Import Packages

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Data preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split

# Neural net moduel
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.callbacks import EarlyStopping
```

Read in the data

```
df = pd.read_csv('housing.csv')
df.head()
```

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

Data Preprocessing and cleaning

```
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
```

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

```
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
```

```
# drop rows with missing values
df.dropna(axis=0, inplace=True)
```

```
df.shape
```

```
(20433, 10)
```

```
df['ocean_proximity'].value_counts()
```

```
ocean_proximity
<1H OCEAN     9034
INLAND        6496
NEAR OCEAN    2628
NEAR BAY      2270
ISLAND           5
Name: count, dtype: int64
```

```
df['ocean_proximity'] = df['ocean_proximity'].map({"<1H OCEAN":0,"INLAND":1,"NEAR OCEAN":2,"NEAR BAY":3,"ISLAND":4})
```

```
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

Train test split

```
# Target variable is 'median_house_value'
y = df['median_house_value']
x = df.drop('median_house_value', axis=1)
print(x.shape)
print(y.shape)
```

```
(20433, 9)
(20433,)
```

```
# convert to numpy array
x = np.array(x)
y = np.array(y)
```

```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=123)

print(x_train.shape)
print(x_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(16346, 9)
(4087, 9)
(16346,)
(4087,)
```

Scaling

```
# Use MinMax scaler
min_max_scaler = MinMaxScaler()
x_train = min_max_scaler.fit_transform(x_train)
x_test = min_max_scaler.transform(x_test)
```

```
print(x_train)
print(x_test)
```

```
[[0.54425229 0.30180659 0.66666667 ... 0.06101974 0.21336947 0.25      ]
 [0.30315361 0.58448459 0.56862745 ... 0.00707237 0.27586516 0.25      ]
 [0.6948118  0.16790648 0.09803922 ... 0.25542763 0.25377581 0.25      ]
 ...
 [0.23601221 0.51328374 0.68627451 ... 0.05411184 0.2810375  0.        ]
 [0.19226857 0.55366631 1.         ... 0.06069079 0.26132053 0.75      ]
 [0.51068159 0.19978746 0.43137255 ... 0.12648026 0.29727866 0.5       ]]
[[0.71922686 0.02231668 0.43137255 ... 0.09769737 0.18199749 0.5       ]
 [0.30620549 0.64718385 0.45098039 ... 0.07976974 0.19436284 0.25      ]
```

```
       [0.3387589  0.54091392 0.74509804 ... 0.09917763 0.08313678 0.25      ]
       ...
       [0.15055951 0.62274176 0.76470588 ... 0.06973684 0.23342437 0.        ]
       [0.6103764  0.17534538 1.         ... 0.04276316 0.29645798 0.        ]
       [0.72227874 0.08926674 0.25490196 ... 0.13289474 0.31384395 0.        ]]
```

```
x_train.shape[1]
```

```
9
```

Building model (ANN)

```
model = Sequential([
    #input layer
    Dense(1000, input_shape=(x_train.shape[1],), activation='relu'),
    Dropout(0.2),
    #two hidden layers
    Dense(500, activation='relu'),
    Dropout(0.2),
    Dense(250, activation='relu'),
    #output layer
    Dense(1,activation='linear') # here 1 shows countinuous value(regression)
])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 1000)              10000

 dropout (Dropout)           (None, 1000)              0

 dense_1 (Dense)             (None, 500)               500500

 dropout_1 (Dropout)         (None, 500)               0

 dense_2 (Dense)             (None, 250)               125250

 dense_3 (Dense)             (None, 1)                 251

=================================================================
Total params: 636001 (2.43 MB)
Trainable params: 636001 (2.43 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Compile the model and set easlystoping
model.compile(optimizer='rmsprop', loss='mse', metrics=['mae'])
es = EarlyStopping(monitor='val_loss',mode='min',patience=50, restore_best_weights=True)
```

```
# fit the model (training)
history = model.fit(x_train, y_train, validation_data= (x_test, y_test), callbacks=[es], epochs=10, batch_size=50, verbose=1)
```

