Assignment 5 - Rutvick Savaliya C0865187

Objectives

Artificial Neural Network (ANN) vs Linear Regression for California Housing Data

Objective: The objective of this assignment is to implement and compare the performance of an Artificial Neural Network (ANN) and Linear Regression for predicting house prices using the California Housing dataset. Dataset: Use the California Housing dataset, which contains various features related to housing in California. You can load the dataset using scikit-learn:

Tasks:

```
Data Preprocessing:
    Explore and understand the features of the dataset.
    Handle any missing values or outliers if present.
    Split the dataset into training and testing sets.

Linear Regression:
    Implement a Linear Regression model using scikit-learn.
    Train the model on the training set.
    Make predictions on the testing set.
    Evaluate the model's performance using appropriate regression metrics
(e.g., Mean Squared Error, R2 Score).

Artificial Neural Network (ANN):
    Implement a simple ANN for regression using a framework like TensorFlow or
Keras.
    Design the architecture of the neural network, including the input and
output layers.
    Train the ANN on the training set.
    Make predictions on the testing set.
    Evaluate the model's performance using the same regression metrics used for
Linear Regression.

Comparison and Analysis:
    Compare the performance metrics of the Linear Regression and ANN models.
    Discuss the strengths and weaknesses of each model.
    Analyze whether the complexity of an ANN provides better predictive
performance compared to Linear Regression.

Visualization:
    Create visualizations (e.g., scatter plots, line plots) to compare the
predicted values of the two models with the actual values.
    Visualize the model architectures if possible.

Conclusion:
    Summarize the key findings.
    Provide insights into which model performed better for predicting house
prices in the California Housing dataset.
    Discuss any challenges encountered during the implementation.
```

Submission: Prepare a report documenting the entire process, upload on Moodle and provide me the link for GitHub in the description or anywhere in the Moodle portal.

In [31]:
```
# Import Libraries
import numpy as np
import pandas as pd
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt
from pandas.plotting import scatter_matrix
```

In [2]:
```
# Load the California Housing dataset
housing_data = fetch_california_housing()
X, y = housing_data.data, housing_data.target
```

In [25]:
```
housing_data.feature_names
```

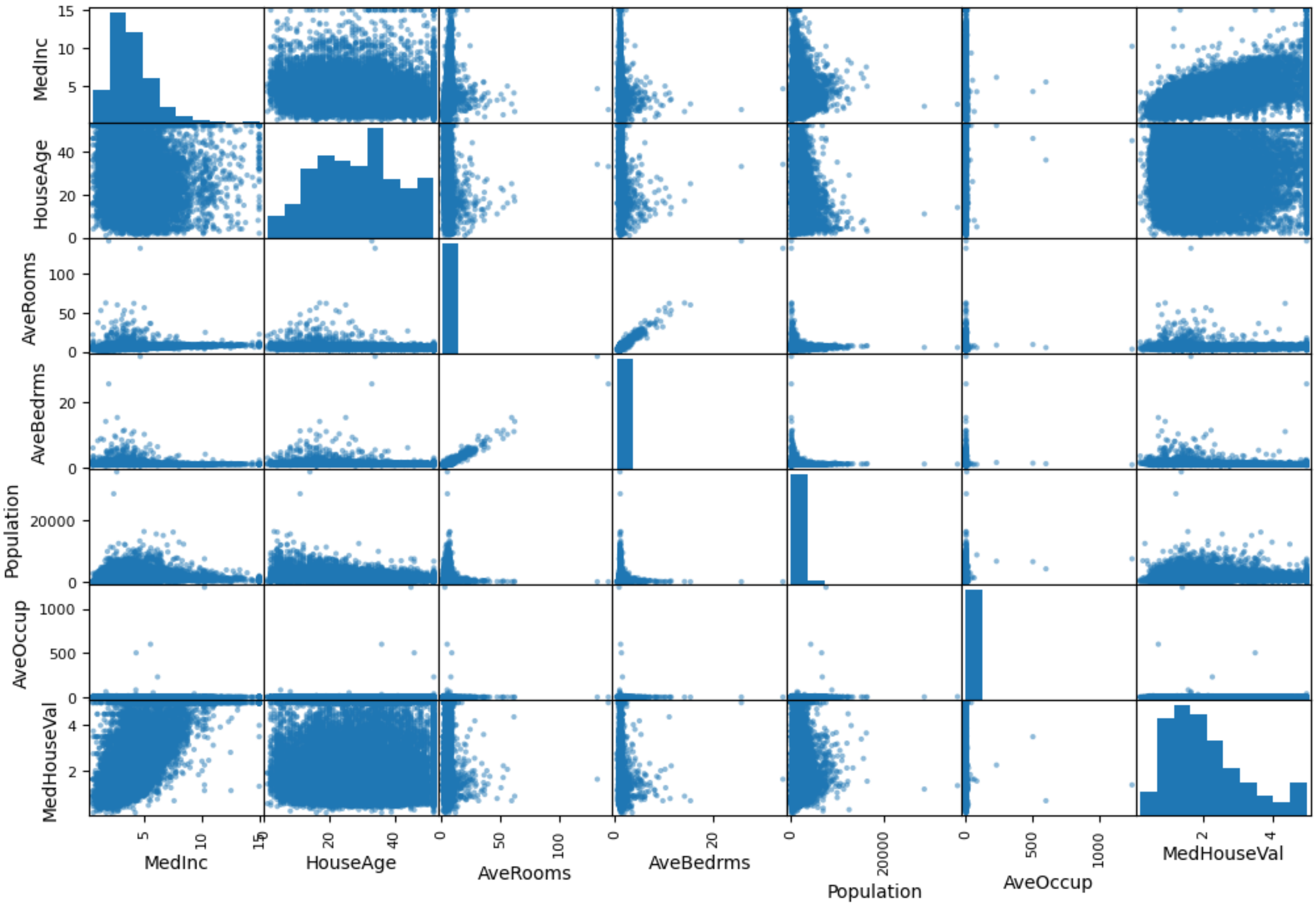
Out[25]:
```
['MedInc',
 'HouseAge',
 'AveRooms',
 'AveBedrms',
 'Population',
 'AveOccup',
 'Latitude',
 'Longitude']
```

