CSC-462 ARTIFICIAL INTELLIGENCE

LAB PROJECT

Manaal Waseem

FA18-BCE-074

BCE-7B

Activities:

Task 1

Code:

In [164]:
```
#import Libraries
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline

#import 'r2_score'
from sklearn.metrics import r2_score

#import 'train_test_split'
from sklearn.model_selection import train_test_split

#import SimpleImputer
from sklearn.impute import SimpleImputer

#import MinMaxScaler
from sklearn.preprocessing import MinMaxScaler

#import pipeline
from sklearn.pipeline import Pipeline

from sklearn.model_selection import learning_curve
```

In [165]:
```
#Load data
housing= pd.read_csv("housing.csv")
```

In [166]:
```
#A Look at Housing data structure
housing.head()
```

Out[166]:

	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

In [167]:
```
housing.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
```

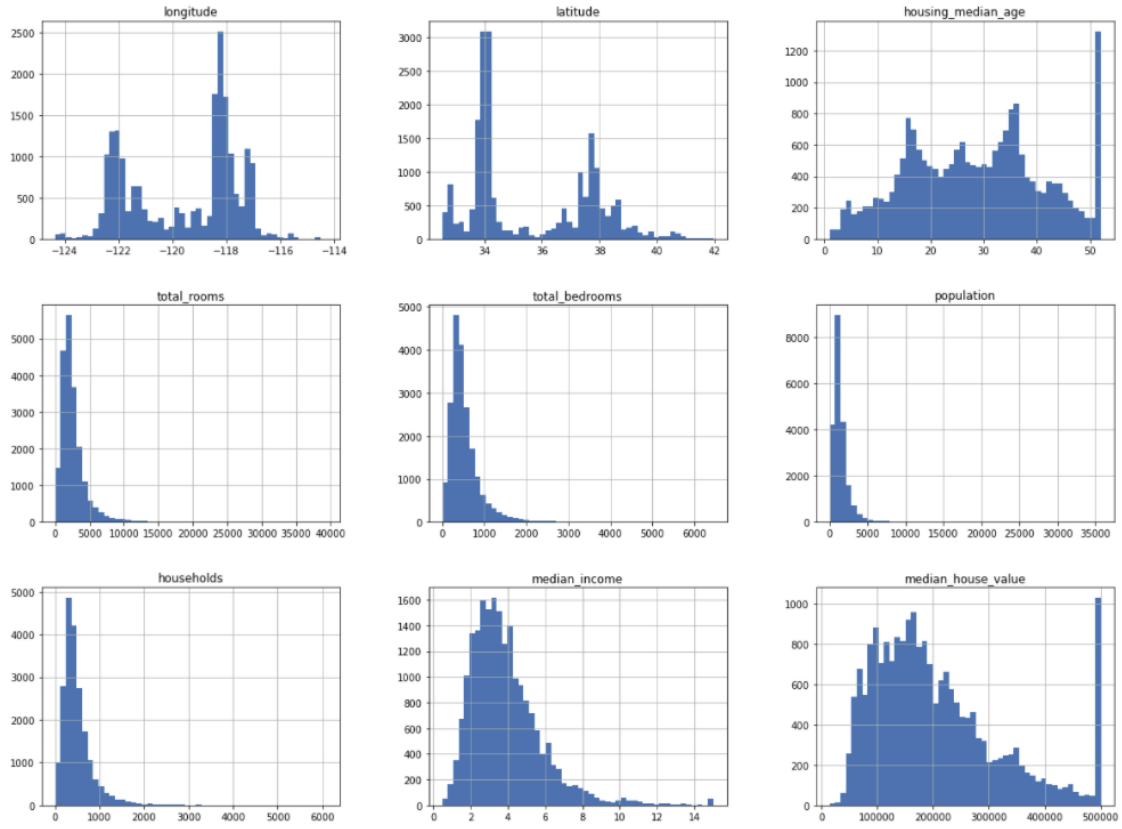
In [168]:
```
housing.describe()
```

Out[168]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income	median_house_value
count	20640.000000	20640.000000	20640.000000	20640.000000	20433.000000	20640.000000	20640.000000	20640.000000	20640.000000
mean	-119.569704	35.631861	28.639486	2635.763081	537.870553	1425.476744	499.539680	3.870671	206855.816909
std	2.003532	2.135952	12.585558	2181.615252	421.385070	1132.462122	382.329753	1.899822	115395.615874
min	-124.350000	32.540000	1.000000	2.000000	1.000000	3.000000	1.000000	0.499900	14999.000000
25%	-121.800000	33.930000	18.000000	1447.750000	296.000000	787.000000	280.000000	2.563400	119600.000000
50%	-118.490000	34.260000	29.000000	2127.000000	435.000000	1166.000000	409.000000	3.534800	179700.000000
75%	-118.010000	37.710000	37.000000	3148.000000	647.000000	1725.000000	605.000000	4.743250	264725.000000
max	-114.310000	41.950000	52.000000	39320.000000	6445.000000	35682.000000	6082.000000	15.000100	500001.000000

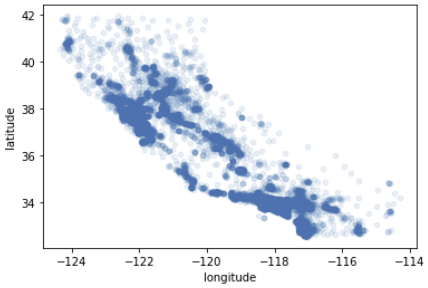
In [169]:
```
#drop ocean_proximity column as it is not required in our project
