California Housing Price

By Prashant DHILLON

IMC-4302C – Statistical learning Project

The aim of this project is to predict median housing prices in California

Table of Contents

Importing Libraries and Reading Data

In [1]:

```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor

from sklearn import metrics
```

About data

The data contains 20,640 observations on 10 variables. This dataset contains the average house value as target variable and the following input variables (features): average income, housing average age, average rooms, average bedrooms, population, average occupation, latitude, longitude and ocean proximity

Below data loading and a dataset header is shown

In [2]:

```
# Reading the dataset
df=pd.read_csv("housing.csv")
print(df.shape)
df.head()
```

```
(20640, 10)
```

Out[2]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	household
0	-122.23	37.88	41.0	880.0	129.0	322.0	126
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259

EDA

Below we will perform some Exploratory Data Analysis to do some initial investigations on our dataset, so that we can discover patterns,to spot anomalies and to check assumptions with the help of summary statistics and graphical representations.

Actually, we are going to spend most of time in this section to explain features and fact findings of dataset. Bascially, from this step we will know what is the relation between our variables and how it can be modelled and solved through Machine Learning problem

In [3]:

```
df.describe()
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	popu
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.(
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.4
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.4
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.(
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.(
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.(
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.(
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.(

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
```

The housing prices are continuous and hence it a regression problem. (ocean_proximity is a categorical feature)

Distribution of data

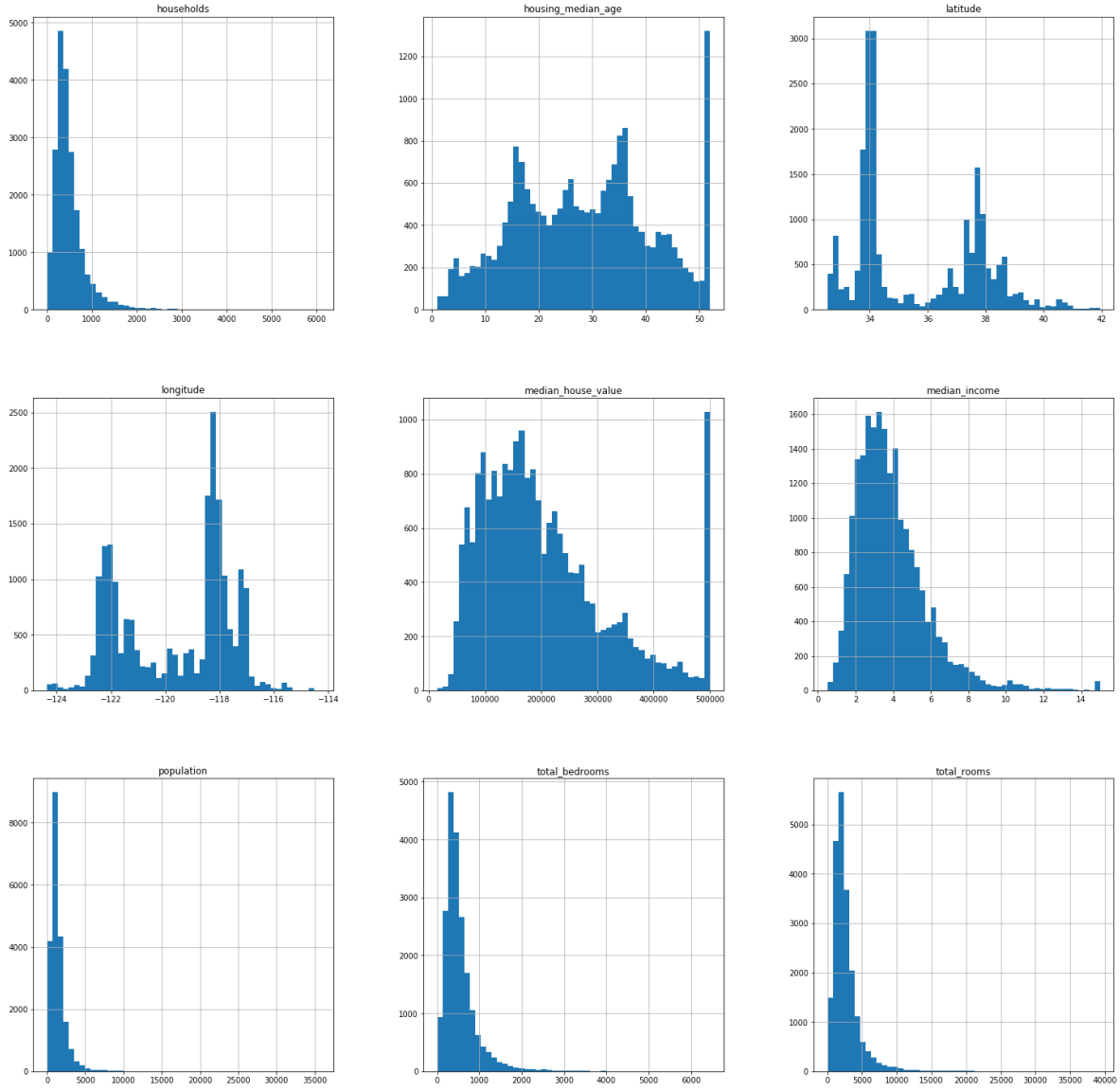
To visualize the distribution of data we can use Histrogram plotting of our each coloumn i.e., each feature. Histogram is a representation of the distribution of data

