PROBLEM STATEMENT: The task is to build a model of housing prices in California using the California census data. This data has metrics such as the population, median income, median housing price, and so on for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). We will just call them "districts" for short. Our model should learn from this data and be able to predict the median housing price in any district, given all the other metrics.

DATASET: The Data consists of 10 attributes and 20640 instances. The dataset looks like below

```
In [7]: dataset.head()
Out[7]:
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximity
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR BAY
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR BAY
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR BAY
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR BAY
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR BAY

The attributes are longitude, latitude, housing_median_age, total_rooms, total_bedrooms, population, households, median_income, median_house_value and ocean_proximity.

Now we will take a look at information about these attributes

```
In [8]: dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20433 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

If we look at the above figure all the attributes are numerical expect the ocean_proximity. And also, the total_bedrooms attribute is having some null values we have to deal with them to avoid errors.

Let's look at the ocean_proximity attribute alone

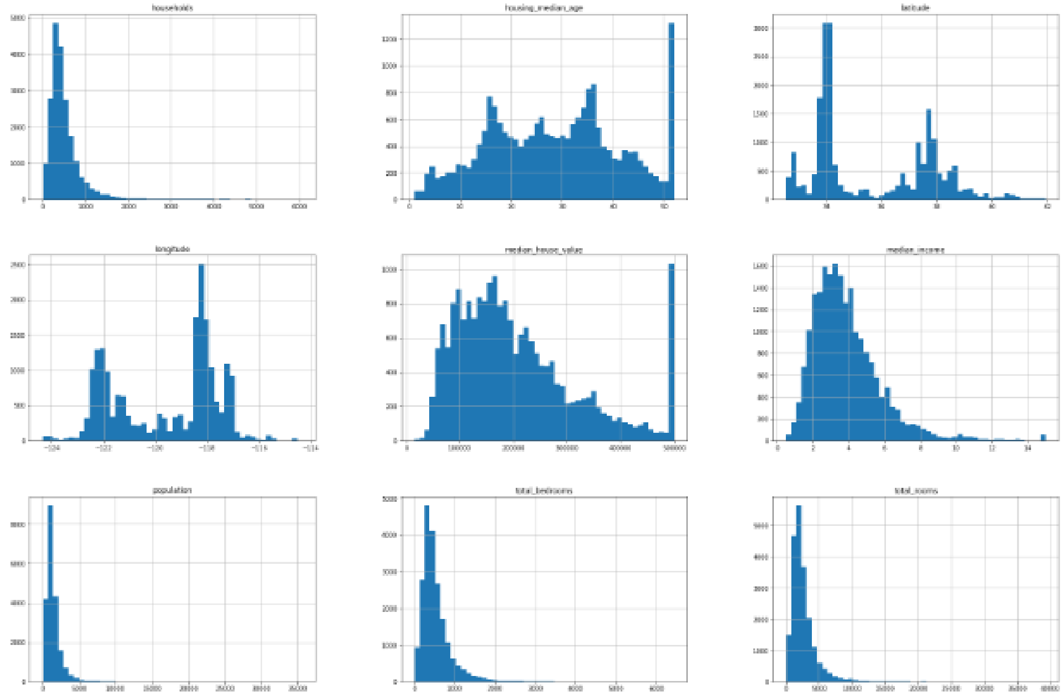
```
In [9]: dataset['ocean_proximity'].value_counts()

Out[9]: <1H OCEAN     9136
        INLAND        6551
        NEAR OCEAN    2658
        NEAR BAY      2290
        ISLAND           5
        Name: ocean_proximity, dtype: int64
```

The ocean_proximity attribute is having 5 categories and we can see the count of each category in the above fig.

DATA VISUALIZATION:

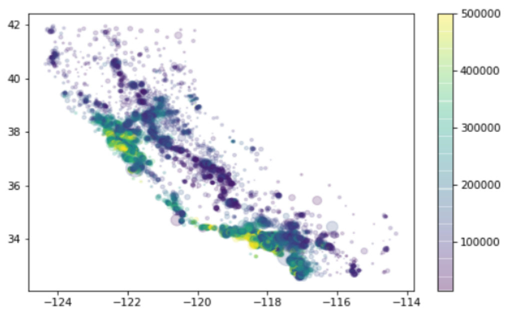
Let’s look at the all numerical attributes distribution using histogram which shows the number of instances (on the vertical axis) that have a given value range (on the horizontal axis).

```
In [11]: dataset.hist(bins=50,figsize=(30,20))

Out[11]: array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAE7BBC88>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEA53EF0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEA85198>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEAAC400>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEAD4668>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEAFD898>],
                [<matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEB27B00>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEB50DA0>,
                 <matplotlib.axes._subplots.AxesSubplot object at 0x000001CBAEB50DD8>]],
               dtype=object)
```