housing = housing.drop("ocean_proximity", axis=1)
```

In [170]:
```
#Data Exploration
housing.hist(bins=50, figsize=(20,15))
plt.show()
```



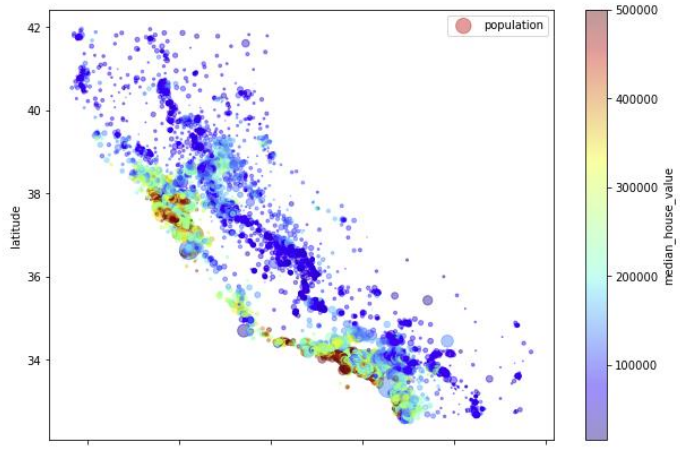
In [171]:
```
#visualize the places where there is a high density of data points
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.1)
```

Out[171]: <AxesSubplot:xlabel='longitude', ylabel='latitude'>



In [172]:
```
#a look at the housing prices
housing.plot(kind="scatter", x="longitude", y="latitude", alpha=0.4,
s=housing["population"]/100, label="population", figsize=(10,7),
c="median_house_value", cmap=plt.get_cmap("jet"), colorbar=True,
)
plt.legend()
```

Out[172]: <matplotlib.legend.Legend at 0x22d08c35250>



In [173]:
```
#Correlation matrix
corr_matrix = housing.corr()
corr_matrix["median_house_value"].sort_values(ascending=False)
```

Out[173]:
```
median_house_value    1.000000
median_income         0.688075
total_rooms           0.134153
housing_median_age    0.105623
households            0.065843
total_bedrooms        0.049686
population           -0.024650
longitude            -0.045967
latitude             -0.144160
Name: median_house_value, dtype: float64
```

In [174]:
```
housing = housing.dropna()
```

In [182]:
```
housing.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20433 entries, 0 to 20639
Data columns (total 9 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
 8   median_house_value  20433 non-null  float64
dtypes: float64(9)
memory usage: 1.6 MB
```

In [175]:
```
#separate features X and target Y
Y = housing['median_house_value']
X = housing.drop('median_house_value', axis = 1)
```

In [176]:
```
def performance_metric(y_true, y_predict):
    score = r2_score(y_true, y_predict)
    return score
```

In [177]:
```
#split the data into training and testing subsets
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print("Training and testing split was successful")
```

```
Training and testing split was successful
```