In [5]:

```
df.hist(figsize=(25,25), bins=50)
```

Out[5]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f323d03191
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cfb565
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cf6ee5
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f323cf2f69
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cee2e9
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cbe66d
0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f323cb9bed
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cb5d71
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f323cb6729
0>]],
      dtype=object)
```



This shows us that all our features are not evenly distributed. Some observed to be very skewed and discrete.

Pairwise Correrlation of Columns

As we can see that it is a regression problem, because we need to predict median price.

We can find correlations using pandas ".corr()" function and can visualize the correlation matrix using a heatmap in seaborn.

To have an idea about what's happening below, Correlation is a measure of the strength of a linear relationship between two quantitative variables. Below two terms are used to inference the results positive correlation and negative correlation. that are,

*Positive correlation is a relationship between two variables in which both variables move in the same direction. This is when one variable increases while the other increases and visa versa.

*Negative correlation is a relationship where one variable increases as the other decreases, and vice versa.

In [6]:

```
df_corr = df.corr()
df_corr.style.background_gradient()
```

Out[6]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms
longitude	1.000000	-0.924664	-0.108197	0.044568	0.069608
latitude	-0.924664	1.000000	0.011173	-0.036100	-0.066983
housing_median_age	-0.108197	0.011173	1.000000	-0.361262	-0.320451
total_rooms	0.044568	-0.036100	-0.361262	1.000000	0.930380
total_bedrooms	0.069608	-0.066983	-0.320451	0.930380	1.000000
population	0.099773	-0.108785	-0.296244	0.857126	0.877747
households	0.055310	-0.071035	-0.302916	0.918484	0.979728
median_income	-0.015176	-0.079809	-0.119034	0.198050	-0.007723
median_house_value	-0.045967	-0.144160	0.105623	0.134153	0.049686

In the Below correlation plot,

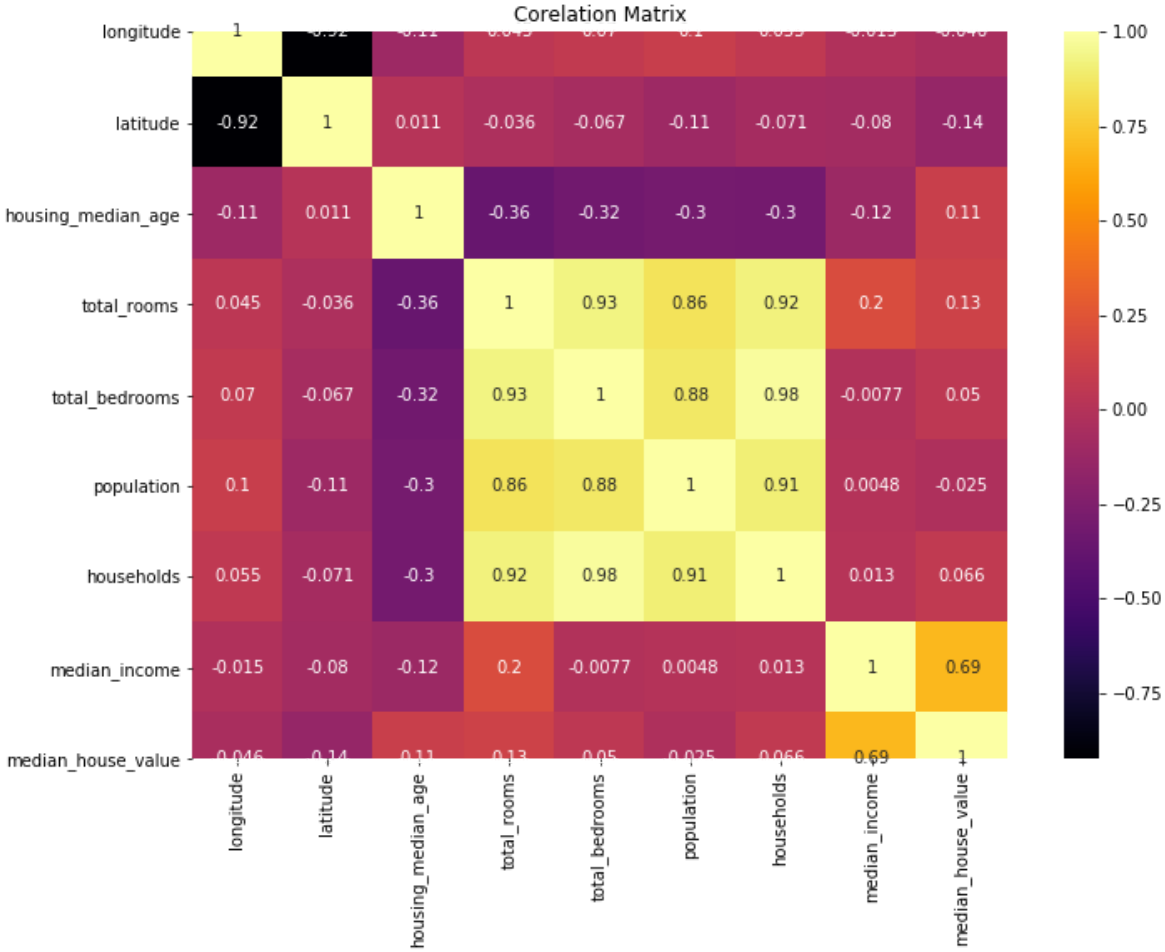
- Dark shades represents positive correlation while lighter shades represents negative correlation.
- If we set annot=True, we'll get values by which features are correlated to each other in grid-cells.

In [7]:

