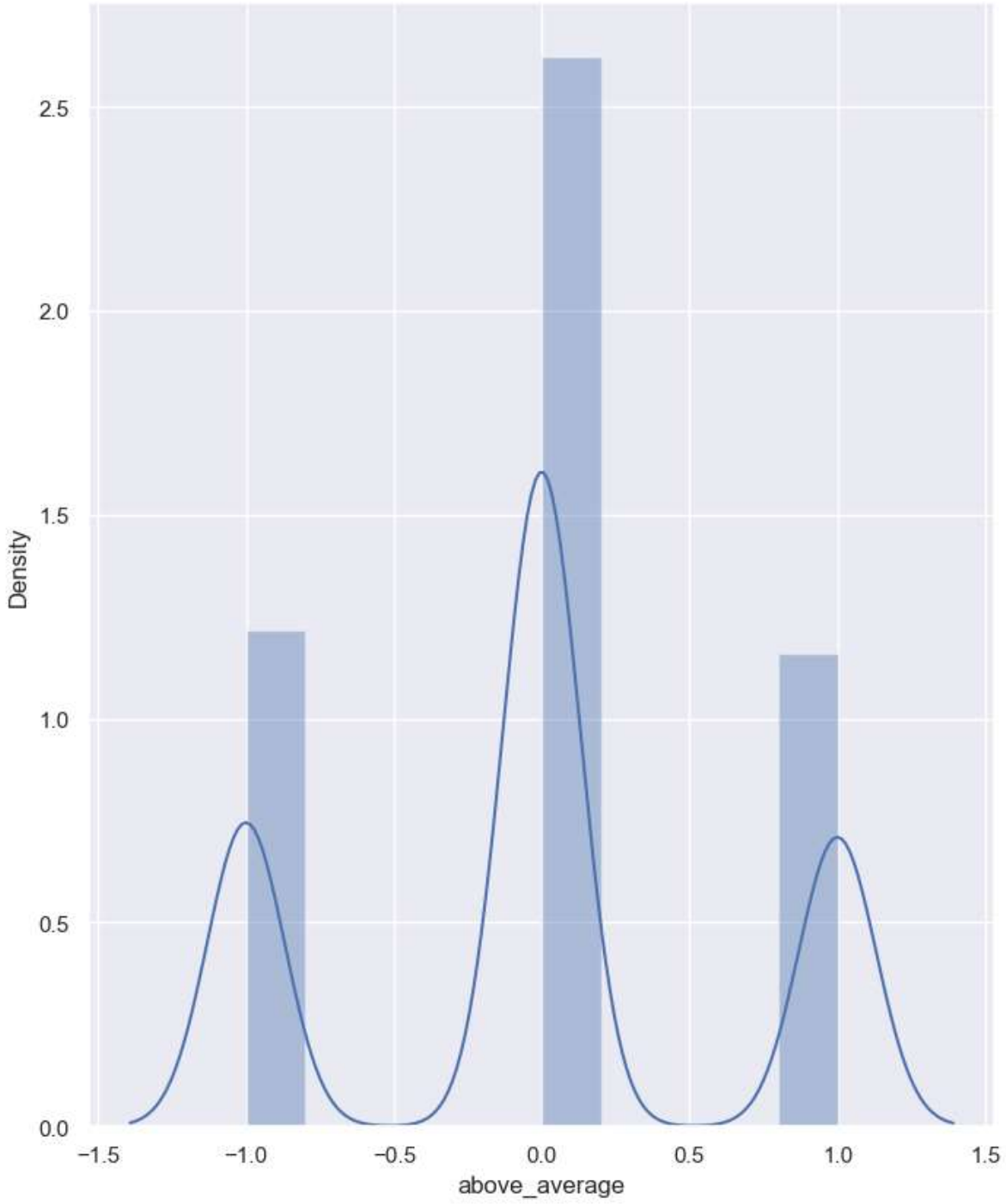
graphical analysis of logistic regression

In [229]:

```
### the residuals are of multimodal distribution.
### it is somewhat normally distributed in the middle.
residuals = y_test - y_pred
sns.distplot(residuals, bins = 10)
```

Out[229]:

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



In [154]:

```
from sklearn.datasets import make_classification
```

In [279]:

