```
print("Welcome to Machine Learning")
```

```
Welcome to Machine Learning
```

```
import pandas as pd
```

```
df = pd.read_csv("housing.csv.zip")
```

```
df.head()
```

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

```
# Steps of Building a ML Model from scratch
1. Getting the data (from online, csv, cloud, etc)
2. Analyse the data - this is crucial- we find patterns/ insights/ other relevant ideas to work with dataset
3. Visualization - sometimes optional(as sometimes its cleaned or need cleaning of data)
4. Core ML practices --
-- 1. split our data into input and output - X (input features), Y (o/p features)
-- 2. split the data into Training and Testing dataset (the model only sees the training data, testing data is completely new data to
-- 3. Import the neccesary modeling liberaries - linearRegression/ multi linear/ random forest etc from sklearn
-- 4. Use the model on my training data/ fiiting my data to the model
-- 5. Make prdictions on new data(Test data)- after getting teh predictions
-- 6. Evaluate the performance of my model by comparing the actual/obeserved data and the predicted data by the model

-- We baiscally will build a model, evaluate its performance, then based on performance we take a decision if we have to continue with

-- Things not working out coz:
- 1. data - darbage in = garbage out
- 2. Wrong model type. Linear Regression/DecisionTree Regression/XBG Regressor....
- 3. Relationship between i/p and o/p is not linear - you are trying to fit a linear model to them -FAIL- relationship and model are r
- 4. Model should be capale to handel the comlexity in the data
```

```
# sklearn - is the ML liberaries of Pthon
```

```
df.head()
```

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house
0	-122.23	37.88	41.0	880.0	129.0	322.0	126.0	8.3252	45
1	-122.22	37.86	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	35
2	-122.24	37.85	52.0	1467.0	190.0	496.0	177.0	7.2574	35
3	-122.25	37.85	52.0	1274.0	235.0	558.0	219.0	5.6431	34
4	-122.25	37.85	52.0	1627.0	280.0	565.0	259.0	3.8462	34

```
# columns which are not needed, just drop them from your df. This is Preprocessing part
df.drop(columns= ['longitude', 'latitude'], inplace=True)
```

```
# I also have another colmn 'ocean_proximity' which is in categorical format. since this cannot be understood by models we have to cor
df['ocean_proximity'].value_counts()
```

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

```
# in the above we see that this coumn have 5 different values. So now we will do a categorical transformation on this column
# 'getdummies' functioncan transform the categorical data to corresponding numeric data into '0' and '1'
```

```
pd.get_dummies(df['ocean_proximity'])
```

	<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

```
# now we want to remove 'ocean_proximity' and add the transformed data to our df. we use concate function
pd.concat([df, pd.get_dummies(df['ocean_proximity'])])
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_pr
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NE
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NE
2	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NE
3	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NE
4	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NE
...	...	...	...	...	...	...	...	
20635	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
20636	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
20637	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
20638	NaN	NaN	NaN	NaN	NaN	NaN	NaN	
20639	NaN	NaN	NaN	NaN	NaN	NaN	NaN	

41280 rows × 13 columns

```
newdf= pd.concat([df, pd.get_dummies(df['ocean_proximity'])], axis=1)    #.. here we concanated or df and newly converted colmn
```

```
newdf.head()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	ocean_proximi
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	NEAR B
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	NEAR B
2	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	NEAR B
3	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	NEAR B
4	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	NEAR B

```
# I can now drop the column 'ocean_proximity' since we have converted the data and they are alreay there in our newdf.

newdf.drop(columns=['ocean_proximity'], inplace=True)
```

```
newdf.head()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	0

```
# the other way of droping the coulmn is directly doing it on the same line of code where we converted the data. folow below
newdf= pd.concat([df, pd.get_dummies(df['ocean_proximity'])], axis=1).drop(columns=['ocean_proximity'])
```

```
# now my dataframe is ready after I processed it
newdf.head()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	0
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	0	0
2	52.0	1467.0	190.0	496.0	177.0	7.2574	352100.0	0	0
3	52.0	1274.0	235.0	558.0	219.0	5.6431	341300.0	0	0
4	52.0	1627.0	280.0	565.0	259.0	3.8462	342200.0	0	0

```
newdf= newdf.fillna(0)   # to handel the missing values/nullvalues
```

```
# step1- split data into X and Y (independent, denpendent values).
X = newdf.drop(columns=['median_house_value'])             # here I have droped dependent column which is Y and taking all other colm
Y = newdf['median_house_value']                            # here I have taken only denpendent column ie, my Y


```

```
X.head()
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	<1H OCEAN	INLAND	ISLAND	NEAR BAY	NEA OCEA
0	41.0	880.0	129.0	322.0	126.0	8.3252	0	0	0	1	
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	0	0	0	1	
2	52.0	1467.0	190.0	496.0	177.0	7.2574	0	0	0	1	
3	52.0	1274.0	235.0	558.0	219.0	5.6431	0	0	0	1	
4	52.0	1627.0	280.0	565.0	259.0	3.8462	0	0	0	1	