In [183]:
```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 20433 entries, 0 to 20639
Data columns (total 8 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   longitude           20433 non-null  float64
 1   latitude            20433 non-null  float64
 2   housing_median_age  20433 non-null  float64
 3   total_rooms         20433 non-null  float64
 4   total_bedrooms      20433 non-null  float64
 5   population          20433 non-null  float64
 6   households          20433 non-null  float64
 7   median_income       20433 non-null  float64
dtypes: float64(8)
memory usage: 1.4 MB
```

In [207]:
```
mean = X.mean(axis=0)
std = X.std(axis=0)
```

In [208]:
```
X = X - mean
X = X / std
```

In [178]:
```
from sklearn.base import BaseEstimator, TransformerMixin
rooms_ix, bedrooms_ix, population_ix, households_ix = 3, 4, 5, 6

class CombinedAttributesAdder(BaseEstimator, TransformerMixin):
    def __init__(self, add_bedrooms_per_room = True):
        self.add_bedrooms_per_room = add_bedrooms_per_room
    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        rooms_per_household = X[:, rooms_ix] / X[:, households_ix]
        population_per_household = X[:, population_ix] / X[:, households_ix]

        if self.add_bedrooms_per_room:
            bedrooms_per_room = X[:, bedrooms_ix] / X[:, rooms_ix]
            return np.c_[X, rooms_per_household, population_per_household, bedrooms_per_room]
        else:
            return np.c_[X, rooms_per_household, population_per_household]

attr_adder = CombinedAttributesAdder(add_bedrooms_per_room=False)
```

In [ ]:
```
```

In [191]:
```
#Pipeline Fill missing valuse, Combine attribute and Feature Normalization
# data_pipeline = Pipeline([
# ('imputer', SimpleImputer(strategy="median")),
# ('attribs_adder', CombinedAttributesAdder()),
# ('min_max_scaler', MinMaxScaler()),
# ])

#X_pipeline = data_pipeline.fit_transform(X_train)
#scalar = MinMaxScaler()
#X_pipeline = scalar.fit_transform(X_train)
X_pipeline = X
```

In [215]:
```
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
#lin_reg.fit(X_pipeline, Y_train)
lin_reg.fit(X, Y)
```

Out[215]:
```
LinearRegression()
```

In [210]:
```
test_df = pd.DataFrame({
    'longitude': [-119.85],
    'latitude': [37.48],
    'housing_median_age': [22],
    'total_rooms': [2850],
    'total_bedrooms': [500],
    'population': [1150],
    'households': [460],
    'median_income': [3.12]
})
```

In [211]:
```
test_df
```

Out[211]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-119.85	37.48	22	2850	500	1150	460	3.12

In [212]:
```
test_df = test_df - mean
test_df = test_df / std
```

In [213]:
```
test_df
```

Out[213]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-0.139406	0.864456	-0.526779	0.097698	-0.089872	-0.242627	-0.103148	-0.395496

In [194]:
```
# some_data = X_train.iloc[:5]
# some_labels = Y_train.iloc[:5]
# some_data_prepared = scalar.fit_transform(some_data)
# print("Predictions:", lin_reg.predict(some_data_prepared))
# print("Y:", list(some_labels))

some_data = X.iloc[:5]
some_labels = Y.iloc[:5]
some_data_prepared = scalar.fit_transform(some_data)
print("Predictions:", lin_reg.predict(some_data_prepared))
print("Y:", list(some_labels))
```

```
Predictions: [ -81100.51493603  165367.37007346  109819.06561593 -453833.36407158
   84713.18283089]
Y: [227600.0, 110400.0, 248100.0, 305600.0, 214600.0]
```

In [195]:
```
from sklearn.metrics import mean_squared_error
housing_predictions = lin_reg.predict(X_pipeline)
lin_mse = mean_squared_error(Y_train, housing_predictions)
lin_rmse = np.sqrt(lin_mse)
lin_rmse
```

Out[195]:
```
69411.6554100667
```

In [196]:
```
#R2 metric
r2=performance_metric(Y_train, housing_predictions)
r2
```

Out[196]:
```
0.6360185727313741
```

In [197]:
```
train_sizes, train_scores, test_scores = learning_curve(estimator=LinearRegression(), X=X_train, y=Y_train,
                                                        cv=10, train_sizes=np.linspace(0.1, 1.0, 10),
                                                       n_jobs=1)
```