```
fig, axe = plt.subplots(figsize=(12,8))
sns.heatmap(data=df_corr,square =True, annot=True,cmap= 'inferno' )
plt.title('Corelation Matrix')
```

Out[7]:

```
Text(0.5, 1, 'Corelation Matrix')
```



Now let's analyse some findings from above plot

Here we can infer that "Median House Value" has strong positive correlation with "Median Income".

"Median House Value" and "Median Income" has almost no correlation with "Households". Since correlation is zero we can infer there is no linear relationship between these two predictors.However it is safe to drop these features while applying Linear Regression model to the dataset.

Remove the Outliers

An outlier is a data point that differs significantly from other observations. An outlier may be due to variability in the measurement or it may indicate experimental error

As we have seen abnormality in data by above graphical methods, we have to remove these outliers because most parametric statistics, like means, standard deviations, and correlations, and every statistic based on these, are highly sensitive to outliers. Since we are going to use linear regression it's gonna affect the results

Below is the function to remove outliers based on upper and lower bound concept

In [8]:

```
def removeOutliers(dataframe,column):
    column = "total_rooms"
    des = dataframe[column].describe()
    desPairs = {"count":0,"mean":1,"std":2,"min":3,"25":4,"50":5,"75":6,"max":7}
    Q1 = des[desPairs['25']]
    Q3 = des[desPairs['75']]
    IQR = Q3-Q1
    lowerBound = Q1-1.5*IQR
    upperBound = Q3+1.5*IQR
    print("(IQR = {})Outlier are anything outside this range: ({},{})".format(IQR,l
    data = dataframe[(dataframe [column] < lowerBound) | (dataframe [column] > uppe

    print("Outliers out of total = {} are \n {}".format(df[column].size,len(data[co
    outlierRemoved = df[~df[column].isin(data[column])]
    return outlierRemoved
```

In [9]:

```
df_outliersRemoved = removeOutliers(df,"total_rooms")
```

```
(IQR = 1700.25)Outlier are anything outside this range: (-1102.625,569
8.375)
Outliers out of total = 20640 are
 1287
```

Missing values

check wheather there are any missing values or null

In [10]:

```
df.isnull().sum()
```

Out[10]:

```
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
```

As we can see only total_bedrooms column has 207 missing values, lets treat it. There are 2 ways to treat missing values

1. We can delete those records which are missing (Not Recommended because it can delete other coloumn vital information)

2. or we can fill those columns using the mean or median - which in this case is a pretty much easier.

But what should we be using Mean or Median, So to decide this we need to first check the outliers

Statistics for missing values

In [11]:

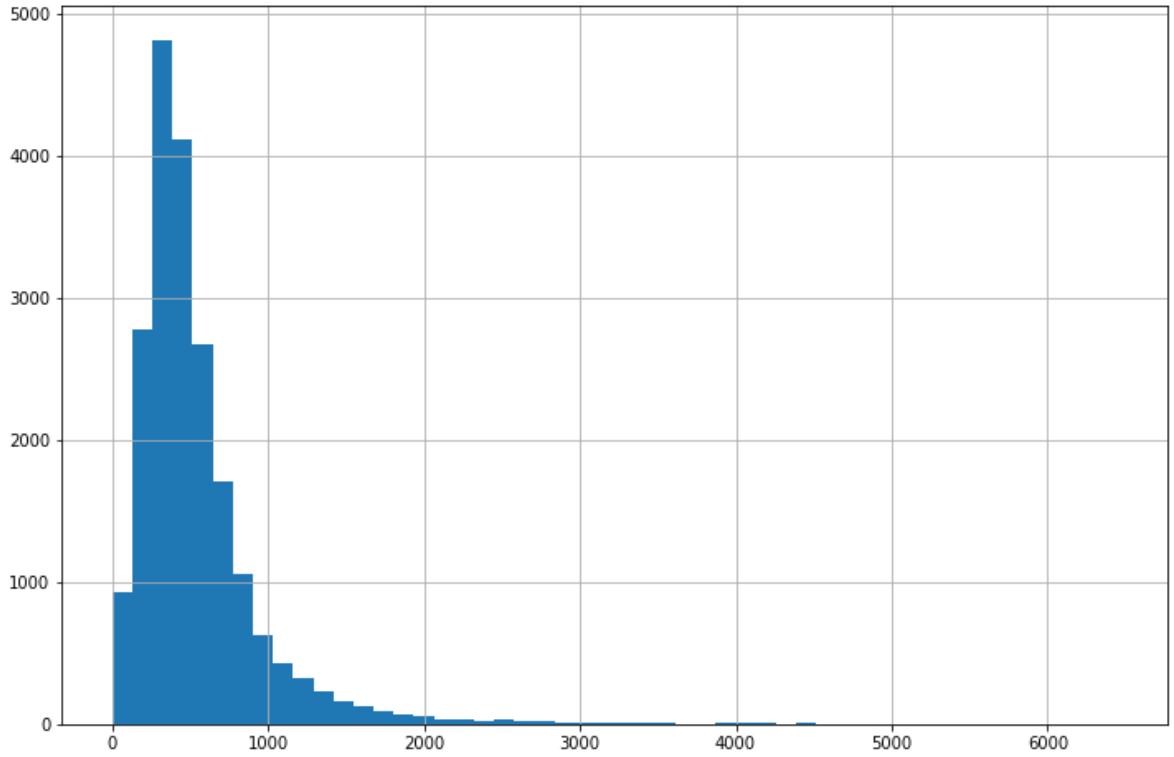
```
print ("Total_bedrooms column Mode is  "+str(df["total_bedrooms"].mode())+"\n")
print(df["total_bedrooms"].describe())
```

```
Total_bedrooms column Mode is  0    280.0
dtype: float64

count    20433.000000
mean       537.870553
std        421.385070
min          1.000000
25%        296.000000
50%        435.000000
75%        647.000000
max       6445.000000
Name: total_bedrooms, dtype: float64
```

In [12]:

```
total_bedroms = df[df["total_bedrooms"].notnull()]["total_bedrooms"]
total_bedroms.hist(figsize=(12,8),bins=50);
```



The data has a lot of missing values and hence filling with mean will affect the prediction. We will impute the missing values using median

In [13]:

```
print(df.iloc[:,4:5].head())
imputer = SimpleImputer(np.nan,strategy ="median")
imputer.fit(df.iloc[:,4:5])
df.iloc[:,4:5] = imputer.transform(df.iloc[:,4:5])
df.isnull().sum()
```

```
   total_bedrooms
0           129.0
1          1106.0
2           190.0
3           235.0
4           280.0
```

Out[13]:

```
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
```

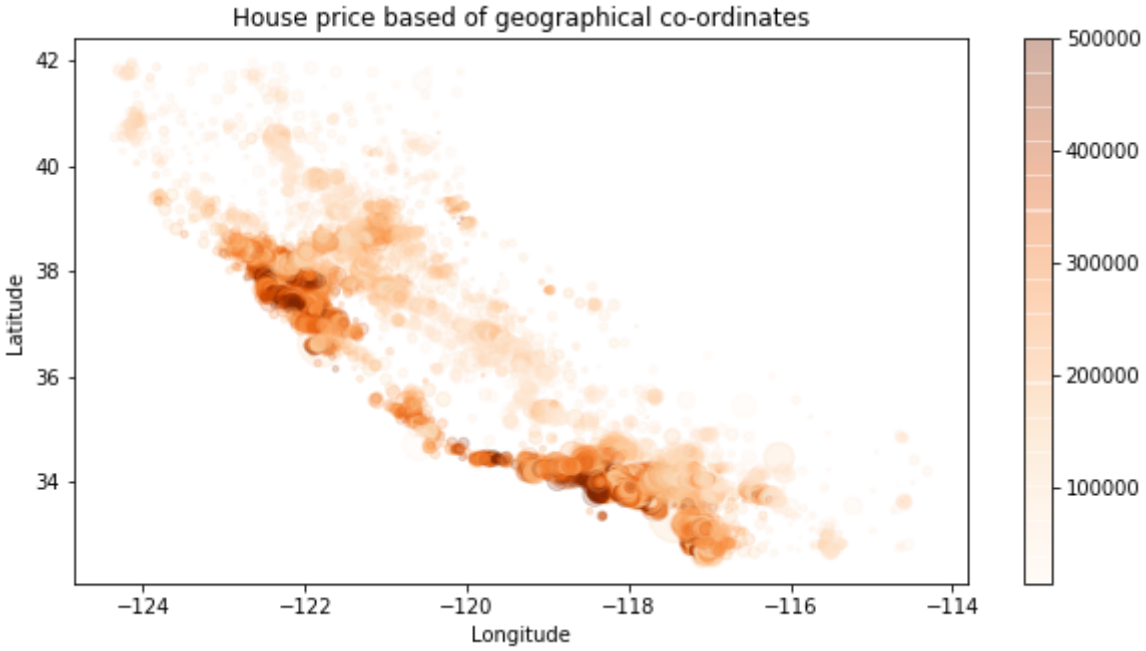
Since we have some geographical data, lets see if get some meaning insights from it

In [14]:

```
plt.figure(figsize=(10,5))
plt.scatter(df["longitude"],df["latitude"],c=df["median_house_value"],
            s=df["population"]/50, alpha=0.2, cmap="Oranges")
plt.colorbar()
plt.xlabel("Longitude")
plt.ylabel("Latitude")
plt.title("House price based of geographical co-ordinates")
```

Out[14]:

```
Text(0.5, 1.0, 'House price based of geographical co-ordinates')
```



One clear inference we can observe by looking at this plot, there are some high density areas in california, so we can say the price of house is a bit realted to location as well

We have one categorical column ("Ocean Proximity") in the data set, lets vizualize it

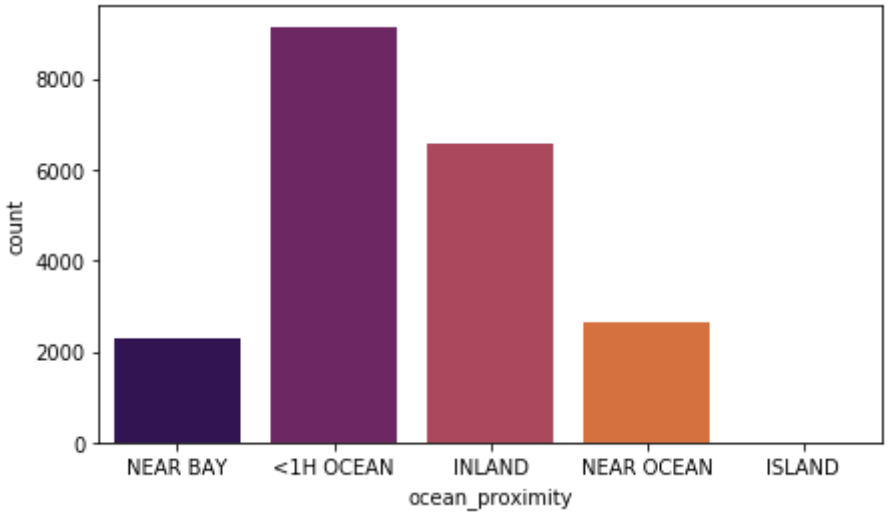
Barplot of categorical column

In [15]:

```
plt.figure(figsize=(7,4))
sns.countplot(data=df,x='ocean_proximity', palette = "inferno")
```

Out[15]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f32397836d0>
```



From this plot we can see value counts of different category of houses like there is a few for Island, Maximum for <1H Ocean and around 2000 for Near Bay