```
Y
```

```
0        452600.0
1        358500.0
2        352100.0
3        341300.0
4        342200.0
           ...
20635     78100.0
20636     77100.0
20637     92300.0
20638     84700.0
20639     89400.0
Name: median_house_value, Length: 20640, dtype: float64
```

```
# now when the X and Y are separated. we are now splitting our data into traning and testing
from sklearn.model_selection import train_test_split
```

```
# lets check if there are null values in our df
newdf.isnull().sum()
```

```
housing_median_age    0
total_rooms           0
total_bedrooms        0
population            0
households            0
median_income         0
median_house_value    0
<1H OCEAN             0
INLAND                0
```

```
ISLAND                0
NEAR BAY              0
NEAR OCEAN            0
dtype: int64
```

```
# in the above we have 207 null values in 'total_bedrooms' column
# so now i go back where we have segregated our X,Y values and fill the null values to '0' and execute the lines of code
# so that now we will have no null values in our df. 'newdf= newdf.fillna(0)' this is to fill null to '0'
```

```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test= train_test_split(X, Y, train_size= 0.8)
```

```
X_train.shape    # here 80% is going to training
```

```
(16512, 11)
```

```
X_test.shape     # 20% is going to testing
```

```
(4128, 11)
```

```
# now we try to implement the model. lets say we will try Linear Regression
from sklearn.linear_model import LinearRegression

```

```
# once i have LR algorithem, i will initialize it

Lr= LinearRegression()            # here we have the equation y=mx+c, during initialization will have empty m and c value

```

```
Lr.fit(X_train,Y_train)             # here i have trained my model with training data
```

```
LinearRegression()
```

```
# now i can predict my test data
y_pred= Lr.predict(X_test)


```

```
# to know whether my predictions are good/bad i will use some performance metrics
# Performance Metrics
from sklearn.metrics import mean_squared_error, r2_score

```

```
# MEan Squared Error will tell me average error. and r2 score will tell me the accuracy percentage
```

```
mean_squared_error(y_pred, Y_test)     # this is meansqerror/average sq error. eauation for this is within mean_squared_error
```

```
4986564372.803906
```

```
r2_score(y_pred, Y_test)              # this compares the actual and predicted value. also tells if its good/bad model
```

```
0.407291879968123
```

```
# we tried single linear regression model and not really happy with accuracy so we will now try other model, say random forest Regress

```

```
# Random Forest Regressor
from sklearn.ensemble import RandomForestRegressor
```

```
rfr= RandomForestRegressor()      # will initialize the model
```

```
rfr.fit(X_train, Y_train)
```

```
RandomForestRegressor()
```

```
Y_predi = rfr.predict(X_test)
```

```
r2_score(Y_predi, Y_test)               # we can now see that a signoficant improvement happened coz of model change
```

```
0.6044690192213942
```

```
mean_squared_error(Y_predi, Y_test)             # now error has further reduced from linear MSE.
```

```
3873243110.926816
```

```
# now lets see what more can we do to improve the model. coz we already chnaged the model from simple linear to randomForest
# we got better results. howvever we need to improve the model further to achieve something better than what we already got
```

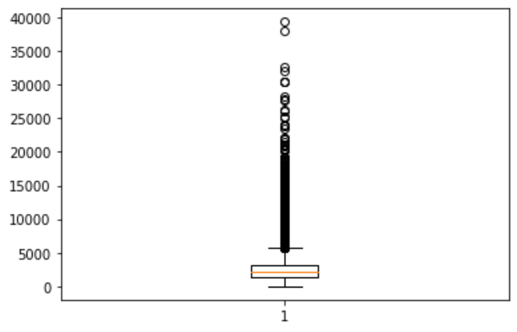
```
newdf.head(2)
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	0
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	0	0

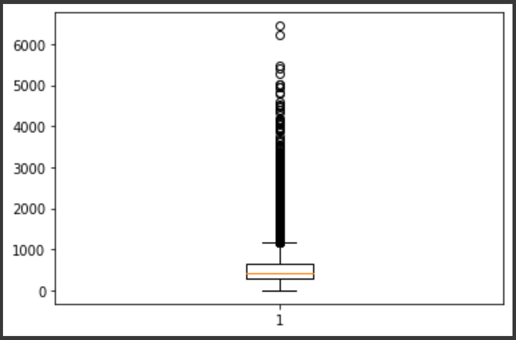
```
# lets visualize the data. and to do that lets plot
import matplotlib.pyplot as plt