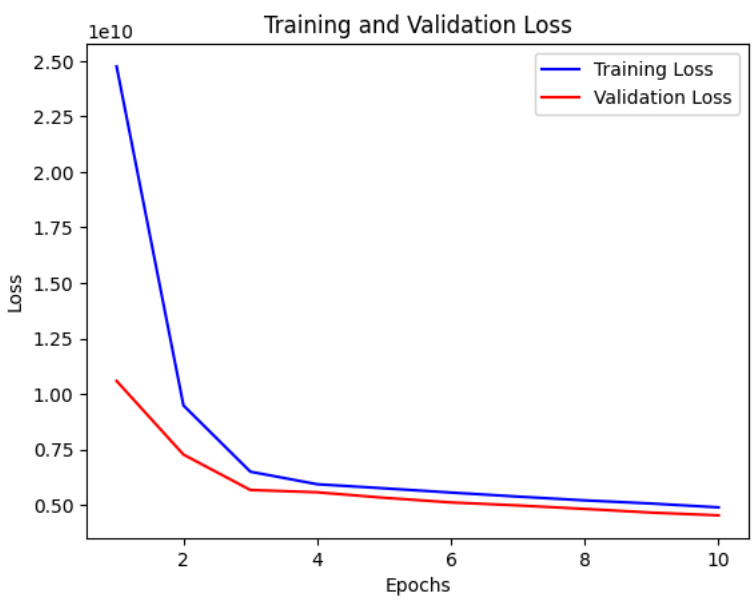
```
Epoch 1/10
327/327 [==============================] - 7s 19ms/step - loss: 24755955712.0000 - mae: 120739.8438 - val_loss: 10583593984.0000 - v
Epoch 2/10
327/327 [==============================] - 5s 15ms/step - loss: 9480070144.0000 - mae: 77307.1484 - val_loss: 7265815040.0000 - val_
Epoch 3/10
327/327 [==============================] - 6s 20ms/step - loss: 6495425024.0000 - mae: 60929.1836 - val_loss: 5670961664.0000 - val_
Epoch 4/10
327/327 [==============================] - 5s 16ms/step - loss: 5929728000.0000 - mae: 56496.6914 - val_loss: 5567024640.0000 - val_
Epoch 5/10
327/327 [==============================] - 5s 14ms/step - loss: 5745964544.0000 - mae: 55423.3672 - val_loss: 5320379904.0000 - val_
Epoch 6/10
327/327 [==============================] - 6s 19ms/step - loss: 5555000832.0000 - mae: 54292.5625 - val_loss: 5109794816.0000 - val_
Epoch 7/10
327/327 [==============================] - 5s 14ms/step - loss: 5373375488.0000 - mae: 53132.9141 - val_loss: 4974953984.0000 - val_
Epoch 8/10
327/327 [==============================] - 6s 17ms/step - loss: 5202966528.0000 - mae: 52000.4180 - val_loss: 4817773056.0000 - val_
Epoch 9/10
327/327 [==============================] - 10s 30ms/step - loss: 5059791872.0000 - mae: 51048.3125 - val_loss: 4650961920.0000 - val
Epoch 10/10
327/327 [==============================] - 8s 23ms/step - loss: 4887821312.0000 - mae: 50195.3477 - val_loss: 4527864832.0000 - val_
```

Metrics and Score

```
from sklearn.metrics import r2_score,mean_absolute_error,mean_squared_error,mean_squared_log_error

y_pred = model.predict(x_test)
print("mae :", mean_absolute_error(y_test,y_pred))
print("mse :", mean_squared_error(y_test,y_pred))
print("mae :", mean_squared_log_error(y_test,y_pred))
print("score :", r2_score(y_test,y_pred))
```

```
128/128 [==============================] - 1s 5ms/step
mae : 47538.12862334077
mse : 4527865104.3238945
mae : 0.10731578783218153
score : 0.6468630546819467
```

```
# Get the training and validation loss from the history object
training_loss = history.history['loss']
validation_loss = history.history['val_loss']
epochs = range(1, len(training_loss) + 1)
# Plot the training and validation loss
plt.plot(epochs, training_loss, 'b', label='Training Loss')
plt.plot(epochs, validation_loss, 'r', label='Validation Loss')

# Label the plot
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

# Show the plot
plt.show()
```



```
# Specify the dimensions
fig, axes = plt.subplots(1,2)

# This makes the individual subplots
# Training Results
axes[0].scatter(x=y_train, y=model.predict(x_train))
axes[0].set_xlabel('Actual', fontsize=10)
axes[0].set_ylabel('Prediction', fontsize=10)
axes[0].set_title('Training')

# Add 45 deg line
x = np.linspace(*axes[0].get_xlim())
axes[0].plot(x, x, color='red')

# Validation Results
axes[1].scatter(x=y_test, y=model.predict(x_test))
axes[1].set_xlabel('Actual', fontsize=10)
axes[1].set_ylabel('Predicted', fontsize=10)
axes[1].set_title('Validation')

# add 45 deg line
x = np.linspace(*axes[1].get_xlim())
axes[1].plot(x, x, color='red')

# Tight layout
fig.tight_layout()
plt.show()
```

```
511/511 [==============================] - 2s 4ms/step
128/128 [==============================] - 1s 7ms/step
```

Predictive System

```
def pred(longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,ocean_proximity):
    features = np.array([longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,ocean_
    features_scaled = min_max_scaler.fit_transform([features])
    results = model.predict(features_scaled).reshape(1,-1)
    return results[0]
```

```
longitude = -122.2300
latitude  = 37.8800
housing_median_age  = 41.0000
total_rooms        = 880.0000
total_bedrooms   =  129.0000
population     = 322.0000
households          =    126.0000
median_income       =      8.3252
ocean_proximity       =     3.0000

price = pred(longitude,latitude,housing_median_age,total_rooms,total_bedrooms,population,households,median_income,ocean_proximity)

```

```
1/1 [==============================] - 0s 31ms/step
```

```
price #452600.0000 #actual value
```

```
array([270706.47], dtype=float32)
```