In [1]:
```python
import pandas as pd
import numpy as np
import warnings
warnings.filterwarnings('ignore')
```

In [3]:
```python
# Importing the dataset
housing = pd.read_excel('housing.xlsx')
housing
```

Out[3]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	m
0	-122.23	37.88	41	880	129.0	322	126	8.3252	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	
...	...	...	...	...	...	...	...	...	
20635	-121.09	39.48	25	1665	374.0	845	330	1.5603	
20636	-121.21	39.49	18	697	150.0	356	114	2.5568	
20637	-121.22	39.43	17	2254	485.0	1007	433	1.7000	
20638	-121.32	39.43	18	1860	409.0	741	349	1.8672	
20639	-121.24	39.37	16	2785	616.0	1387	530	2.3886	

20640 rows × 10 columns

In [4]:
```python
housing.info()
# housing.size
# housing.shape

# Exploratory Data Analysis
# 20640 corresponds to the number of districts, 10 responds to the characteristics prevailing in ea
# One categorical variable - ocean_proximity
# median_house_value is the median sales of each district
```

```
<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  int64  
 3   total_rooms         20640 non-null  int64  
 4   total_bedrooms      20433 non-null  float64
 5   population          20640 non-null  int64  
 6   households          20640 non-null  int64  
 7   median_income       20640 non-null  float64
 8   median_house_value  20640 non-null  int64  
 9   ocean_proximity     20640 non-null  object 
dtypes: float64(4), int64(5), object(1)
memory usage: 1.6+ MB
```

In [5]:
```
# Looking at the basic statistics
housing.describe()
housing.describe().T
```

Out[5]:

	count	mean	std	min	25%	50%	75%	
longitude	20640.0	-119.569704	2.003532	-124.3500	-121.8000	-118.4900	-118.01000	-114.
latitude	20640.0	35.631861	2.135952	32.5400	33.9300	34.2600	37.71000	41.
housing_median_age	20640.0	28.639486	12.585558	1.0000	18.0000	29.0000	37.00000	52.
total_rooms	20640.0	2635.763081	2181.615252	2.0000	1447.7500	2127.0000	3148.00000	39320.
total_bedrooms	20433.0	537.870553	421.385070	1.0000	296.0000	435.0000	647.00000	6445.
population	20640.0	1425.476744	1132.462122	3.0000	787.0000	1166.0000	1725.00000	35682.
households	20640.0	499.539680	382.329753	1.0000	280.0000	409.0000	605.00000	6082.
median_income	20640.0	3.870671	1.899822	0.4999	2.5634	3.5348	4.74325	15.
median_house_value	20640.0	206855.816909	115395.615874	14999.0000	119600.0000	179700.0000	264725.00000	500001.

In [6]:
```
# Checking for null values
# total bedrooms has 207 NaN values
housing.isnull()
housing.isnull().sum()
```

Out[6]:
```
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
```

In [7]:
```
# Dropping null values
housing.dropna(inplace = True)
```

In [8]:
```
housing.isnull().sum()
```

Out[8]:
```
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
```

In [9]:
```
housing.columns
```

Out[9]:
```
Index(['longitude', 'latitude', 'housing_median_age', 'total_rooms',
       'total_bedrooms', 'population', 'households', 'median_income',
       'median_house_value', 'ocean_proximity'],
      dtype='object')
```

In [10]:
```python
# Encode categorical data - Convert categorical to numerical data
# The feature column ocean_proximity is in categorical format
# The are multiple ways to convert categorical data - label encoding, binary encone, dummies, etc
# Pandas supports many more methods for the same

# Label encoding involves converting each value from the column into a number
# Ocean_proximity -'Near Bay' | '<1H Ocean' | 'Inland' | 'Near Ocean' | 'Island'
# Assigning a specific number
# <1H Ocean = 0, Inland = 1, Island = 2, Near Bay = 3, Near Ocean = 4

from sklearn.preprocessing import LabelEncoder
LabelEncoder = LabelEncoder()
```

In [11]:
```python
# Converting the column from object to category
housing['ocean_proximity'] = housing['ocean_proximity'].astype('category')
housing.dtypes
```