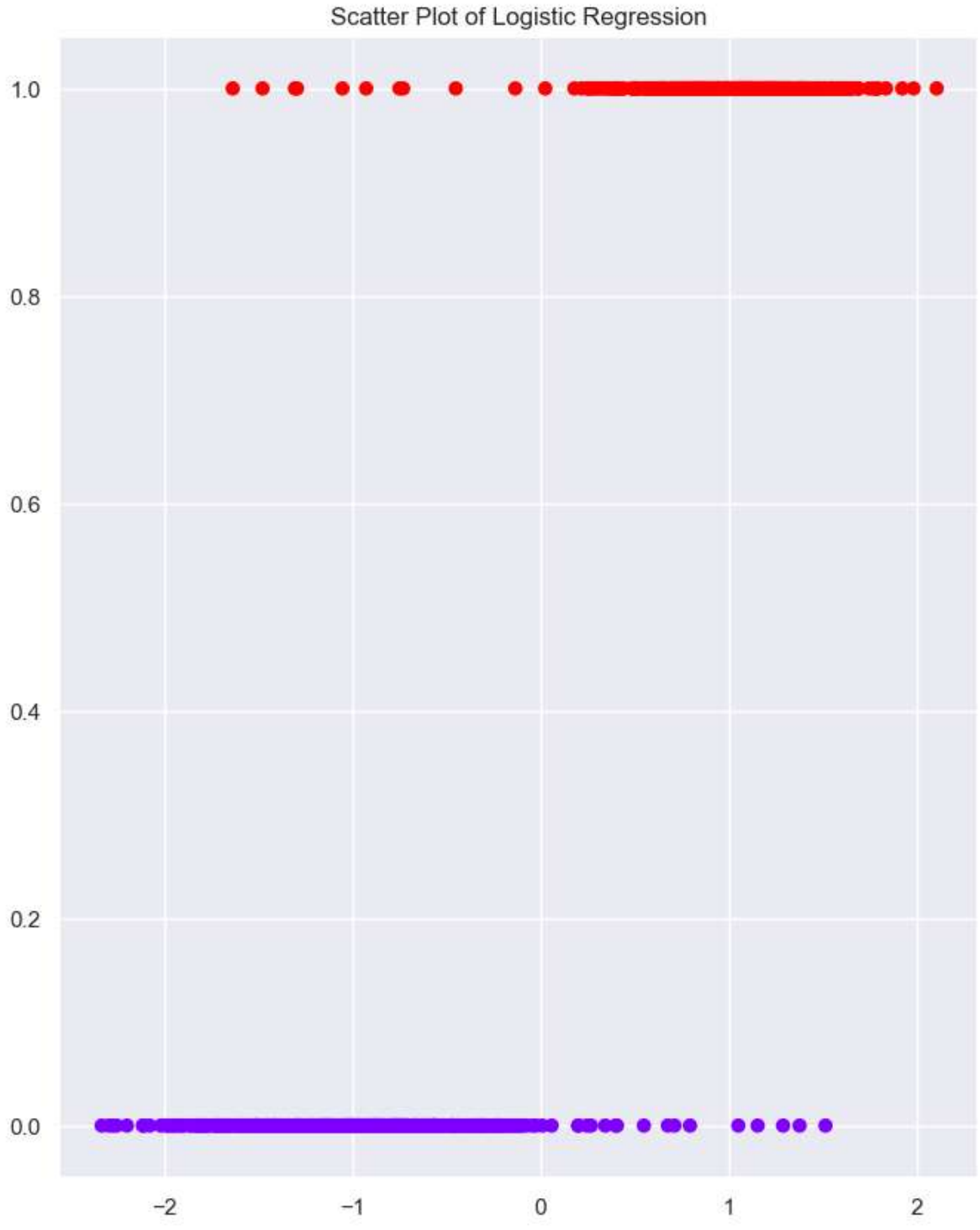
```
x, y = make_classification(
    n_samples=100,
    n_features=1,
    n_classes=2,
    n_clusters_per_class=1,
    flip_y=0.03,
    n_informative=1,
    n_redundant=0,
    n_repeated=0
)
print(y)
```

```
[0 0 1 1 1 0 1 0 1 1 0 0 0 1 0 0 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1 1 1 1 1 0
 1 0 0 0 1 1 1 0 0 0 0 1 1 0 1 0 0 0 1 1 1 1 0 1 1 0 0 1 1 1 0 0 1 0 1 1 1
 1 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 0 0 0 0 0]
```

logistic regression also tries not to focused on outliers so it makes them straight line.

In [156]:

```
## red colour is for x (1 values of above average) and
## purple colour for y values(o values of above average)
plt.scatter(x, y, c=y, cmap='rainbow')
plt.title('Scatter Plot of Logistic Regression')
plt.show()
```



observation - values of 0 are higher than values of 1.

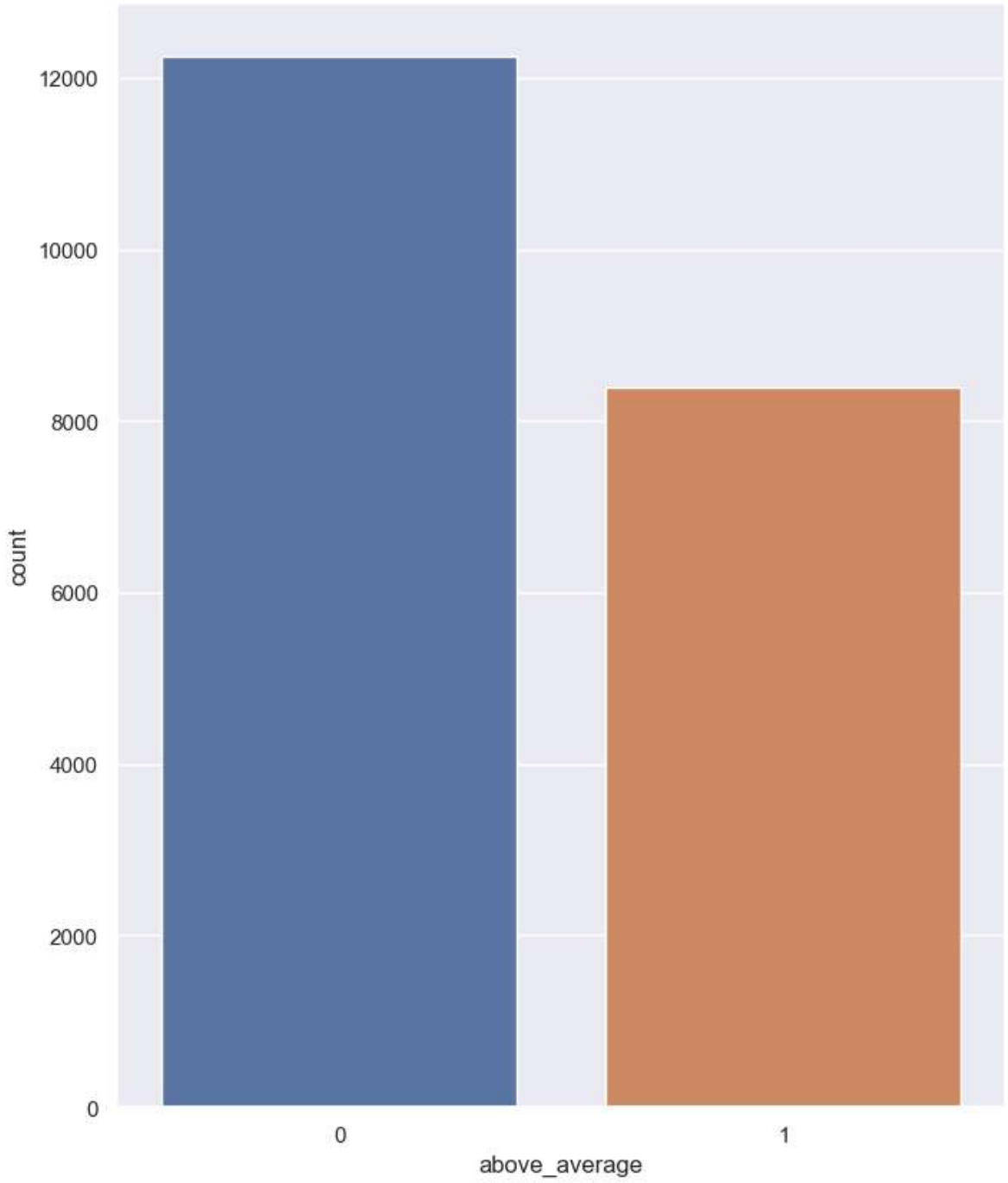
if not visible clearly, let us interpret the same result via countplot of above average feature.