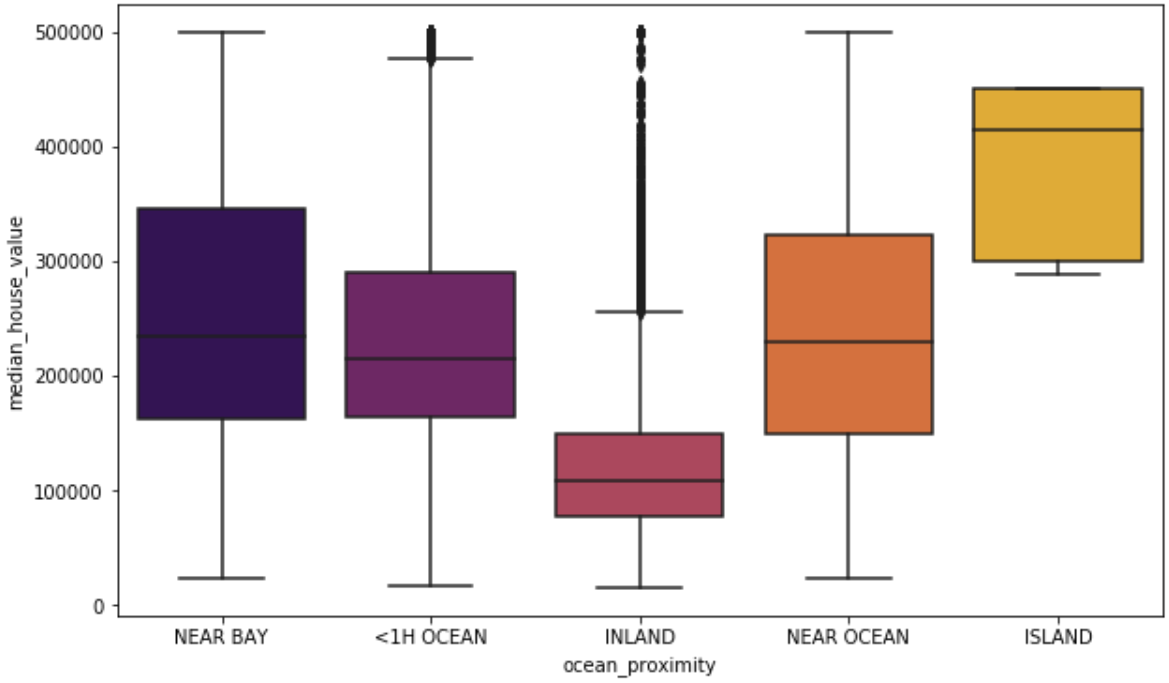
Boxplot of house value on ocean_proximity categories

In [16]:

```
plt.figure(figsize=(10,6))
sns.boxplot(data=df,x='ocean_proximity',y='median_house_value',palette='inferno')
plt.plot()
```

Out[16]:

```
[]
```



Observations: *A clear inference can be taken from this plot that island houses are most expensive and inland houses are least expensive. While Near Bay and Near Ocean houses are most popular and fairly expensive than inland*

Label encode for categorical feature (ocean_proximity)

In [17]:

```
labelEncoder = LabelEncoder()
print(df["ocean_proximity"].value_counts())
df["ocean_proximity"] = labelEncoder.fit_transform(df["ocean_proximity"])
df["ocean_proximity"].value_counts();
```

```
<1H OCEAN     9136
INLAND        6551
NEAR OCEAN    2658
NEAR BAY      2290
ISLAND           5
Name: ocean_proximity, dtype: int64
```

Split the dataset into 80% train and 20% test dataset

We use train_test_split function from sklearn model selection for splitting data arrays into two subsets:

- for training data
- for testing data

By default, Sklearn train_test_split will make random partitions for the two subsets. However, you can also specify a random state for the operation for reproducinng same results.

In [18]:

```
df_ind = df.drop("median_house_value",axis=1)
df_dep = df["median_house_value"]
X_train,X_test,y_train,y_test = train_test_split(df_ind,df_dep,test_size=0.2,random
print("X_train shape {} and size {}".format(X_train.shape,X_train.size))
print("X_test shape {} and size {}".format(X_test.shape,X_test.size))
print("y_train shape {} and size {}".format(y_train.shape,y_train.size))
print("y_test shape {} and size {}".format(y_test.shape,y_test.size))
```

```
X_train shape (16512, 9) and size 148608
X_test shape (4128, 9) and size 37152
y_train shape (16512,) and size 16512
y_test shape (4128,) and size 4128
```

Standardize the data

Feature scaling is to bring all the independent variables in a dataset into same scale, to avoid any variable dominating the model. Here we will not transform the dependent variables.

Sklearn's StandardScaler() standardize features by removing the mean and scaling to unit variance The standard score of a sample x is calculated as:
z = (x - u) / s
where u is the mean of the training samples or zero if with_mean=False, and s is the standard deviation of the training samples or one if with_std=False.