Out[11]:
```
longitude              float64
latitude               float64
housing_median_age       int64
total_rooms              int64
total_bedrooms         float64
population               int64
households               int64
median_income          float64
median_house_value       int64
ocean_proximity       category
dtype: object
```

In [12]:
```python
# Assigning the encoded variable to a new column using the cat.codes accesor
housing['ocean_proximity_value'] = housing['ocean_proximity'].cat.codes
housing.head()
```

Out[12]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	mediar
0	-122.23	37.88	41	880	129.0	322	126	8.3252	
1	-122.22	37.86	21	7099	1106.0	2401	1138	8.3014	
2	-122.24	37.85	52	1467	190.0	496	177	7.2574	
3	-122.25	37.85	52	1274	235.0	558	219	5.6431	
4	-122.25	37.85	52	1627	280.0	565	259	3.8462	

In [13]:
```
# Since we are using regression analysis, it is better to standardize the data
# Standardization involves shifting the distribution of each data point to a mean of 0 and an SD o
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
x =  pd.DataFrame(sc_x.fit_transform(housing.drop(["ocean_proximity", "median_house_value"], axis
       columns=['longitude', 'latitude', 'housing_median_age', 'total_rooms',
      'total_bedrooms', 'population', 'households', 'median_income', 'ocean_proximity_value'])
x.head()
# x.shape
```

Out[13]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	ocear
0	-1.327314	1.051717	0.982163	-0.803813	-0.970325	-0.973320	-0.976833	2.345163	
1	-1.322323	1.042355	-0.606210	2.042130	1.348276	0.861339	1.670373	2.332632	
2	-1.332305	1.037674	1.855769	-0.535189	-0.825561	-0.819769	-0.843427	1.782939	
3	-1.337296	1.037674	1.855769	-0.623510	-0.718768	-0.765056	-0.733562	0.932970	
4	-1.337296	1.037674	1.855769	-0.461970	-0.611974	-0.758879	-0.628930	-0.013143	

In [14]:
```
# Standardizing the target variable - median_house_value
y = pd.DataFrame(sc_x.fit_transform(housing.drop(['ocean_proximity','longitude', 'latitude', 'hous
      'total_bedrooms', 'population', 'households', 'median_income', 'ocean_proximity_value' ], a
                 columns=['median_house_value'])
y.head()
# y.shape
```

Out[14]:

	median_house_value
0	2.128819
1	1.313626
2	1.258183
3	1.164622
4	1.172418

In [15]:
```
# splitting the data
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = .20, random_state = 1,)
```

In [16]:
```
# Linear Regression
from sklearn.linear_model import LinearRegression
model1 = LinearRegression()
model1.fit(x_train, y_train)
```

Out[16]:

```
▾ LinearRegression
LinearRegression()
```

In [17]:
```
# Printing the coefficients and intercept
print(model1.intercept_)
print(model1.coef_)
```

```
[0.00280687]
[[-0.74612343 -0.79186502  0.12810082 -0.15758988  0.41665016 -0.36991155
   0.15622207  0.66336783  0.00130051]]
```

In [18]:
```
# Predicting on the test data
y_p = model1.predict(x_test)
print(y_p)
print(y_test)
```

```
[[ 1.47657526]
 [ 0.5809054 ]
 [-0.51896355]
 ...
 [-0.62287874]
 [ 1.92401337]
 [ 0.58810714]]
       median_house_value
6643             1.334417
4084             0.733202
15284           -0.654618
20068            2.539456
11726           -0.762906
...                   ...
6607             0.307846
9765             0.096468
9364            -1.187395
16981            2.539456
3933             0.115527

[4087 rows x 1 columns]
```

In [19]:
```
# Calculating the Root Mean Squared Error - RMSE
# Test and train scores
from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y_test, y_p)))
print(np.sqrt(metrics.mean_squared_error(y_train, model1.predict(x_train))))
```

```
0.5949339345602463
0.6045058776630716
```

In [20]:
```
# Univariate regression
x1 = x[['median_income']]
y1 = y[['median_house_value']]
```

In [21]:
```
# Splitting the data
x1_train, x1_test, y1_train, y1_test = train_test_split(x1, y1, test_size = .20, random_state = 1,
```