In [157]:

```
sns.countplot(x = 'above_average', data = data_cleaned)
```

Out[157]:

```
<AxesSubplot:xlabel='above_average', ylabel='count'>
```



observation- the counts of 0 are more than 1200 whereas for the counts 0f 1 are more than 8000 but less than 8500.

the data is imbalace and need to be balanced (done later on)

A..... Performance metric of Logistic Regression

1. confusion matrix

2. precision score

3. recall score

4. F1 scaore

A.1 confusion matrix

In [158]:

```
from sklearn.metrics import confusion_matrix
```

In [159]:

```
confusion_matrix = confusion_matrix(y_test,y_pred)
```

In [160]:

```
confusion_matrix
```

Out[160]:

```
array([[1471, 1005],
       [ 957,  695]], dtype=int64)
```

observation : the data tested 1471 times true and predicted values as 0 most of the times correctly or true positive.

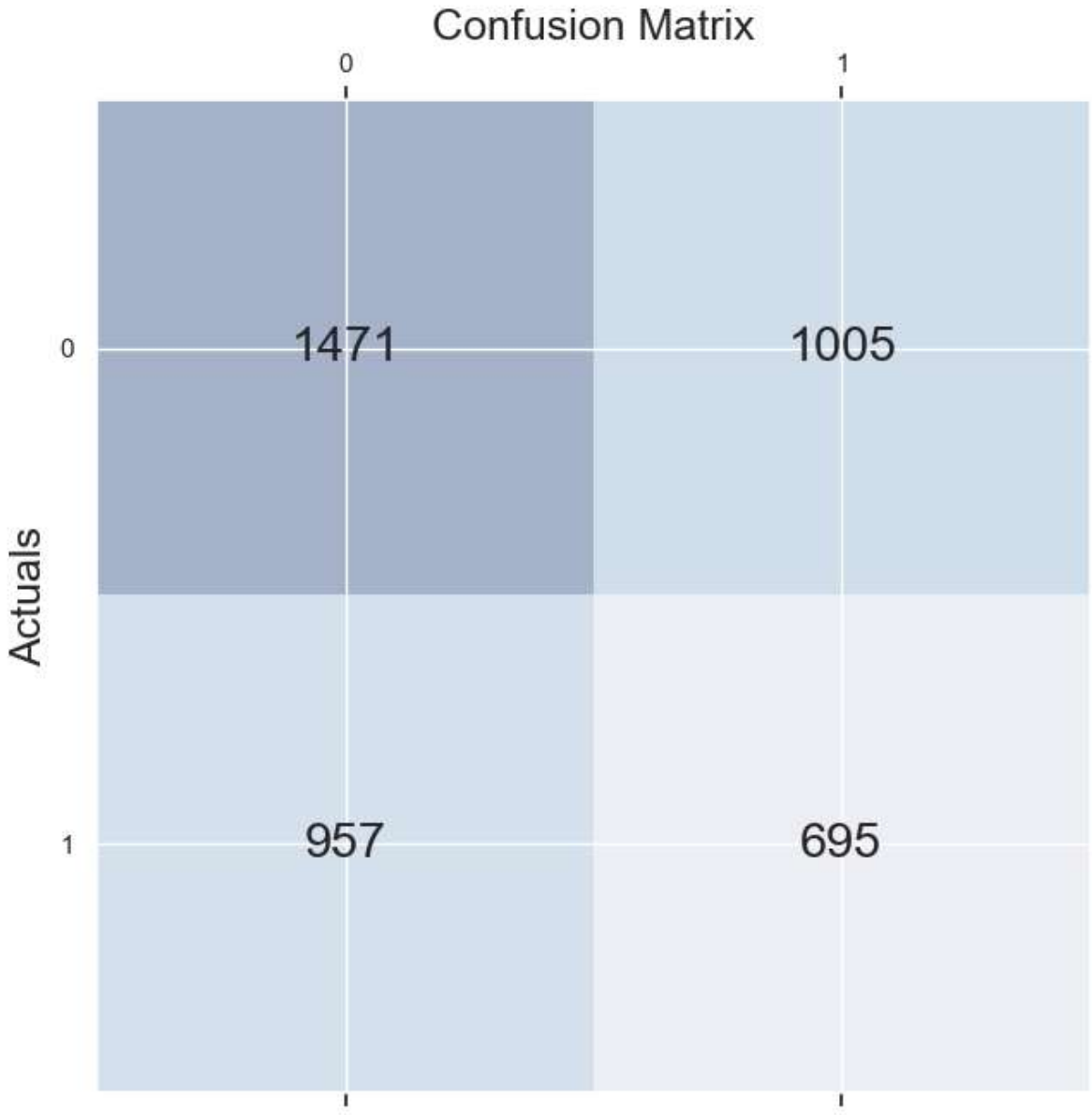
```
                ## the least times (695) the true negative occurs, that is actual va
    lue is 1 and model also predicted 1.
```

observation : it is showing that the actual data if have 0 values than model also predicted 0 1471 times.

the data that's why has more accuracy.