In [19]:

```
independent_scaler = StandardScaler()
X_train = independent_scaler.fit_transform(X_train)
X_test = independent_scaler.transform(X_test)
```

Perform Linear Regression

LinearRegression fits a linear model with coefficients w = (w1, …, wp) to minimize the residual sum of squares between the observed targets in the dataset, and the targets predicted by the linear approximation.

- Perform Linear Regression on training data.
- Predict output for test dataset using the fitted model.
- Print root mean squared error (RMSE) from Linear Regression.

In [20]:

```
#initantiate the linear regression
linearRegModel = LinearRegression(n_jobs=-1)
#fit the model to the training data (learn the coefficients)
linearRegModel.fit(X_train,y_train)
#print the intercept and coefficients
print("Intercept is "+str(linearRegModel.intercept_))
print("coefficients  is "+str(linearRegModel.coef_))
```

```
Intercept is 207194.69373788778
coefficients  is [-85854.94724101 -90946.06271148  14924.30655143 -176
93.23405277
  48767.60670995 -43884.16852449  17601.31495096  77144.10164179
   -451.52015229]
```

In [21]:

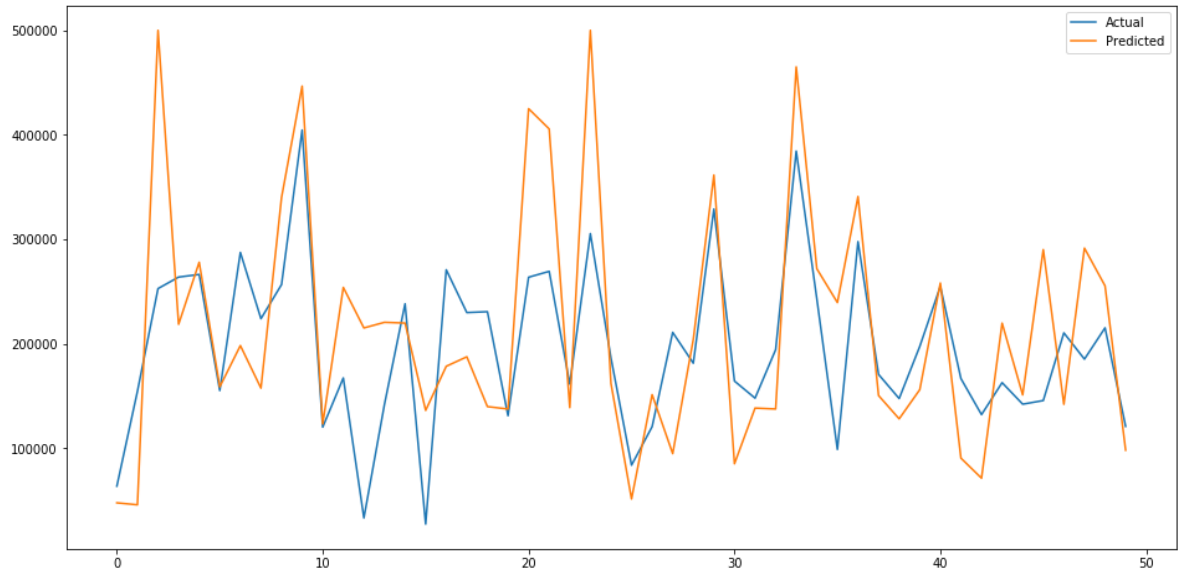
```
#predict on the test data
y_pred = linearRegModel.predict(X_test)
```

In [22]:

```
test = pd.DataFrame({'Predicted':y_pred,'Actual':y_test})
fig= plt.figure(figsize=(16,8))
test = test.reset_index()
test = test.drop(['index'],axis=1)
plt.plot(test[:50])
plt.legend(['Actual','Predicted'])
```

Out[22]:

```
<matplotlib.legend.Legend at 0x7f32395a2f10>
```



Root Mean Squared Error (RMSE)

In [23]:

```
print('Root mean square error for test data is : {}'
      .format(np.sqrt(metrics.mean_squared_error(y_test,y_pred))))

print('Root mean square error for train data is : {}'
      .format(np.sqrt(metrics.mean_squared_error(y_train,linearRegModel.predict(X_t
```

```
Root mean square error for test data is : 71147.87146118375
Root mean square error for train data is : 69361.0714290645
```

Score

The coefficient R^2 is defined as (1 - u/v), where u is the residual sum of squares ((y_true - y_pred) ** 2).sum() and v is the total sum of squares ((y_true - y_true.mean()) ** 2).sum(). The best possible score is 1.0 and it can be negative (because the model can be arbitrarily worse). A constant model that always predicts the expected value of y, disregarding the input features, would get a R^2 score of 0.0.