In [22]:
```
# Linear Regression
from sklearn.linear_model import LinearRegression
model2 = LinearRegression()
model2.fit(x1_train, y1_train)
```

Out[22]:
```
▾ LinearRegression

LinearRegression()
```

In [23]:
```
# Printing the coefficients and intercept
print(model2.intercept_)
print(model2.coef_)
```

```
[0.00341938]
[[0.68861017]]
```

In [24]:
```python
# Predicting on the test data
y_p1 = model2.predict(x1_test)
print(y_p1)
print(y1_test)
```

```
[[ 1.39584913]
 [ 0.52009629]
 [-0.38636627]
 ...
 [-0.85803426]
 [ 1.58873665]
 [ 0.46574697]]
       median_house_value
6643             1.334417
4084             0.733202
15284           -0.654618
20068            2.539456
11726           -0.762906
...                   ...
6607             0.307846
9765             0.096468
9364            -1.187395
16981            2.539456
3933             0.115527

[4087 rows x 1 columns]
```

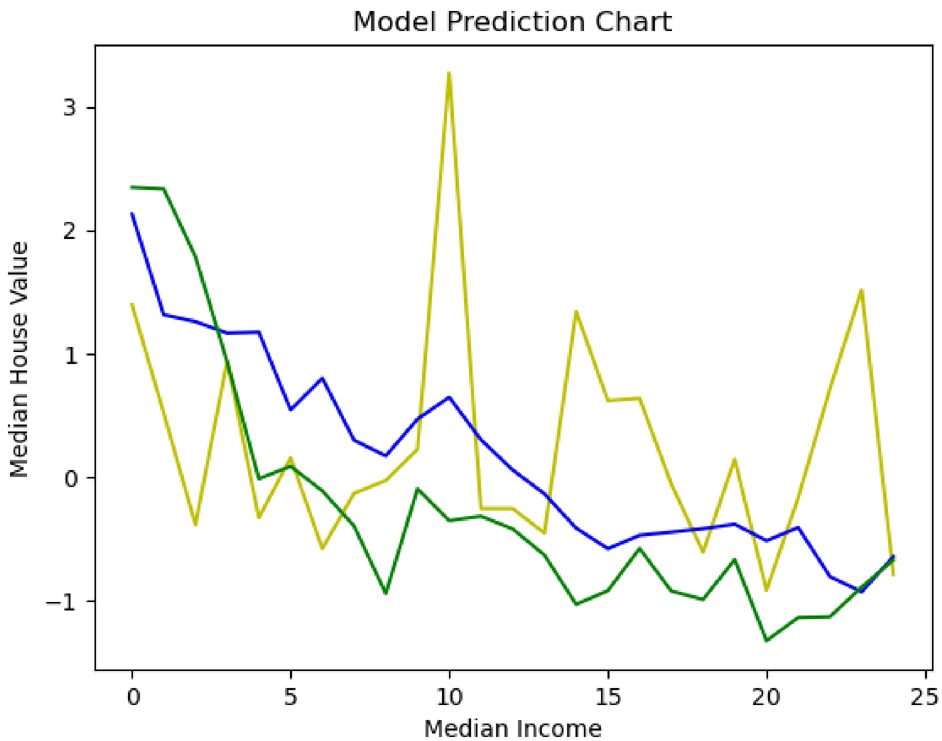
In [25]:
```python
# Calculating the Root Mean Squared Error - RMSE
# Test and train scores
from sklearn import metrics
print(np.sqrt(metrics.mean_squared_error(y1_test, y_p1)))
print(np.sqrt(metrics.mean_squared_error(y1_train, model2.predict(x1_train))))
```

```
0.7103347089635074
0.7290952831251405
```

In [26]:
```
# Lets plot the fitted model2 with the revised dataset
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(y_p1[0:25], 'y')
plt.plot(y1[0:25], 'b')
plt.plot(x1[0:25], 'g')
plt.xlabel('Median Income')
plt.ylabel('Median House Value')
plt.title('Model Prediction Chart')
```

Out[26]:
```
Text(0.5, 1.0, 'Model Prediction Chart')
```



In [27]:
```
# Thank You :)
```

In [ ]:
```

```