In [162]:

```
fig,ax = plt.subplots(figsize = (7.5,7.5))
ax.matshow(confusion_matrix, cmap = plt.cm.Blues, alpha =0.3)
for i in range (confusion_matrix.shape[0]):
    for j in range (confusion_matrix.shape[1]):
        ax.text(x=j , y=i, s = confusion_matrix[i,j], va= 'center', ha = 'center', size = '
#plt.xlabel('Predictions', fontsize=18)
plt.ylabel('Actuals', fontsize = 18)
plt.title('Confusion Matrix', fontsize = 18)
plt.show()
```



importing libraries where we can directly find out the ather performance matrix options.

i will calculate manually over here via using formula

In [163]:

```
from sklearn.metrics import precision_recall_curve
from sklearn.metrics import plot_precision_recall_curve
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
```

exctracting values of matrix column from confusion matrix

In [164]:

```
true_positive = confusion_matrix[0][0]
false_positive = confusion_matrix[0][1]
false_negative = confusion_matrix[1][0]
```

observation - even after 97% accuracy the model is calculating very low recall, precison and F1 score. why?

F1 score will provide a better assessment of model performance (due to balanced and imbalanced the result changes; F1 score)

recall and precision separately gives clear picture of actual positive and proportion of positive.

A.2 Precision score

precison measures the proportion of positively predicted labels that are actually correct.

In [285]:

```
precision = true_positive/(true_positive + false_positive)
precision
```

Out[285]:

```
0.5941033925686591
```

A.3 Recall score

Model recall score represents the model's ability to correctly predict the positives out of actual positives.

This is unlike precision which measures how many predictions made by models are actually positive out of all positive predictions made.

In [166]:

```
recall = true_positive/(true_positive + false_negative)
```

In [167]:

```
recall
```

Out[167]:

```
0.6058484349258649
```

A.4 F1 score

F-score is a machine learning model performance metric

```
    ### that gives equal weight to both the Precision and Recall for measuring its p
   erformance in terms of accuracy.
```

In [168]:

```
F1 = (2 *(precision * recall))/ (precision + recall)
```

In [169]:

```
F1
```

Out[169]:

```
0.5999184339314845
```

classification report

In [201]:

```
from sklearn.metrics import classification_report
```

In [171]:

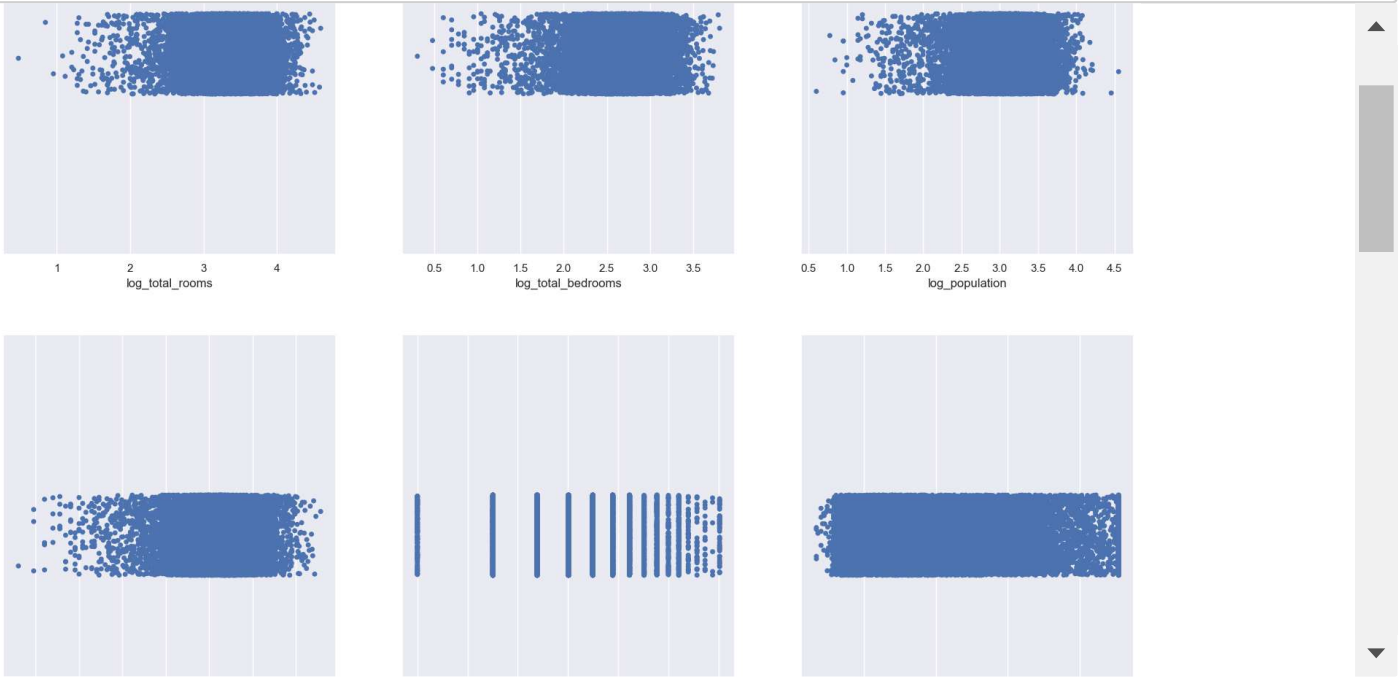
```
print(classification_report (y_test,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.61      0.59      0.60      2476
           1       0.41      0.42      0.41      1652

    accuracy                           0.52      4128
   macro avg       0.51      0.51      0.51      4128
weighted avg       0.53      0.52      0.53      4128
```

```
### strip plot of each numerical feature which gives clear picture of
 ### uniform, outliers and distribution of the data.
```

In [315]:

```
plt.figure(figsize = (20,50))
for i in enumerate(numerical_features):
    plt.subplot(6,3, i[0]+1)
    sns.set(rc={'figure.figsize':(8,10)})
    sns.stripplot(x= i[1] , data = data_cleaned)
plt.show()
```



data is imbalanced and we need to again set the model by balancing it.