In [29]:
```
housing = fetch_california_housing(as_frame=True)
housing = housing.frame
housing.head()
```

Out[29]:

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MedHouseVal
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23	4.526
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22	3.585
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24	3.521
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25	3.413
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25	3.422

In [32]:
```
attributes = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup','M
scatter_matrix(housing[attributes], figsize=(12,8))
plt.show()
```




```
In [33]: corr = housing.corr()
         corr['MedHouseVal'].sort_values(ascending=True)
```

```
Out[33]: Latitude      -0.144160
         AveBedrms     -0.046701
         Longitude     -0.045967
         Population    -0.024650
         AveOccup      -0.023737
         HouseAge       0.105623
         AveRooms       0.151948
         MedInc         0.688075
         MedHouseVal    1.000000
         Name: MedHouseVal, dtype: float64
```

```
In [35]: # Checking for null values
         housing.isna().sum()
```

```
Out[35]: MedInc         0
         HouseAge       0
         AveRooms       0
         AveBedrms      0
         Population     0
         AveOccup       0
         Latitude       0
         Longitude      0
         MedHouseVal    0
         dtype: int64
```

Linear Regression

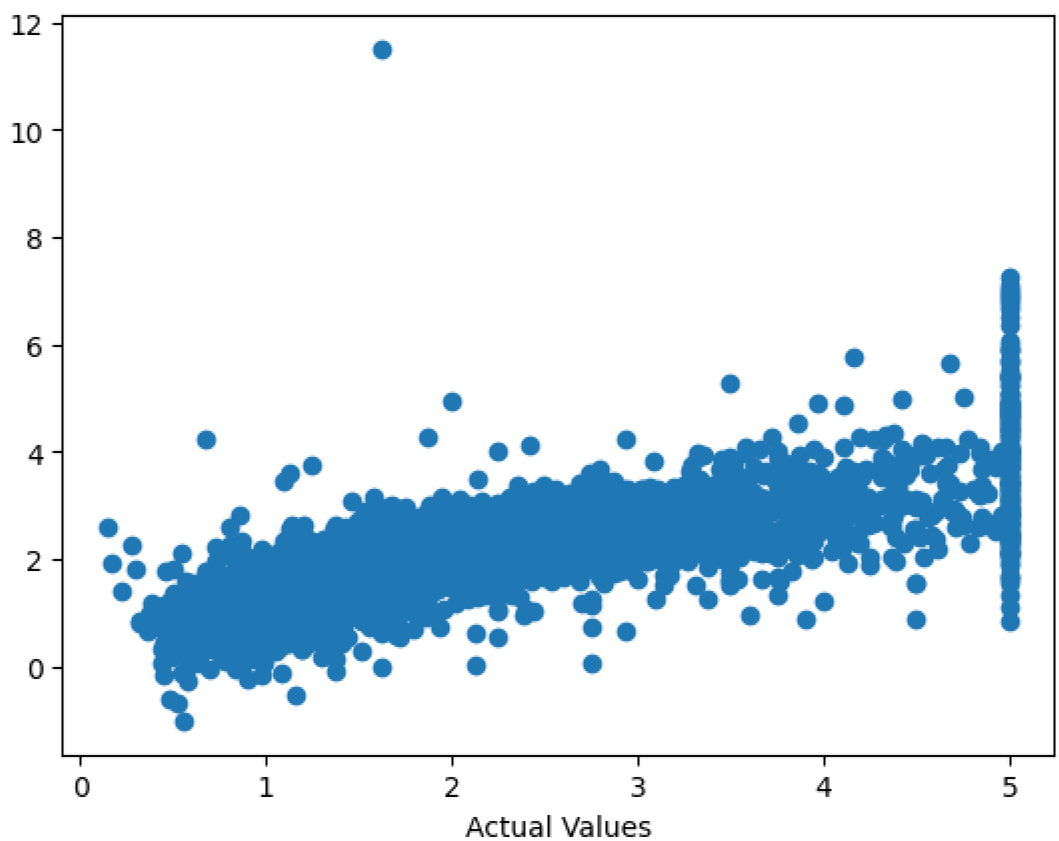
```
In [36]: # Split the dataset into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
```

```
# Implement a Linear Regression model
linear_model = LinearRegression()
```