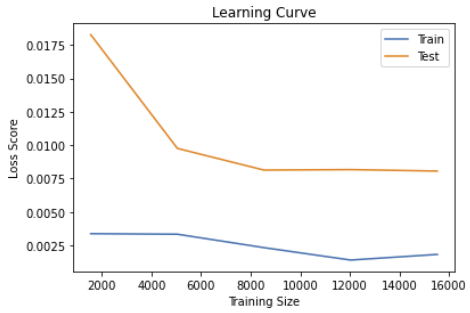
In [198]:
```
# Learning Curve

size, train_scores, test_scores = learning_curve(lr, X_train, Y_train)
```

In [ ]:
```
train_std = np.std(train_scores, axis=1)
test_std = np.std(test_scores, axis=1)
```

In [225]:
```
plt.plot(size, train_std)
plt.plot(size, test_std)
plt.legend(['Train', 'Test'])
plt.title('Learning Curve')
plt.xlabel('Training Size')
plt.ylabel('Loss Score')
```

Out[225]:
```
Text(0, 0.5, 'Loss Score')
```



In [200]:
```
# values of the trained weights
print(lin_reg.coef_)
```

```
[ -420361.45533591  -398382.16450844    58909.95253161  -331860.3736203
   746410.9383576  -1311516.057383     253215.48155619   584260.59557271]
```

In [201]:
```
#Test cases
X_test_pipeline = scalar.fit_transform(X_test)
final_predictions = lin_reg.predict(X_test_pipeline)
final_mse = mean_squared_error(Y_test, final_predictions)
final_rmse = np.sqrt(final_mse)
```

In [202]:
```
# final_rmse
final_rmse
```

Out[202]:
```
92901.10019388044
```

In [203]:
```
#R2 metric
r2=performance_metric(Y_test, final_predictions)
r2
```

Out[203]:
```
0.368884802895016
```

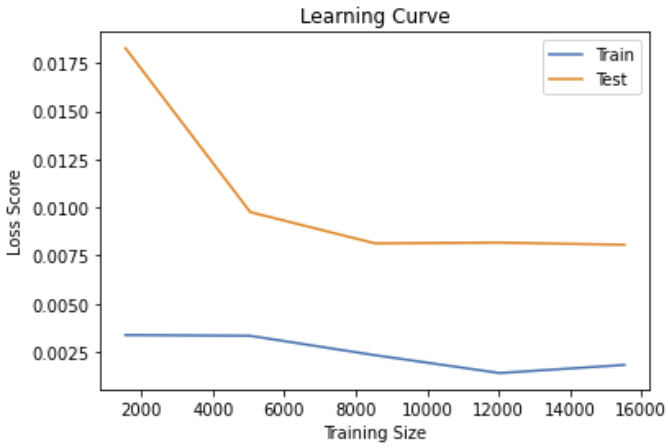
In [204]:
```
#final training loss value
print('Your final training loss value rmse = {} R2 = {}', final_rmse, r2)
```

```
Your final training loss value rmse = {} R2 = {} 92901.10019388044 0.368884802895016
```

In [224]:
```
print(f'Predicted value for median house value: {lr.predict([[-119.85,37.48,22,2850,500,1150,400,3.12]])}')
print(f'Value of training weights: {lr.coef_}')
actual = np.array(list(Y_train))
predicted = np.array(list(pri))
print(mean_squared_error(actual,predicted,squared=False))
```

```
Predicted value for median house value: [102411.04824779]
Value of training weights: [-4.30776320e+04 -4.28059533e+04  1.13640741e+03 -8.26537227e+00
  1.19764264e+02 -3.81207981e+01  4.00312819e+01  4.01947682e+04]
69633.72002531342
```

Learning Curve



Weights

```
In [154]: # values of the trained weights
          print(lin_reg.coef_)
```

```
[ -418268.88736635  -387070.03453979    60557.81471414    77206.23571651
    91860.60238544 -1469411.46051484   663328.64835277   618217.58800007
   495057.21138978    71127.52066782   292806.6403382 ]
```

Training Loss Value:

```
In [161]: #final training loss value
          print('Your final training loss value rmse = {} R2 = {}', final_rmse, r2)
```

```
Your final training loss value rmse = {} R2 = {} 74292.19350137576 0.5788085928215299
```