```

```
#I want the bar chart/freequency distribution plot to obeserve if there are any insights/skewness in data/say any outliers
# Boxplot - is a visualization tool which helps to figure out if there are any outliers inthe data or not
# so lets now see what 'total rooms' data will tel us in Boxplot.
```

```
# lets plot
plt.boxplot(newdf['total_rooms'])
plt.show()
```



```
 # in  the above we can see there are lots of outliers in 'total rooms' the black staright lines are outliers
 # similarly lets check boxplot for 'total bedrooms'
 plt.boxplot(newdf['total_bedrooms'])
 plt.show()
```



```
# even here we have many outliers. lets check population now
plt.boxplot(newdf['population'])
plt.show()                                     # this has compartively less outliers
```

```
# we canalready see that data has lots of outliers when we did the visualisation analysis
# data analysis
# another analysis we can do is by creating ratio's and proportions
# in this case we can get the ratio's on 'population : household'
# so here 'population' and 'household' data indivudualy may have lots of outliers but the ratio between them maynot have much outliers
# -- in a country where pop is a lot, households will also be lot
# -- inividualy pop and household might show an outliers
# -- but if the ratio households : population is not an outlier, then the row is not an outlier
# -- whe we take the ration it normalises the population and household and hence it should not be an outlier
```

```
# now lets take a ratio of household and population
newdf['households'] / newdf['population']
```

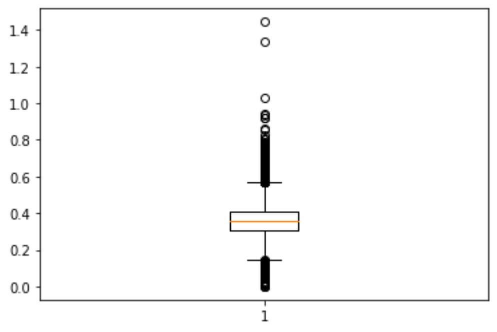
```
0        0.391304
1        0.473969
2        0.356855
3        0.392473
4        0.458407
           ...
20635    0.390533
20636    0.320225
20637    0.429990
20638    0.470985
20639    0.382120
Length: 20640, dtype: float64
```

```
 # lets now create another new column with ratio of these 2, let it be 'households_to_population'
newdf['households_to_population']  = newdf['households'] / newdf['population']

```

```
newdf.head(2)
```

	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value	<1H OCEAN	INLAND
0	41.0	880.0	129.0	322.0	126.0	8.3252	452600.0	0	0
1	21.0	7099.0	1106.0	2401.0	1138.0	8.3014	358500.0	0	0

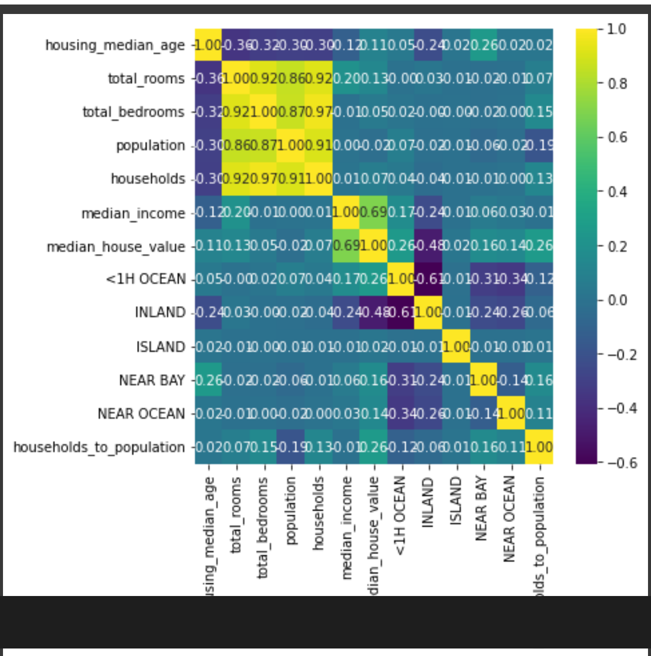
```
# we can see that our df has a new column at the end with ratio we want. now we can check if there are any outliers in the new ration

plt.boxplot(newdf['households_to_population'])
plt.show()
```



```
# here we can see outliers are on both sides and ther are not much now. its reduced to a lot extent
# we are now doing a data analysis where we have to see each and every aspect to see where we can alter or clean the data
# we can have another approach that is data correlation
# correlation is the similarity of data in multiple colmns. lets check this
```

```
# for correlation we have to import seaborn
import seaborn as sns
```

```
# seaborn has a ability to checkout the heatmap plot.here we have to paas correlation o/p of df and the color map
# i can add other functionalities also like annotation=true, format=.2f. you can increase the size of the plot too
plt.figure(figsize=(6,6))              # this is to increase the size of teh plot
sns.heatmap(newdf.corr(), cmap= 'viridis', annot= True, fmt= '.2f')
plt.show()     # this will remove unwanted text and details of image
```


housing_median_age
total_rooms
total_bedrooms
population
households
median_income
median_house_value
<1H OCEAN
INLAND
ISLAND
NEAR BAY
NEAR OCEAN
households_to_population
sing_median_age
total_rooms
total_bedrooms
population
households
median_income
dian_house_value
<1H OCEAN
INLAND
ISLAND
NEAR BAY
NEAR OCEAN
olds_to_population
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6