In [37]:
```
# Train the model on the training set
linear_model.fit(X_train, y_train)

# Make predictions on the testing set
y_pred = linear_model.predict(X_test)

# Evaluate the model's performance
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Print performance metrics
print(f'Mean Squared Error: {mse}')
print(f'R2 Score: {r2}')

# Visualize the predicted vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
```

```
Mean Squared Error: 0.555891598695242
R2 Score: 0.5757877060324526
```

Out[37]:
```
Text(0.5, 0, 'Actual Values')
```



Artificial Neural Network (ANN)

In [38]:
```
# Importing the model
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense

ann_model = Sequential()
ann_model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
ann_model.add(Dense(1, activation='linear'))
ann_model.compile(optimizer='adam', loss='mean_squared_error')
```

In [39]:
```
# Training the data
ann_model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_tes
```

```
Epoch 1/50
516/516 [==============================] - 2s 2ms/step - loss: 388.5208 - val_loss: 2.06
46
Epoch 2/50
516/516 [==============================] - 1s 2ms/step - loss: 1.6165 - val_loss: 1.3141
Epoch 3/50
516/516 [==============================] - 1s 2ms/step - loss: 1.2149 - val_loss: 1.1055
Epoch 4/50
516/516 [==============================] - 1s 2ms/step - loss: 1.5307 - val_loss: 1.1403
Epoch 5/50
516/516 [==============================] - 1s 2ms/step - loss: 1.2344 - val_loss: 1.1086
Epoch 6/50
516/516 [==============================] - 1s 2ms/step - loss: 1.4288 - val_loss: 1.7835
Epoch 7/50
516/516 [==============================] - 1s 2ms/step - loss: 3.0682 - val_loss: 1.1704
Epoch 8/50
516/516 [==============================] - 1s 2ms/step - loss: 5.6383 - val_loss: 0.9284
Epoch 9/50
516/516 [==============================] - 1s 2ms/step - loss: 1.2580 - val_loss: 0.9619
Epoch 10/50
516/516 [==============================] - 1s 2ms/step - loss: 24.2572 - val_loss: 0.833
3
Epoch 11/50
516/516 [==============================] - 1s 2ms/step - loss: 6.3395 - val_loss: 0.8010
Epoch 12/50
516/516 [==============================] - 1s 2ms/step - loss: 1.5076 - val_loss: 0.7796
Epoch 13/50
516/516 [==============================] - 1s 2ms/step - loss: 1.1052 - val_loss: 0.7149
Epoch 14/50
516/516 [==============================] - 1s 2ms/step - loss: 9.9249 - val_loss: 4.9339
Epoch 15/50
516/516 [==============================] - 1s 2ms/step - loss: 4.0178 - val_loss: 0.8908
Epoch 16/50
516/516 [==============================] - 1s 2ms/step - loss: 1.1588 - val_loss: 0.7016
Epoch 17/50
516/516 [==============================] - 1s 2ms/step - loss: 5.1756 - val_loss: 0.7587
Epoch 18/50
516/516 [==============================] - 1s 2ms/step - loss: 7.9308 - val_loss: 10.168
6
Epoch 19/50
516/516 [==============================] - 1s 2ms/step - loss: 1.6906 - val_loss: 0.8376
Epoch 20/50
516/516 [==============================] - 1s 2ms/step - loss: 8.1866 - val_loss: 1.8031
Epoch 21/50
516/516 [==============================] - 1s 2ms/step - loss: 2.5948 - val_loss: 0.7873
Epoch 22/50
516/516 [==============================] - 1s 2ms/step - loss: 10.7802 - val_loss: 2.300
0
Epoch 23/50
516/516 [==============================] - 1s 2ms/step - loss: 3.8254 - val_loss: 1.0935
Epoch 24/50
516/516 [==============================] - 1s 2ms/step - loss: 11.3016 - val_loss: 0.612
4
Epoch 25/50
516/516 [==============================] - 1s 2ms/step - loss: 0.9886 - val_loss: 0.6751
Epoch 26/50
516/516 [==============================] - 1s 2ms/step - loss: 1.8814 - val_loss: 0.7872
Epoch 27/50
516/516 [==============================] - 1s 2ms/step - loss: 5.2131 - val_loss: 19.867