Predicted Value for Test Case:

```
Predicted value for median house value: [102411.04824779]
Value of training weights: [-4.30776320e+04 -4.28059533e+04  1.13640741e+03 -8.26537227e+00
  1.19764264e+02 -3.81207981e+01  4.00312819e+01  4.01947682e+04]
69633.72002531342
```

Task 2

Code:

In [3]:
```
import tensorflow.keras as keras
import pandas as pd
import numpy as numpy
import matplotlib.pyplot as plt
import seaborn as sns
```

In [28]:
```
df = pd.read_csv('housing.csv')
```

In [3]:
```
df.shape
```

Out[3]:
```
(20640, 10)
```

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
```

In [5]:
```
df = df.dropna()
```

first 8 columns are features, 9th col is label and 10th is not used

In [6]:
```
labels = df.median_house_value
```

In [7]:
```
features = df.copy()
```

In [8]:
```
features.drop(columns=['median_house_value', 'ocean_proximity'], inplace=True)
```

In [9]:
```
features.head()
```

Out[9]:

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

In [10]:
```
features.shape
```

Out[10]:
```
(20433, 8)
```

In [11]:
```
labels.shape
```

Out[11]:
```
(20433,)
```

Normalization

In [12]:
```
mean = features.mean(axis=0)
std = features.std(axis=0)
```

In [13]:
```
features = features - mean
features = features / std
```

In [14]:
```
mean
```

Out[14]:
```
longitude           -119.570689
latitude              35.633221
housing_median_age    28.633094
total_rooms         2636.504233
total_bedrooms       537.870553
population          1424.946949
households           499.433465
median_income          3.871162
dtype: float64
```

In [15]:
```
features.head()
```

Out[15]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-1.327281	1.051692	0.982139	-0.803793	-0.970301	-0.973296	-0.976809	2.345106
1	-1.322290	1.042330	-0.606195	2.042080	1.348243	0.861318	1.670332	2.332575
2	-1.332272	1.037649	1.855723	-0.535176	-0.825541	-0.819749	-0.843406	1.782896
3	-1.337263	1.037649	1.855723	-0.623495	-0.718750	-0.765037	-0.733544	0.932947
4	-1.337263	1.037649	1.855723	-0.461959	-0.611959	-0.758860	-0.628914	-0.013143

In [16]:
```
labels.head()
```

Out[16]:
```
0    452600.0
1    358500.0
2    352100.0
3    341300.0
4    342200.0
Name: median_house_value, dtype: float64
```

Neural Network Model

In [17]:
```
model = keras.models.Sequential()
model.add(keras.layers.Dense(8, activation='relu', input_shape=(8, )))
model.add(keras.layers.Dense(1))

model.compile(
    optimizer='rmsprop',
    loss='mse',
    metrics=['mae']
)
```

In [21]:
```
model.build()
```

In [22]:
```
model.fit(features, labels, epochs=50, batch_size=32)
```