In [203]:

```
data_cleaned
```

Out[203]:

	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

data segregation for dependent (y) and independent variable (x).

In [210]:

```
y = data_bal['above_average']
```

In [211]:

```
y
```

Out[211]:

```
0        1
1        1
2        1
3        1
4        1
       ..
20635    0
20636    0
20637    0
20638    0
20639    0
Name: above_average, Length: 20640, dtype: int64
```

In [207]:

```
x = data_bal.iloc[:,:-1]
```

In [148]:

```
x
```

Out[148]:

	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 × 10 columns

In [209]:

```
x = x.drop(columns = ['ocean_proximity'])
```

balacing the data via using oversampling technique by using SMOTE.

In [158]:

```
# Oversample and plot imbalanced dataset with SMOTE
from collections import Counter
from sklearn.datasets import make_classification
from imblearn.over_sampling import SMOTE
from matplotlib import pyplot
from numpy import where
# define dataset
X, y = make_classification(n_samples=20640, n_features=2, n_redundant=0,
    n_clusters_per_class=1, weights=[0.99], flip_y=0, random_state=1)
# summarize class distribution
counter = Counter(y)
print(counter)
# transform the dataset
oversample = SMOTE()
X, y = oversample.fit_resample(x, y)
# summarize the new class distribution
counter = Counter(y)
print(counter)
```

```
Counter({0: 20434, 1: 206})
Counter({0: 20434, 1: 20434})
```

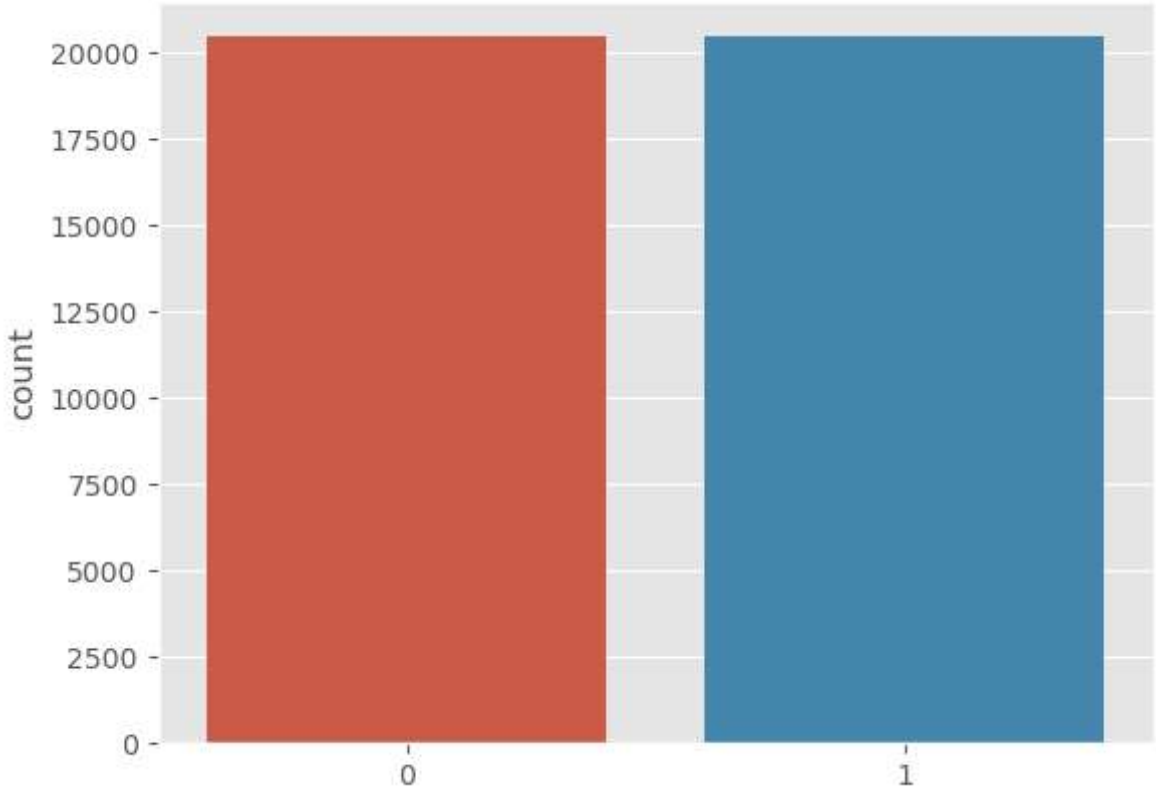
now the data is balanced which was earlier not.

In [159]:

```
sns.countplot(y , data = data_cleaned)
```

Out[159]:

```
<AxesSubplot:ylabel='count'>
```



In [213]:

```
X.shape
```

Out[213]:

```
(40868, 9)
```

In [163]:

```
y.shape
```

Out[163]:

```
(40868,)
```

transforming the data for testing and training.

In [185]:

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

In [264]:

```
X_train1_scaled=scaler.fit_transform(X_train1)
X_test1_scaled=scaler.fit(X_test1)
```

testing and training the new balanced data

In [134]:

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

In [214]:

```
X_train1, X_test1, y_train1, y_test1 = train_test_split(
    X_bal, y_bal, test_size=0.15, random_state=10)
```

In [215]:

```
len(X_train1), len(X_test1), len(y_train1), len(y_test1)
```

Out[215]:

```
(34724, 6128, 34724, 6128)
```

using SMOTE for fitting the sample

In [160]:

```
from imblearn.combine import SMOTETomek
```

In [161]:

```
smt = SMOTETomek()
```

In [165]:

```
X_bal,y_bal = smt.fit_resample(X,y)
```

again setting the threshold for binary classification of the data in order to perform logistic regression

In [141]:

```
data_bal['above_average'] = data_bal['scaled_median_house_value'].apply(lambda x: 1 if x >
```

In [142]:

```
ohe
```

Out[142]:

	<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

joining the one hot encoding (ohe) for categorical feature with data cleaned for performing logistic regression.