7
Epoch 28/50
516/516 [==============================] - 1s 2ms/step - loss: 30.7247 - val_loss: 0.607
1
```

```
Epoch 29/50
516/516 [==============================] - 1s 2ms/step - loss: 0.6820 - val_loss: 0.6177
Epoch 30/50
516/516 [==============================] - 1s 2ms/step - loss: 1.7895 - val_loss: 0.6327
Epoch 31/50
516/516 [==============================] - 1s 2ms/step - loss: 1.2578 - val_loss: 0.6224
Epoch 32/50
516/516 [==============================] - 1s 2ms/step - loss: 4.5140 - val_loss: 0.8206
Epoch 33/50
516/516 [==============================] - 1s 2ms/step - loss: 10.3189 - val_loss: 0.607
9
Epoch 34/50
516/516 [==============================] - 1s 2ms/step - loss: 3.2966 - val_loss: 0.8652
Epoch 35/50
516/516 [==============================] - 1s 2ms/step - loss: 3.5715 - val_loss: 0.6381
Epoch 36/50
516/516 [==============================] - 1s 2ms/step - loss: 0.8817 - val_loss: 0.6030
Epoch 37/50
516/516 [==============================] - 1s 2ms/step - loss: 1.7741 - val_loss: 0.6134
Epoch 38/50
516/516 [==============================] - 1s 2ms/step - loss: 2.0672 - val_loss: 0.7639
Epoch 39/50
516/516 [==============================] - 1s 2ms/step - loss: 4.9203 - val_loss: 1.2553
Epoch 40/50
516/516 [==============================] - 1s 2ms/step - loss: 25.5611 - val_loss: 32.59
68
Epoch 41/50
516/516 [==============================] - 1s 2ms/step - loss: 2.4261 - val_loss: 0.6638
Epoch 42/50
516/516 [==============================] - 1s 2ms/step - loss: 1.0502 - val_loss: 0.6216
Epoch 43/50
516/516 [==============================] - 1s 2ms/step - loss: 0.8611 - val_loss: 0.8127
Epoch 44/50
516/516 [==============================] - 1s 2ms/step - loss: 1.4150 - val_loss: 0.6972
Epoch 45/50
516/516 [==============================] - 1s 2ms/step - loss: 3.2833 - val_loss: 0.6531
Epoch 46/50
516/516 [==============================] - 1s 2ms/step - loss: 1.6082 - val_loss: 0.7523
Epoch 47/50
516/516 [==============================] - 1s 2ms/step - loss: 2.0990 - val_loss: 0.7619
Epoch 48/50
516/516 [==============================] - 1s 2ms/step - loss: 4.5446 - val_loss: 0.6257
Epoch 49/50
516/516 [==============================] - 1s 2ms/step - loss: 0.9967 - val_loss: 0.9675
Epoch 50/50
516/516 [==============================] - 1s 2ms/step - loss: 3.3029 - val_loss: 0.6265
```

Out[39]:
```
<keras.callbacks.History at 0x1d73e775c60>
```

In [40]:
```python
# make predictions
y_pred_ann = ann_model.predict(X_test)
```

```
129/129 [==============================] - 0s 1ms/step
```

In [41]:
```python
# evaluating the performance
mse_ann = mean_squared_error(y_test, y_pred_ann)
r2_ann = r2_score(y_test, y_pred_ann)
print(f'Mean Squared Error (ANN): {mse_ann}')
print(f'R2 Score (ANN): {r2_ann}')
```

```
Mean Squared Error (ANN): 0.6264570945928923
R2 Score (ANN): 0.5219377270797916
```

Conclusion

In [43]:
```python
# Print performance metrics of Linear Rgression
print('--- Performance metrics of Linear Regression ---')
print(f'Mean Squared Error: {mse}')
print(f'R2 Score: {r2}')

print("")
# Print performance metrics of ANN
print('--- Performance metrics of ANN ---')
print(f'Mean Squared Error (ANN): {mse_ann}')
print(f'R2 Score (ANN): {r2_ann}')
```

```
--- Performance metrics of Linear Regression ---
Mean Squared Error: 0.555891598695242
R2 Score: 0.5757877060324526

--- Performance metrics of ANN ---
Mean Squared Error (ANN): 0.6264570945928923
R2 Score (ANN): 0.5219377270797916
```

From the above comparison we can conclude that the Liner Regression model is performing slightly better compare to ANN as the MSE of LR is lesser than that of ANN.