```
Epoch 1/50
639/639 [==============================] - 2s 3ms/step - loss: 24784873472.0000 - mae: 123846.9688
Epoch 2/50
639/639 [==============================] - 2s 3ms/step - loss: 24534173696.0000 - mae: 123053.7344
Epoch 3/50
639/639 [==============================] - 2s 3ms/step - loss: 24286550016.0000 - mae: 122263.8125
Epoch 4/50
639/639 [==============================] - 2s 3ms/step - loss: 24042905600.0000 - mae: 121491.8906
Epoch 5/50
639/639 [==============================] - 2s 3ms/step - loss: 23798994944.0000 - mae: 120715.5703
Epoch 6/50
639/639 [==============================] - 2s 3ms/step - loss: 23559567360.0000 - mae: 119947.5703
Epoch 7/50
639/639 [==============================] - 2s 3ms/step - loss: 23321169920.0000 - mae: 119187.1641
Epoch 8/50
639/639 [==============================] - 2s 3ms/step - loss: 23085172736.0000 - mae: 118433.7969
Epoch 9/50
639/639 [==============================] - 2s 3ms/step - loss: 22850301952.0000 - mae: 117681.8359
Epoch 10/50
639/639 [==============================] - 2s 3ms/step - loss: 22619633664.0000 - mae: 116939.9922
Epoch 11/50
639/639 [==============================] - 2s 3ms/step - loss: 22389725184.0000 - mae: 116208.4141
Epoch 12/50
639/639 [==============================] - 2s 3ms/step - loss: 22163886080.0000 - mae: 115477.0859
Epoch 13/50
639/639 [==============================] - 2s 3ms/step - loss: 21939034112.0000 - mae: 114750.6562
Epoch 14/50
639/639 [==============================] - 2s 3ms/step - loss: 21716807680.0000 - mae: 114036.6406
Epoch 15/50
639/639 [==============================] - 2s 3ms/step - loss: 21498261504.0000 - mae: 113327.4922
Epoch 16/50
639/639 [==============================] - 2s 3ms/step - loss: 21283094528.0000 - mae: 112626.4297
Epoch 17/50
639/639 [==============================] - 2s 3ms/step - loss: 21069518848.0000 - mae: 111933.3750
Epoch 18/50
639/639 [==============================] - 2s 3ms/step - loss: 20857874432.0000 - mae: 111247.3516
Epoch 19/50
639/639 [==============================] - 2s 3ms/step - loss: 20650741760.0000 - mae: 110577.6016
Epoch 20/50
639/639 [==============================] - 2s 3ms/step - loss: 20444252160.0000 - mae: 109907.5312
Epoch 21/50
639/639 [==============================] - 2s 3ms/step - loss: 20243519488.0000 - mae: 109252.6875
```

Out[22]:
```
<keras.callbacks.History at 0x7f5f7c1fd7d0>
```

In [35]:
```
test_df = pd.DataFrame({
    'longitude': [-119.85],
    'latitude': [37.48],
    'housing_median_age': [22],
    'total_rooms': [2850],
    'total_bedrooms': [500],
    'population': [1150],
    'households': [460],
    'median_income': [3.12]
})
```

In [36]:
```
test_df
```

Out[36]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-119.85	37.48	22	2850	500	1150	460	3.12

In [37]:
```
test_df = test_df - mean
test_df = test_df / std
```

In [38]:
```
test_df
```

Out[38]:

	longitude	latitude	housing_median_age	total_rooms	total_bedrooms	population	households	median_income
0	-0.139406	0.864456	-0.526779	0.097698	-0.089872	-0.242627	-0.103148	-0.395496

In [39]:
```
model.predict(test_df)
```

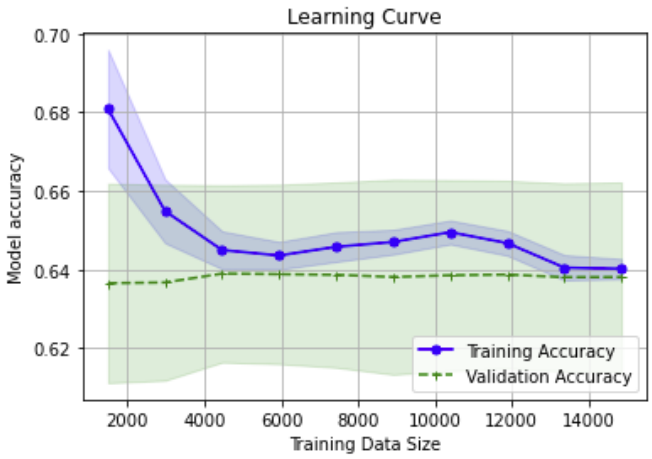
Out[39]:
```
array([[38523.855]], dtype=float32)
```

The predicted median house value is 38523.855

```
In [40]:
```

```
Out[40]: 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: 20433, dtype: float64
```

Learning Curve



Weights

```
In [154]: # values of the trained weights
          print(lin_reg.coef_)
```

```
[ -418268.88736635  -387070.03453979    60557.81471414    77206.23571651
    91860.60238544 -1469411.46051484   663328.64835277   618217.58800007
   495057.21138978    71127.52066782   292806.6403382 ]
```

Training Loss Value:

```
In [161]: #final training loss value
          print('Your final training loss value rmse = {} R2 = {}', final_rmse, r2)
```

```
Your final training loss value rmse = {} R2 = {} 74292.19350137576 0.5788085928215299
```

Predicted Value for Test Case:

The predicted median house value is 38523.855

THE END