We calculate the score between X_train and y_train because the library internally uses lr.predict(X_train) to get y_train'

In [24]:

```
# print the R-squared value for the model
print('Linear Regression Model Score: {}'.format(linearRegModel.score(X_train,y_tra
```

```
Linear Regression Model Score: 0.6401079709888613
```

Perform Random Forest Regression

- Perform Random Forest Regression on training data.
- Predict output for test dataset using the fitted model.
- Print RMSE (root mean squared error) from Random Forest Regression.

In [25]:

```
rfReg = RandomForestRegressor(30)
rfReg.fit(X_train,y_train)
```

Out[25]:

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_no
des=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.
0,
                      n_estimators=30, n_jobs=None, oob_score=False,
                      random_state=None, verbose=0, warm_start=False)
```

In [26]:

```
rfReg_y_pred = rfReg.predict(X_test)
print(np.sqrt(metrics.mean_squared_error(y_test,rfReg_y_pred)))
```

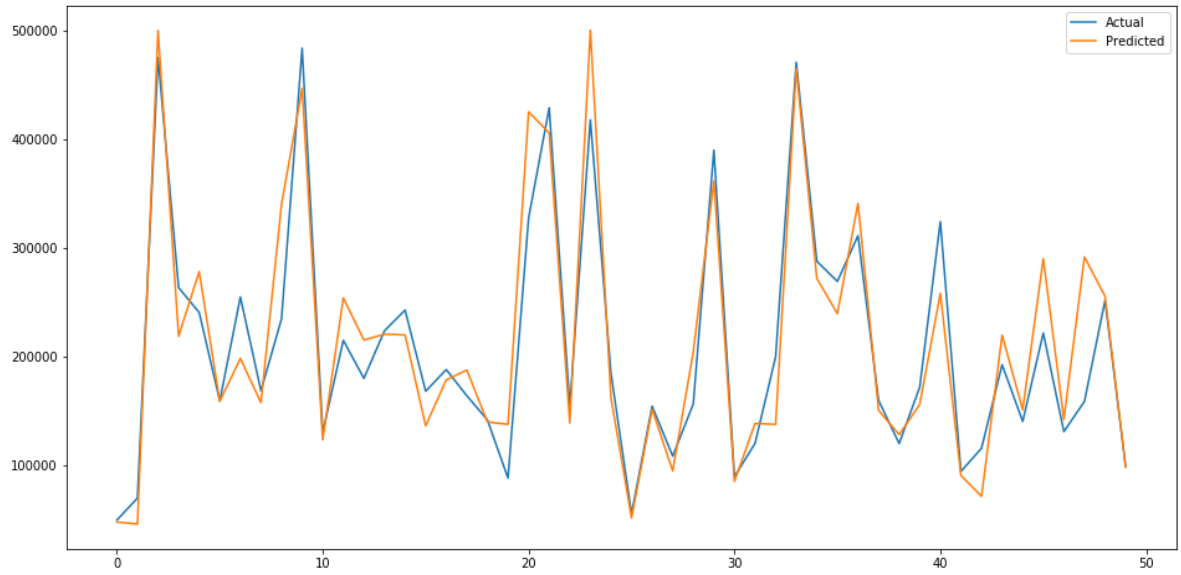
```
50660.68220381428
```

In [27]:

```
test = pd.DataFrame({'Predicted':rfReg_y_pred,'Actual':y_test})
fig= plt.figure(figsize=(16,8))
test = test.reset_index()
test = test.drop(['index'],axis=1)
plt.plot(test[:50])
plt.legend(['Actual','Predicted'])
```

Out[27]:

```
<matplotlib.legend.Legend at 0x7f3238515150>
```



RMSE

In [28]:

```
print('Root mean square error for test date is : {}'
      .format(np.sqrt(metrics.mean_squared_error(y_test,rfReg_y_pred))))

print('Root mean square error for train date is : {}'
      .format(np.sqrt(metrics.mean_squared_error(y_train,rfReg.predict(X_train)))))
```

```
Root mean square error for test date is : 50660.68220381428
Root mean square error for train date is : 19304.387103449433
```

Score

In [29]:

```
# print the R-squared value for the model
print('Random Forest Model Score: {}'.format(rfReg.score(X_train,y_train)))
```

```
Random Forest Model Score: 0.9721225736510937
```

Conclusion

In this project we did the exploratory data analysis of the California Housing Dataset. We got rid to the outliers and filled the missing values. We plotted different plots to understand the dataset. We found that some of the features of the data are highly skewed like median income and total bedroom.

We implemented two different regression techniques to predict housing prices, the linear regression and the random forest regression. We found that the random forest regression is performing better than the linear regression because it has lower RMSE and higher r^2 value(score).