Now we will plot the scatter plot between longitude and latitude. The radius of each circle represents the district’s population, and the color represents the median house value.

```
In [12]: plt.figure(figsize=(8,5))
         plt.scatter(x=dataset.longitude, y=dataset.latitude,alpha=0.2,c=dataset.median_house_value,s=dataset.population/100)
         plt.colorbar()

Out[12]: <matplotlib.colorbar.Colorbar at 0x1cbaf047780>
```



HANDLING MISSING DATA:

As we seen earlier there are some missing values in the total_bedrooms attribute, now we will deal with them.

```
In [13]: median = dataset["total_bedrooms"].median()
         dataset["total_bedrooms"].fillna(median, inplace=True)
         dataset.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20640 entries, 0 to 20639
Data columns (total 10 columns):
longitude             20640 non-null float64
latitude              20640 non-null float64
housing_median_age    20640 non-null float64
total_rooms           20640 non-null float64
total_bedrooms        20640 non-null float64
population            20640 non-null float64
households            20640 non-null float64
median_income         20640 non-null float64
median_house_value    20640 non-null float64
ocean_proximity       20640 non-null object
dtypes: float64(9), object(1)
memory usage: 1.6+ MB
```

As you see the above code snippet, we have replaced the null values with the median values of the total_bedrooms. Now we don't have any missing data so we can proceed with the further analysis.

Dividing Dataset into X and Y:

Its time to split the data into Independent variables(X) and Dependent variables(Y) using the below code.

X=dataset.drop('median_house_value',axis=1)
Y=dataset['median_house_value']

Handling Categorical Variables and Dummy Variable Trap:

We have to handle categorical attribute, because machine learning model will not understand the text data so we have to convert them to dummy variables like below

```
In [15]: dummies=pd.get_dummies(X['ocean_proximity'])
         X=X.join(dummies)
         X.head()
```

Out[15]:

titude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocean_proximity	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEAR OCEAN
37.88	41.0	880.0	129.0	322.0	126.0	8.3252	NEAR BAY	0	0	0	1	0
37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	NEAR BAY	0	0	0	1	0
37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	NEAR BAY	0	0	0	1	0
37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	NEAR BAY	0	0	0	1	0
37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	NEAR BAY	0	0	0	1	0

Now the categorical variable is having either zero or one. Then there is some problem with dummy variable trap, we have to remove one column and ocean_proximity column which we will never use.

```
In [17]: X=X.drop('<1H OCEAN',axis=1)
         X=X.drop('ocean_proximity',axis=1)
```

Spliting Data into Training and Test Set:

Its time to split the data into training and test set. We will train our model on the training set and will check the performance of the model on the test set data.

```
In [19]: X_train,X_test,Y_train,Y_test= train_test_split(X,Y,test_size=0.2,random_state=143)
         print("X_Train Set",len(X_train))
         print("X_Test Set",len(X_test))
         print("Y_Train Set",len(Y_train))
         print("Y_Test Set",len(Y_test))

         X_Train Set 16512
         X_Test Set 4128
         Y_Train Set 16512
         Y_Test Set 4128
```

In the above code we have used the library of sklearn's train_test_split to split the data into train and test set.

Model Creation using Linear Regression:

Everything has been done so far, now its time to create our model.

First, we will create our model using Linear Regression, for that we need to import the LinearRegression library from sklearn.

```
In [20]: regressor=LinearRegression()
         regressor.fit(X_train,Y_train)

Out[20]: LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

We have made our first model using linear regression and we fitted the model with the training data as you can see in the above fig.

Now we will test our model on the test set data

```
In [21]: Y_pred=regressor.predict(X_test)
         Y_pred

Out[21]: array([168594.84520048, 292322.74291219, 131587.83185124, ...,
                113453.93174537, 232017.62419617, 237401.86830014])
```

Our model has been tested on the test set data now we will check the performance of the model as below.

```
In [22]: score=r2_score(Y_test,Y_pred)
         print(score*100)
         print(round(score*100),"% Accuracy")

         63.94433518894762
         64.0 % Accuracy
```

To check the performance of the model we have to import the r2_score from sklearn.meterics.

We will check the r2_score of Y_test and Y_pred data.

As you can see that we got 64% accuracy to our model which is not that bad but we have to improve our score.

Till now we have deal the problem with linear approach, lets look at the other side.

Model Creation using Random Forest (Non-Linear Model):

Now we will create our second model using Random Forest Regression which is Non-Linear model.

```
In [23]: randreg=RandomForestRegressor(n_estimators=500,random_state=143)
         randreg.fit(X_train,Y_train)
         yrand=randreg.predict(X_test)

In [34]: yrand

Out[34]: array([182654.802, 321836.202, 231693.754, ..., 133997.2  , 189351.   ,
                271304.   ])

In [35]: score1=r2_score(Y_test,yrand)
         print(score1*100)
         print(round(score1*100),"% Accuracy")

         81.90210676988397
         82.0 % Accuracy
```

We have fitted the model and tested on the test data. The performance of the model also checked by using r2_score.

As we can see clearly in the above fig we got 82% accuracy which is way better than the previous linear model.