In [140]:

```
data_bal = data_cleaned.join(pd.DataFrame(ohe),lsuffix='_left', rsuffix='_right')
```

In [217]:

```
data_bal
```

Out[217]:

	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 × 20 columns

checking the shape of x_bal and y_bal created from SMOTE.

In [166]:

```
### it is a dataframe.
X_bal.head()
```

Out[166]:

	log_total_rooms	log_total_bedrooms	log_population	log_households	log_median_income	s
0	2.944976	2.113943	2.509203	2.103804	0.954243	
1	3.851258	3.044148	3.380573	3.056524	0.954243	
2	3.166726	2.281033	2.696356	2.250420	0.903090	
3	3.105510	2.372912	2.747412	2.342423	0.778151	
4	3.211654	2.448706	2.752816	2.414973	0.602060	

In [167]:

```
### it is a series.
y_bal
```

Out[167]:

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

both x_bal and y_bal has equal length data.

In [221]:

```
len(X_bal), len(y_bal)
```

Out[221]:

```
(39158, 39158)
```

again checking and segregating the numerical and categorical feature for new dataset , that is data_cleaned.

In [222]:

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

```
we have ['log_total_rooms', 'log_total_bedrooms', 'log_population', 'log_hou
seholds', 'log_median_income', 'scaled_median_house_value', 'scaled_housing_
median_age', 'scaled_latitude', 'scaled_longitude', 'other_income_source',
'above_average', 'rooms_per_household', 'bedrooms_per_room', 'population_per
_household'] as our numerical feature
```

In [223]:

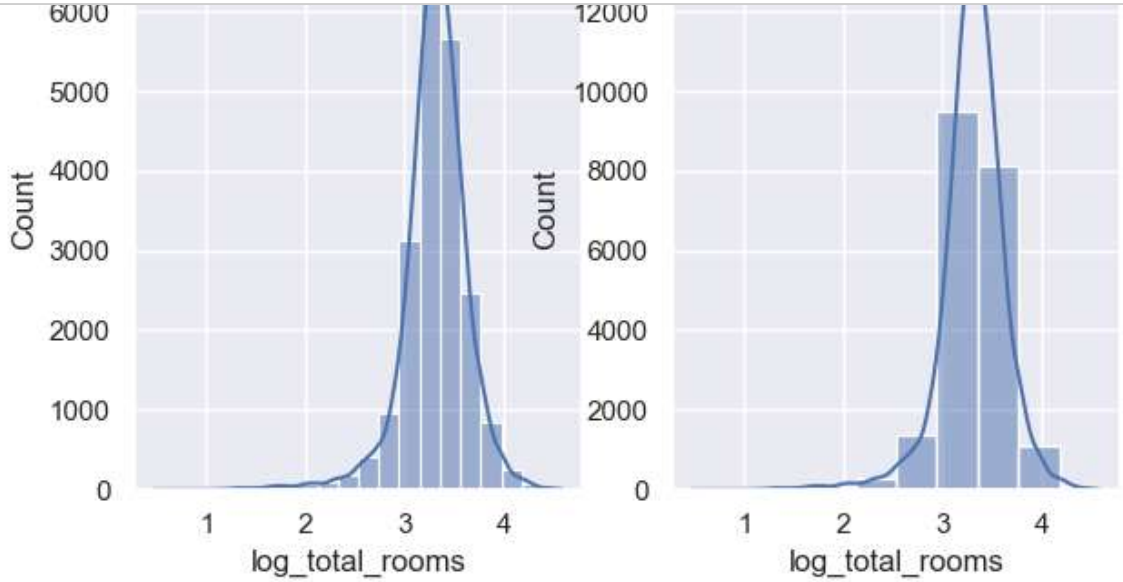
```
categorical_features = [fea for fea in data_cleaned.columns if data_cleaned[fea].dtypes ==
print(f'we have {categorical_features} as our categorical feature')
```

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

checking and comparing the distribution of the data for new data_bal and old data_cleaned data.

In [224]:

```
for i in numerical_features:
    plt.figure(figsize= (7,4))
    plt.subplot(121)
    sns.histplot(data = data_cleaned , x=i, kde = True, bins = 20)

    plt.subplot(122)
    sns.histplot(data = data_bal , x=i, kde = True, bins = 10)
```



basic description of the data like covariance and description of the data.

observation:

for our target feature has good and positive realtion between above average , median age of house and income.

In [225]:

```
data_bal.cov().T
```

Out[225]:

	log_total_rooms	log_total_bedrooms	log_population	log_household
log_total_rooms	0.106092	0.096294	0.090063	0.0957
log_total_bedrooms	0.096294	0.098614	0.089679	0.0961
log_population	0.090063	0.089679	0.102236	0.0940
log_households	0.095782	0.096125	0.094086	0.0996
log_median_income	0.014852	0.001114	0.002109	0.0029
scaled_median_house_value	0.051928	0.016662	-0.006780	0.0232
scaled_housing_median_age	-0.102674	-0.085060	-0.078104	-0.0766
scaled_latitude	-0.010527	-0.021335	-0.043633	-0.0277
scaled_longitude	0.009556	0.019008	0.034862	0.0174
other_income_source	5991.952232	1922.740213	-782.386634	2681.0299
above_average	0.024578	0.009632	-0.000974	0.0130
rooms_per_household	-0.009494	-0.013416	-0.015535	-0.0161
bedrooms_per_room	0.004633	0.007766	0.006470	0.0070
population_per_household	-0.011996	-0.012126	-0.005281	-0.0120
<1H OCEAN	0.003511	0.006480	0.018452	0.0114
INLAND	-0.002104	-0.006540	-0.010816	-0.0129
ISLAND	-0.000038	-0.000009	-0.000064	-0.0000
NEAR BAY	-0.001665	-0.001650	-0.005964	-0.0005
NEAR OCEAN	0.000296	0.001720	-0.001608	0.0020

In [226]:

```
data_bal.describe().T
```

Out[226]:

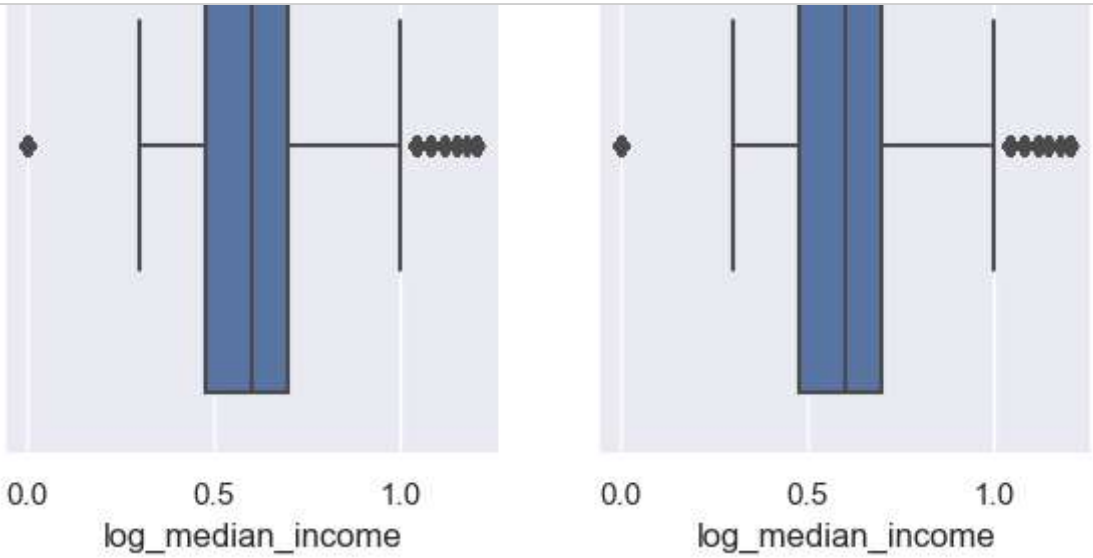
	count	mean	std	min	25%
log_total_rooms	20640.0	3.313396e+00	0.325717	0.477121	3.160993
log_total_bedrooms	20640.0	2.629626e+00	0.314029	0.301030	2.474216
log_population	20640.0	3.051137e+00	0.319744	0.602060	2.896526
log_households	20640.0	2.599123e+00	0.315619	0.301030	2.448706
log_median_income	20640.0	6.049060e-01	0.181709	0.000000	0.477121
scaled_median_house_value	20640.0	8.950635e-16	1.000024	-1.662641	-0.756163
scaled_housing_median_age	20640.0	8.557001e-16	1.000024	-2.196180	-0.845393
scaled_latitude	20640.0	1.256263e-15	1.000024	-1.447568	-0.796789
scaled_longitude	20640.0	-6.527810e-15	1.000024	-2.385992	-1.113209
other_income_source	20640.0	2.068524e+05	115394.305074	14995.000000	119598.000000
above_average	20640.0	4.062500e-01	0.491144	0.000000	0.000000
rooms_per_household	20640.0	1.281022e+00	0.078692	0.941589	1.241660
bedrooms_per_room	20640.0	7.922364e-01	0.037487	0.315465	0.771423
population_per_household	20640.0	1.178560e+00	0.073632	0.861353	1.144911
<1H OCEAN	20640.0	4.426357e-01	0.496710	0.000000	0.000000
INLAND	20640.0	3.173934e-01	0.465473	0.000000	0.000000
ISLAND	20640.0	2.422481e-04	0.015563	0.000000	0.000000
NEAR BAY	20640.0	1.109496e-01	0.314077	0.000000	0.000000
NEAR OCEAN	20640.0	1.287791e-01	0.334963	0.000000	0.000000

checking and comparing that after data bal how our new and old data get affected in outliers.

```
### it is visible that most of the data presention is same and does not affect mu
ch after overfitting, which is a good part.
```

In [234]:

```
for i in numerical_features:
    plt.figure(figsize= (7,4))
    plt.subplot(121)
    sns.boxplot(data = data_cleaned , x=i)

    plt.subplot(122)
    sns.boxplot(data = data_bal , x=i)
```



In [172]:

```
X_train1
```

Out[172]:

	log_total_rooms	log_total_bedrooms	log_population	log_households	log_median_incom
27465	3.791589	3.034771	3.492416	3.023570	0.78589
19098	3.892595	3.143327	3.550473	3.143951	0.69897
3279	3.340841	2.695482	2.832509	2.570543	0.30103
28466	3.456084	3.024585	3.597984	2.985618	0.45580
25749	3.220452	2.486348	2.909821	2.466251	0.53967
...	...	...	...	...	.
40059	3.120349	2.405702	2.833513	2.366285	0.47712
28017	3.933407	3.248530	3.629213	3.209348	0.57733
29199	2.678920	1.951674	2.462416	1.942577	0.63437
40061	3.307346	2.592652	3.037910	2.567259	0.66754
17673	3.271144	2.562293	3.263873	2.615950	0.77815

34724 rows × 9 columns

The class DictVectorizer can be used to convert

feature arrays represented as lists of standard Python dict objects to the NumPy/SciPy

```
### representation used by scikit-learn estimators
```

In [191]:

```
train_dict = X_train1.to_dict(orient='records')
val_dict = X_test1.to_dict(orient='records')
```

In [185]:

```
dv = DictVectorizer(sparse=False)
dv.fit(train_dict)
```

Out[185]:

```
DictVectorizer(sparse=False)
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

transforming the data using DictVectorizer.

In [186]:

```
X_train1 = dv.transform(train_dict)
X_test1 = dv.transform(val_dict)
```

checking the length of these transformed data, clearly visible the length has reduced.

In [187]:

```
X_test1.shape
```

Out[187]:

```
(6128, 9)
```

In [197]:

```
y_test1.shape
```

Out[197]:

```
(6128,)
```

checking the coefficients slope for each slope feature

```
### target feature : house value is negative which means that people buy less if t
he value of house is high and vice versa.
```

In [383]:

```
print(list(zip(data_bal.columns.tolist(),model.coef_[0])))
```

```
[('ocean_proximity', 0.08670137017032359), ('log_total_rooms', -0.0798954962
3669007), ('log_total_bedrooms', -0.24916368559787155), ('log_population', -
0.03840771574432033), ('log_households', -0.30075839976242413), ('log_median
_income', -0.2535476249854106), ('scaled_median_house_value', -0.31463391990
50268), ('scaled_housing_median_age', -0.018847931144951265), ('scaled_latit
ude', -0.0756545514804287), ('scaled_longitude', 2.0938332389282295e-05),
('other_income_source', -0.002020679334205198), ('above_average', -0.0029172
744490678034), ('rooms_per_household', 1.8291228186399753e-05), ('bedrooms_p
er_room', -0.12048662913744133), ('population_per_household', -0.12625633120
008578), ('<1H OCEAN', 0.014410841920801243), ('INLAND', -0.0196690924518007
94), ('ISLAND', -0.019399422122915982)]
```

maximum value of the model can take if all the slopes value are zero.

In [386]:

```
print(list(zip(data_bal.columns.tolist(),model.intercept_)))
```

```
[('ocean_proximity', -0.09941949807626368)]
```

the accuracy of the model after balacing the data is low (only 55%), calculate the the true value most of the times.

In [188]:

```
model = LogisticRegression(solver="liblinear", C=1.0, max_iter=10000, random_state=42)

model.fit(X_train1, y_train1)

y_pred1 = model.predict(X_test1)

accuracy = accuracy_score(y_test1, y_pred1)
print(np.round(accuracy, 4))
```

```
0.5496
```

In [249]:

```
model.fit(X_train1, y_train1)
```

Out[249]:

```
LogisticRegression(max_iter=10000, random_state=42, solver='liblinear')
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

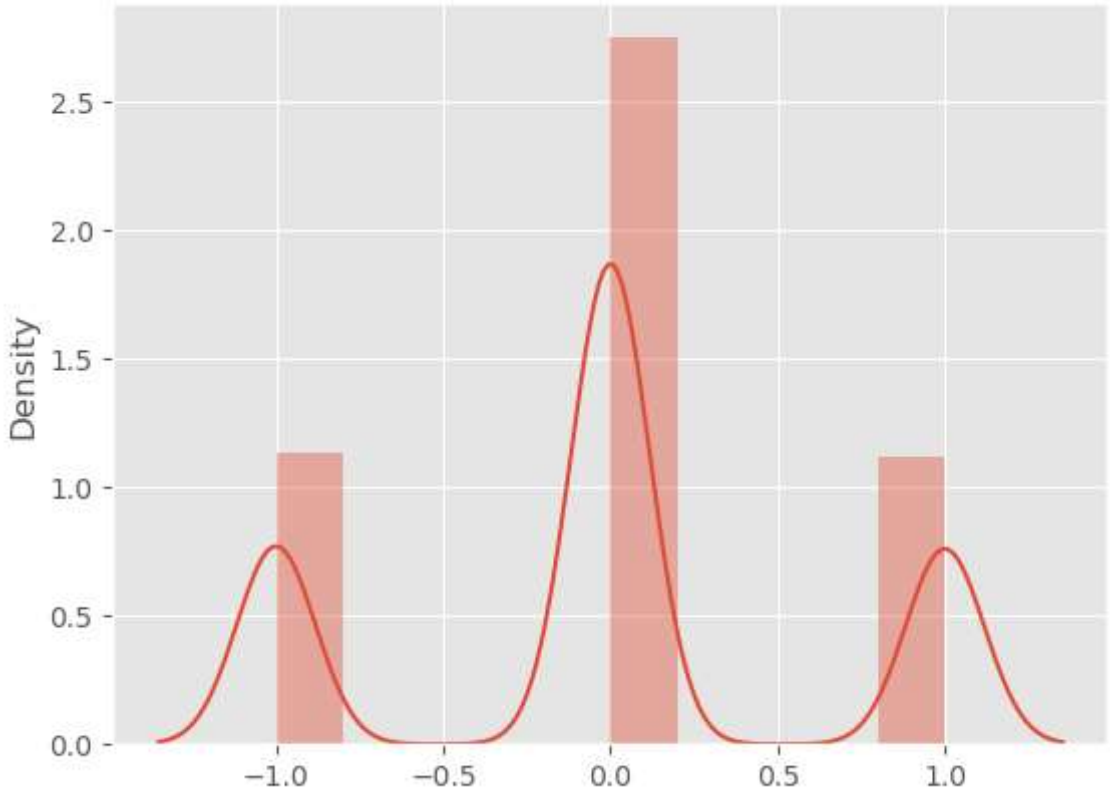
the model has residuals in multimodal distribution and centre one is quite normally distributed.

In [194]:

```
### the residuals are of multimodal distribution.
### it is somewhat normally distributed in the middle.
residuals = y_test1 - y_pred1
sns.distplot(residuals, bins = 10)
```

Out[194]:

```
<AxesSubplot:ylabel='Density'>
```



classification report saying how true values are predicted for 0 and 1.

In [202]:

```
print(classification_report (y_test1,y_pred1))
```

```
              precision    recall  f1-score   support

           0       0.55      0.54      0.55      3050
           1       0.55      0.55      0.55      3078

    accuracy                           0.55      6128
   macro avg       0.55      0.55      0.55      6128
weighted avg       0.55      0.55      0.55      6128
```

conclusion:

with imbalanced data or original data (the model prediction is more accurate) as the accuracy is 97%.

with balancing the data the data accuracy falls and even fails to predict correct most of the true values (only 54%).

logistic regression squased the outliers and make them in thr straight line.

our target feature is median house value for this dataset.

Binary classification of above average feature is on the high and low mean of traget feature.

```
  ### which means high mean high house value and then how population and household
will react (where to buy house
            ###    and how many rooms per household, etc.,) and vice versa.
```

observation:

```
### simple linear regression is best for california housing house with R square an
d adjusted R square as 99%.
### if logistic is to be used than orginial data is to be used as accuracy is 97%.
```

---------- end of logistic regression model----------

In [ ]: