Saiful Islam No: 12143891
Umut Kandemir No:12329185

Deep Learning KU (708.220) WS23

Assignment 2: Training Neural Networks

a) (3 pts) : Get familiar with the dataset. Construct a validation set consisting of samples from the training data, which will be used during the model selection process. You will use the test set only for final evaluations. Investigate the feature distributions, and normalize the data if it is necessary.

Code:

In [26]:
```
import pickle
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
from tensorflow.keras.callbacks import LearningRateScheduler
```

In [28]:
```
california_data = pickle.load(open('california-housing-dataset.pkl', 'rb')) # Load the data with pickle and readning it as d
x_train, y_train = california_data['x_train'], california_data['y_train']
x_test, y_test = california_data['x_test'], california_data['y_test']
```

In [29]:
```
print("Training set dimensions:", x_train.shape, y_train.shape)
```

```
Training set dimensions: (15480, 8) (15480,)
```

In [30]:
```
print("Shape of x_train:", x_train.shape)
print("Shape of y_train:", y_train.shape)
print("Sample x_train:", x_train[0])
print("Sample y_train:", y_train[0])
```

```
Shape of x_train: (15480, 8)
Shape of y_train: (15480,)
Sample x_train: [ 2.66180000e+00  1.00000000e+01  6.20408163e+00  1.25510204e+00
  1.40300000e+03  2.04518950e+00  3.41000000e+01 -1.16320000e+02]
Sample y_train: 0.81
```

In [31]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.2, random_state=42)
```

In [32]:
```
feature_names = ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']

for i in range(x_train.shape[1]):
    sns.histplot(x_train[:, i], bins=50, kde=True)
    plt.xlabel(feature_names[i])  # Use the feature name as x-axis label
    plt.ylabel('Frequency')
    plt.show()
```

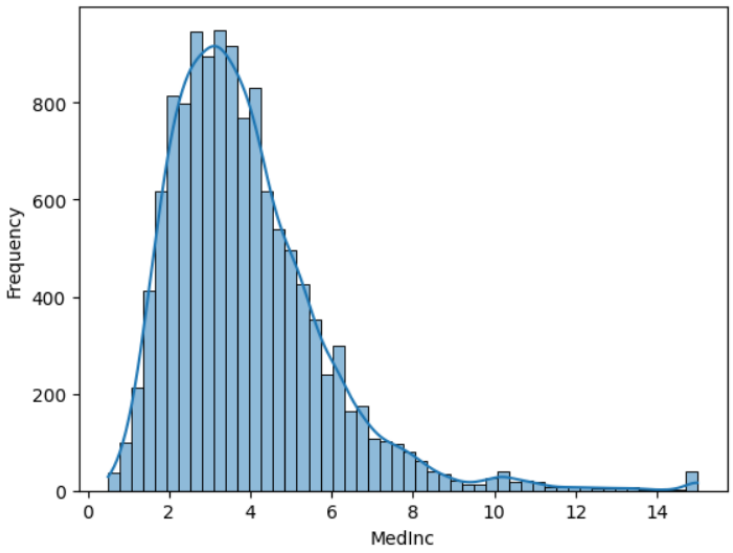
At task number a, it is needed to understand the structure of the data and getting familiar with data set. First ,program loads the dataset using the pickle module and extracts the training&test sets “x_train”,”y_train”,”x_test”,”y_test” and displays the dimensions of the set. After loading and extraction, the shape of the training data printed. The print

statements show the values of the first example in the training set (x_train[0]) and its corresponding label (y_train[0]).

According to task, its essential to construct a validation set. To make this, the training data splitted into training and validation sets with the test size 0.2. Which determines the proportion of the dataset, in this case 0.2 refers to original dataset and remaining 0.8 refers to training . As a test, we defined random_state as 42 means data split will be same everytime when we execute the code. This will provide us the consistency of the results and we'll be able to reach better understanding of the results.

For loop created to repeat over each feature, 8 feature in total, and creates a histogram using Seaborn's "histplot" function. After setting the bins and adding kernal density, plots displayed.

Below histogram of the median income , histogram displayed below shows us the distribution of the median income values. Other feature histograms will be shown at the bottom.



The x axis represents the median income and the y axis represents the frequency. As we mentioned above, histogram gives us idea about the distribution of median incomes in the dataset. In this case, we can come to conclusion that histogram is right-skewed. It is clear to define it as right-skewed since there are more districts with lower median incomes and less with higher median incomes

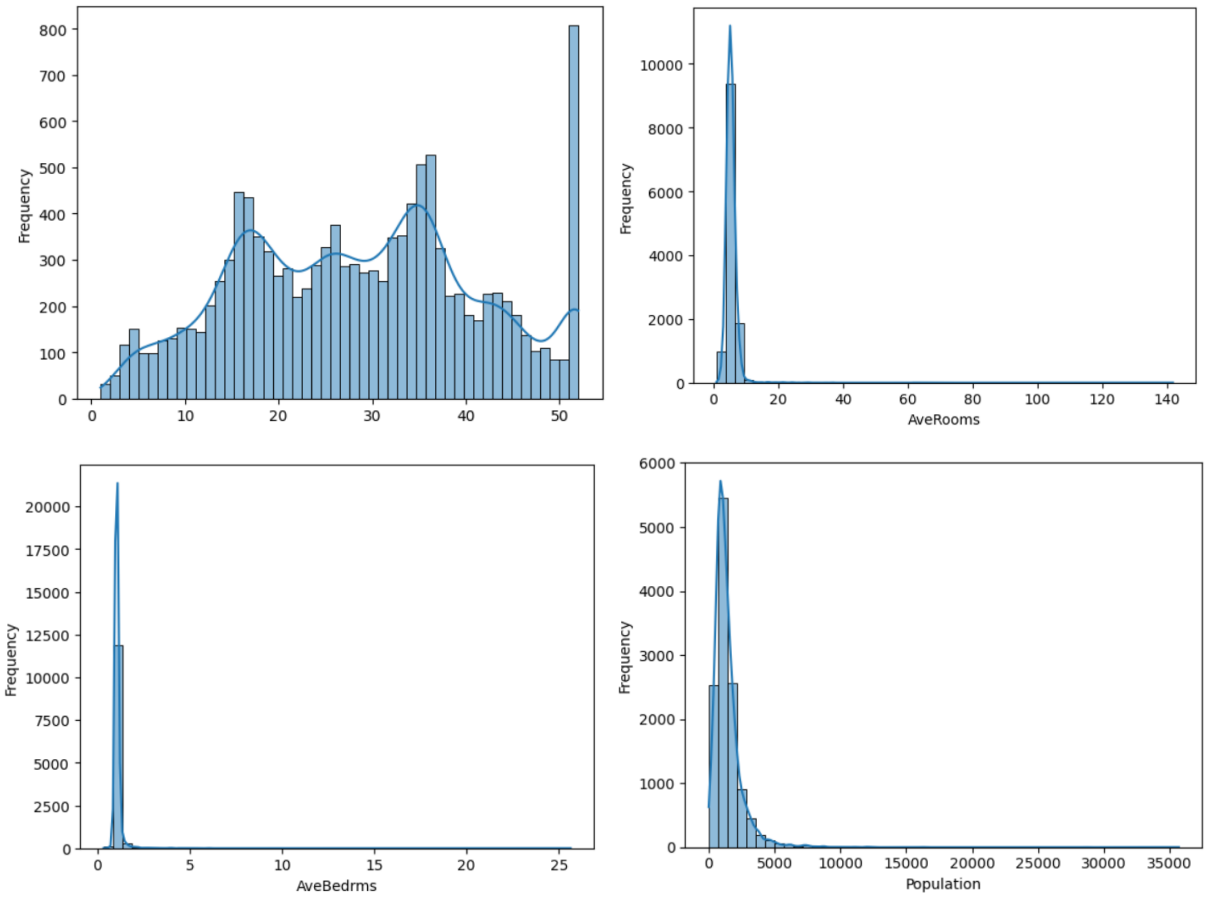
Saiful Islam No: 12143891
Umut Kandemir No:12329185

In [33]:
```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

x_train_normalized = scaler.fit_transform(x_train)
x_val_normalized = scaler.transform(x_val)
```

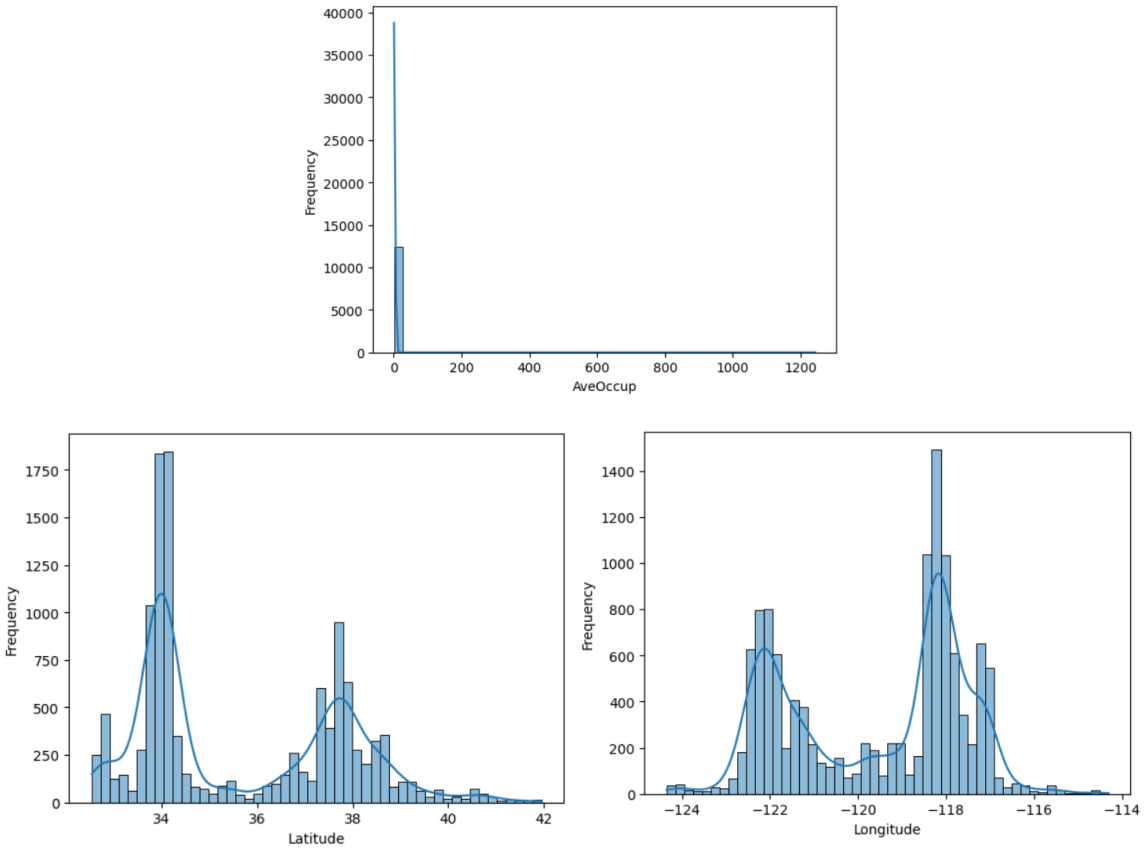
In [34]:
```
print("Sample normalized x_train:", x_train_normalized[0])
print("Sample normalized x_val:", x_val_normalized[0])
```

```
Sample normalized x_train: [-0.24641734 -1.63084141  0.10467077 -0.06579177  0.68050224  0.0337
7491
 -0.59199349  1.11993362]
Sample normalized x_val: [ 0.12303892  0.66191525 -0.06989343 -0.09129668 -0.12244089  0.022851
62
 -0.74176514  0.81603931]
```

Lastly, data is normalized. “StandardScaler()” is used to ensure that all inputs have the same scale. Normalized data printed and displayed. Below, remaining histograms for other 7 features can be seen.



Saiful Islam No: 12143891
Umut Kandemir No:12329185



Lastly, data is normalized. “StandardScale()” is used to ensure that all inputs have the same scale. Normalized data printed and displayed.

b) (8 pts) : Design your neural network architecture for the regression task. Explain your choices for the output layer and the error function that you will use. Minimize the error by using mini-batches of suitable size. Test different architectures with varying numbers of hidden units and hidden layers. Compare these choices and report training and validation set errors in a table.

Code:

In [35]:
```
model = keras.Sequential([
    keras.layers.Dense(units=64, activation='relu', input_shape=(x_train_normalized.shape[1],)),
    keras.layers.Dense(units=32, activation='relu'),
    keras.layers.Dense(units=1)  # Output layer for regression task
])
```

In [36]:
```
model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
```

In [37]:
```
batch_size = 32
epochs = 10

history = model.fit(
    x_train_normalized, y_train,
    batch_size=batch_size,
    epochs=epochs,
    validation_data=(x_val_normalized, y_val),
    verbose=2
)
```

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
Epoch 1/10
387/387 - 1s - loss: 1.0336 - mae: 0.6726 - val_loss: 0.5193 - val_mae: 0.4940 - 1s/epoch - 4ms/step
Epoch 2/10
387/387 - 0s - loss: 0.4546 - mae: 0.4788 - val_loss: 0.4058 - val_mae: 0.4510 - 417ms/epoch - 1ms/step
Epoch 3/10
387/387 - 0s - loss: 0.4089 - mae: 0.4525 - val_loss: 0.3693 - val_mae: 0.4345 - 414ms/epoch - 1ms/step
Epoch 4/10
387/387 - 1s - loss: 0.3900 - mae: 0.4404 - val_loss: 0.3632 - val_mae: 0.4210 - 506ms/epoch - 1ms/step
Epoch 5/10
387/387 - 0s - loss: 0.3861 - mae: 0.4318 - val_loss: 0.3560 - val_mae: 0.4213 - 420ms/epoch - 1ms/step
Epoch 6/10
387/387 - 0s - loss: 0.3916 - mae: 0.4300 - val_loss: 0.3504 - val_mae: 0.4208 - 413ms/epoch - 1ms/step
Epoch 7/10
387/387 - 0s - loss: 0.3546 - mae: 0.4173 - val_loss: 0.3251 - val_mae: 0.4048 - 449ms/epoch - 1ms/step
Epoch 8/10
387/387 - 0s - loss: 0.3442 - mae: 0.4090 - val_loss: 0.3200 - val_mae: 0.4053 - 425ms/epoch - 1ms/step
Epoch 9/10
387/387 - 0s - loss: 0.3348 - mae: 0.4040 - val_loss: 0.3232 - val_mae: 0.3945 - 395ms/epoch - 1ms/step
Epoch 10/10
387/387 - 0s - loss: 0.3349 - mae: 0.4009 - val_loss: 0.3124 - val_mae: 0.3851 - 410ms/epoch - 1ms/step
```

In [38]:

```
train_loss, train_mae = model.evaluate(x_train_normalized, y_train, verbose=0)
val_loss, val_mae = model.evaluate(x_val_normalized, y_val, verbose=0)

print(f"Training Set - Loss: {train_loss}, MAE: {train_mae}")
print(f"Validation Set - Loss: {val_loss}, MAE: {val_mae}")
```

```
Training Set - Loss: 0.3507591187953949, MAE: 0.3877808749675751
Validation Set - Loss: 0.31237438321113586, MAE: 0.38508906960487366
```

In [44]:

```
layers = [
    [64, 32],
    [128, 64, 32],
    [256, 128, 64, 32],  # Adding more layers
    [64, 64, 64],        # More units in each layer
    [128, 128, 64, 32],   # Balanced number of units
    [64, 128, 256, 128, 64],  # Increasing and then decreasing units
]

results = []

for arch in layers:
    model = keras.Sequential([
        keras.layers.Dense(units=units, activation='relu', input_shape=(x_train_normalized.shape[1],))
        for units in arch
    ])
    model.add(keras.layers.Dense(units=1))  # Output layer for regression task

    model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

    history = model.fit(
        x_train_normalized, y_train,
        batch_size=batch_size,
        epochs=epochs,
        validation_data=(x_val_normalized, y_val),
        verbose=0
    )

    train_loss, _ = model.evaluate(x_train_normalized, y_train, verbose=0)
    val_loss, _ = model.evaluate(x_val_normalized, y_val, verbose=0)

    results.append({'layers': arch, 'train_loss': train_loss, 'val_loss': val_loss})

for result in results:
    print(f"layers: {result['layers']}, Train MSE: {result['train_loss']}, Val MSE: {result['val_loss']}")
```

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
layers: [64, 32], Train MSE: 0.3334043025970459, Val MSE: 0.2984708547592163
layers: [128, 64, 32], Train MSE: 0.29775676131248474, Val MSE: 0.2883550822734833
layers: [256, 128, 64, 32], Train MSE: 0.28082725405693054, Val MSE: 0.27340200543403625
layers: [64, 64, 64], Train MSE: 0.29555457830429077, Val MSE: 0.2894843518733978
layers: [128, 128, 64, 32], Train MSE: 0.2819809317588806, Val MSE: 0.27532896399497986
layers: [64, 128, 256, 128, 64], Train MSE: 0.28669166564941406, Val MSE: 0.299778014421463
```

In [45]:
```
results_df = pd.DataFrame(results)
results_df
```

Out[45]:

	layers	train_loss	val_loss
0	[64, 32]	0.333404	0.298471
1	[128, 64, 32]	0.297757	0.288355
2	[256, 128, 64, 32]	0.280827	0.273402
3	[64, 64, 64]	0.295555	0.289484
4	[128, 128, 64, 32]	0.281981	0.275329
5	[64, 128, 256, 128, 64]	0.286692	0.299778

Interperations:

Since we have regression task like predicting the house prices, we choose **output layer** as single dense layer with one unit. We believe it is an apporpriate choice to predict for the median house value. **For Error function**, we choose MSE(Mean Squared Error). Reason behind it is MSE ables to penalizes larger errors better than smaller errors and in our case since we have a regression task, it will be important to accurately predict the magnitude of the target variable. For house price prediction, we need to accurately estimate the difference between predicted and actual values which is critical. By choosing the combination of single dense output layer with one unit and the MSE function suits well for our task to predict house prices

Mini-batch training applied to minimize the error, which is efficient to apply it on large datasets. Batch size selected as 32 to balance computational effiency and model convergence. Different architectures tested and all models are compiled with the help of Adam optimizer and the chosen MSE function and it trained for 10 epochs on the normalized training set by using mini batches.

To compare the training and validation set MSE for each architecture are displayed on the table as its needed at task. When we examine the table, we can say that models generally had reasonable performance with MSE values range approximately from 0.27 to 0.33. As a architectural comparison, [256,128,64,32] hidden units performs the best in terms of MSE value. Lower MSE value indicates better performance. In this case [256,128,64,32] has the best performance among other models. We can say that more complex model performs better.

c) (5 pts) : Investigate and compare different optimization procedures such as stochastic gradient descent (SGD), momentum SGD, and ADAM. Accordingly, try a number of learning rates and also try out adapting the learning rate during training by scheduling. Provide a table where training and validation set errors of various optimization hyper-parameters are compared.

Saiful Islam No: 12143891
Umut Kandemir No:12329185

Code:

```
from tensorflow.keras.optimizers import SGD, Adam
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers.schedules import ExponentialDecay
import tensorflow as tf
import pandas as pd

def create_and_compile_model(optimizer):
    model = Sequential()
    model.add(Dense(units=64, activation='relu', input_shape=(x_train_normalized.shape[1],)))
    model.add(Dense(units=32, activation='relu'))
    model.add(Dense(units=1))  # Output layer for regression task

    model.compile(optimizer=optimizer, loss='mean_squared_error', metrics=['mae'])
    return model

def clipped_momentum_optimizer(learning_rate_schedule):
    optimizer = SGD(learning_rate=learning_rate_schedule, momentum=0.9)

    @tf.function
    def train_step(inputs, targets):
        with tf.GradientTape() as tape:
            predictions = model(inputs)
            loss = model.compiled_loss(targets, predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        clipped_gradients = [tf.clip_by_norm(grad, 1.0) for grad in gradients]
        optimizer.apply_gradients(zip(clipped_gradients, model.trainable_variables))
        return loss

    return optimizer, train_step
```

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
results_optimizers = []

for optimizer_name in optimizers:
    for learning_rate in learning_rates:
        if optimizer_name == 'sgd':
            optimizer = SGD(learning_rate=learning_rate)
            model = create_and_compile_model(optimizer)
        elif optimizer_name == 'momentum':
            # Use learning rate scheduling with clipped gradients for Momentum optimizer
            learning_rate_schedule = ExponentialDecay(
                initial_learning_rate=learning_rate,
                decay_steps=10000,
                decay_rate=0.9,
                staircase=True
            )
            optimizer, train_step = clipped_momentum_optimizer(learning_rate_schedule)
        elif optimizer_name == 'adam':
            optimizer = Adam(learning_rate=learning_rate)
            model = create_and_compile_model(optimizer)

        history = model.fit(
            x_train_normalized, y_train,
            batch_size=batch_size,
            epochs=epochs,
            validation_data=(x_val_normalized, y_val),
            verbose=0
        )

        train_loss, _ = model.evaluate(x_train_normalized, y_train, verbose=0)
        val_loss, _ = model.evaluate(x_val_normalized, y_val, verbose=0)

        results_optimizers.append({
            'optimizer': optimizer_name,
            'learning_rate': learning_rate,
            'train_loss': train_loss,
            'val_loss': val_loss
        })
```

```
results_df_optimizers = pd.DataFrame(results_optimizers)
results_df_optimizers
```

	optimizer	learning_rate	train_loss	val_loss
0	sgd	0.0100	0.335123	0.314088
1	sgd	0.0010	0.476922	0.452220
2	sgd	0.0001	0.963634	0.964968
3	momentum	0.0100	0.717072	0.690924
4	momentum	0.0010	0.660352	0.631619
5	momentum	0.0001	0.622399	0.594543
6	adam	0.0100	0.307279	0.290677
7	adam	0.0010	0.333190	0.309473
8	adam	0.0001	0.453925	0.427494

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
def create_model(optimizer='adam', learning_rate=0.001):
    model = keras.Sequential()

    model.add(keras.layers.InputLayer(input_shape=(x_train_normalized.shape[1],)))

    model.add(keras.layers.Dense(units=128, activation='relu'))
    model.add(keras.layers.Dense(units=64, activation='relu'))

    model.add(keras.layers.Dense(units=1))

    model.compile(optimizer=optimizer(learning_rate=learning_rate), loss='mean_squared_error')

    return model

def lr_schedule(epoch):
    if epoch < 10:
        return 0.001
    elif epoch < 20:
        return 0.0001
    else:
        return 0.00001

optimizers = [('SGD', keras.optimizers.SGD), ('MomentumSGD', keras.optimizers.SGD), ('Adam', keras.optimizers.Adam)]
learning_rates = [0.01, 0.001, 0.0001]
scheduled_results = []

for opt_name, optimizer in optimizers:
    for lr in learning_rates:
        model = create_model(optimizer=optimizer, learning_rate=lr)

        history = model.fit(
            x_train_normalized, y_train,
            validation_data=(x_val_normalized, y_val),
            epochs=20, batch_size=32, verbose=0,  # Adjust epochs and batch_size as needed
            callbacks=[LearningRateScheduler(lr_schedule)]  # Use learning rate schedule
        )
```

```
        train_mse = mean_squared_error(y_train, model.predict(x_train_normalized))
        val_mse = mean_squared_error(y_val, model.predict(x_val_normalized))

        scheduled_results.append({
            'Optimizer': opt_name,
            'Learning Rate': lr,
            'Train MSE': train_mse,
            'Validation MSE': val_mse
        })

scheduled_results_df = pd.DataFrame(scheduled_results)
results_df.head()
```

```
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
387/387 [==============================] - 1s 2ms/step
97/97 [==============================] - 0s 2ms/step
387/387 [==============================] - 1s 1ms/step
97/97 [==============================] - 0s 1ms/step
```

Saiful Islam No: 12143891
Umut Kandemir No:12329185

Out[11]:

	Architecture	Train MSE	Validation MSE
0	1 layers, 32 units	0.513484	0.519450
1	1 layers, 64 units	0.455158	0.465864
2	2 layers, 64 units	0.478749	0.491555
3	2 layers, 128 units	0.466545	0.479707

In [12]:
```
results_df
```

Out[12]:

	Architecture	Train MSE	Validation MSE
0	1 layers, 32 units	0.513484	0.519450
1	1 layers, 64 units	0.455158	0.465864
2	2 layers, 64 units	0.478749	0.491555
3	2 layers, 128 units	0.466545	0.479707

Interperations:

Different optimization procedures such as SGD,momentum SGD and ADAM investigated. Learning rate scheduling and architectural experiments have been done.

Obtained results are: Learning rates are used to control how fast or slow the model learns. When we look at the SGD and momentum SGD, smaller sensitivity rate often performs better. ADAM achieves faster convergence and lower validation loss. With a higher learning rate(0.1000) ADAM learns fast with low validation loss. In general after experiments, we come up conclusions that ADAM i a smart learner at our neural network , when we use ADAM as a optimizer, it adapts well without needing many adjustments. We can call it as flexible.

SGD optimizer works through data slowly but steadily. It doesn't rush and makes small steps to minimize error. It can be very useful at large and complex data since it is harder to deal with it and by using SGD optimizer, we can feel more comfartable since it will handle the process more cautious. SGD momentum provided us good results especially at low learning rate but it was slower. ADAM consistenly provided lower losses compared to SGD and SGD momentum. It demonstrated better optimization results in training and validation ser errors.

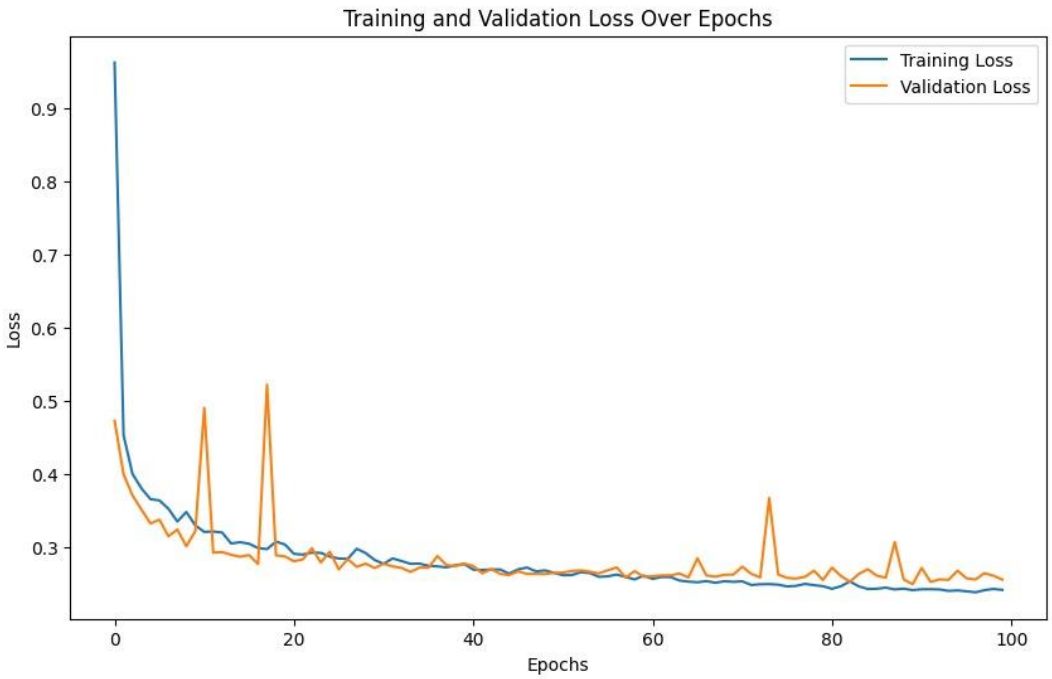
d) (4 pts) : Clearly summarize your final model once your architecture choices are fixed. Provide a plot where the evolution of the training and validation set errors during training are shown throughout iterations. Perform a final training with this model on the whole training set. Report and comment on the final test error. Provide a scatter plot in which you compare model predictions with their ground truth values (on the test set).

Code:

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
final_model = keras.Sequential([
    keras.layers.Dense(units=64, activation='relu', input_shape=(x_train_normalized.shape[1],)),
    keras.layers.Dense(units=32, activation='relu'),
    keras.layers.Dense(units=1)  # Output layer for regression task
])

final_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

final_epochs = 100
history_final = final_model.fit(
    x_train_normalized, y_train,
    batch_size=batch_size,
    epochs=final_epochs,
    validation_data=(x_val_normalized, y_val),
    verbose=2
)
```

```
plt.figure(figsize=(10, 6))
plt.plot(history_final.history['loss'], label='Training Loss')
plt.plot(history_final.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

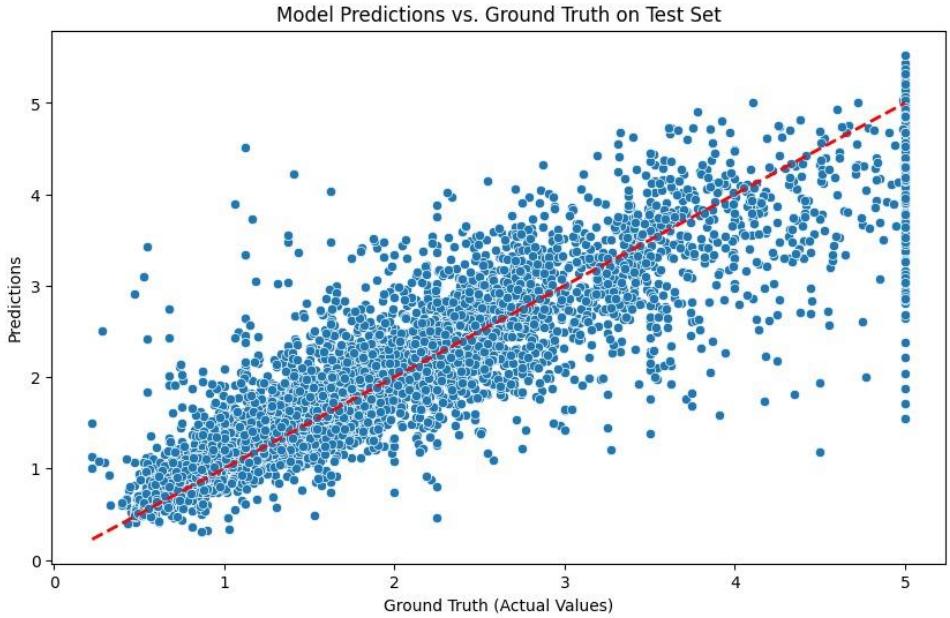


Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
x_test_normalized = scaler.transform(x_test)
y_pred = final_model.predict(x_test_normalized)
test_loss = mean_squared_error(y_test, y_pred)
print(f"Final Test Error (MSE): {test_loss}")
```

```
162/162 [==============================] - 0s 1ms/step
Final Test Error (MSE): 0.269711152040648
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x=y_test.flatten(), y=y_pred.flatten())
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], color='red', linestyle='--', linewidth=2)
plt.title('Model Predictions vs. Ground Truth on Test Set')
plt.xlabel('Ground Truth (Actual Values)')
plt.ylabel('Predictions')
plt.show()
```



Interpretations:

Until now out final test error, measured by Mean Squared Error (MSE), is 0.2697, which indicates the average squared difference between the predicted median house values and the actual values in the test set. We know a lower MSE suggests better model performance, and in this case, our obtained value is reasonable, considering the complexity of predicting housing prices.

Our training process involved 100 epochs, and the model demonstrated a consistent decrease in both training and validation errors throughout the epochs. The training and validation Mean Absolute Error (MAE) values gradually decreased, that means our model learned to make predictions with higher precision. In general it can be said that model seems to be performing well, with conssitent improvement in both training and validation metrics.

Saiful Islam No: 12143891
Umut Kandemir No:12329185

e) (5 pts) : Now assume that we want to use a similar architecture for the binary classification problem of determining if the median house value is below or over $\$200,000$. Explain which parts of the architecture and the training pipeline you would need to change, and which test set evaluation metrics should be investigated in this case. Explain the reasons for these differences.

Implement these changes to the architecture and the training pipeline you had in part (d), and train a single model for this binary classification task using the whole training set. Note that you will also need to redefine your target variables by simply executing the following lines in the order:

```
y_train[y_train<2], y_test[y_test<2] = 0, 0
y_train[y_train>=2], y_test[y_test>=2] = 1, 1
```

Evaluate your model on the test set, report and comment on its performance.

Code:

```
y_train[y_train < 2], y_test[y_test < 2] = 0, 0
y_train[y_train >= 2], y_test[y_test >= 2] = 1, 1

binary_classification_model = keras.Sequential([
    keras.layers.Dense(units=64, activation='relu', input_shape=(x_train_normalized.shape[1],)),
    keras.layers.Dense(units=32, activation='relu'),
    keras.layers.Dense(units=1, activation='sigmoid')  # One neuron for binary classification
])

binary_classification_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])


y_train = np.squeeze(y_train)

binary_classification_model.fit(
    x_train_normalized, y_train,
    batch_size=batch_size,
    epochs=final_epochs,
    verbose=2
)


x_test_normalized = scaler.transform(x_test)
y_pred_prob = binary_classification_model.predict(x_test_normalized)
y_pred_binary = (y_pred_prob[:, 0] > 0.5).astype("int32")

accuracy = accuracy_score(y_test, y_pred_binary)

print(f"Accuracy: {accuracy}")
```

Saiful Islam No: 12143891
Umut Kandemir No:12329185

```
y_pred_binary = (binary_classification_model.predict(x_test_normalized) > 0.5).astype("int32")

precision = precision_score(y_test, y_pred_binary)
recall = recall_score(y_test, y_pred_binary)
f1 = f1_score(y_test, y_pred_binary)

conf_matrix = confusion_matrix(y_test, y_pred_binary)
print("Confusion Matrix:")
print(conf_matrix)

print(f"Accuracy: {accuracy}")
print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
```

```
162/162 [==============================] - 0s 1ms/step
Confusion Matrix:
[[2686  274]
 [ 358 1842]]
Accuracy: 0.8775193798449612
Precision: 0.8705103969754253
Recall: 0.8372727272727273
F1 Score: 0.8535681186283596
```

Interpretations:

Accuracy: Our model achieved an accuracy of approximately 87.75%, which indicates that the model is making correct predictions for a large portion of the test set.

Confusion Matrix: From the confusion matrix, we see the model has 2686 true negatives (TN), 274 false positives (FP), 358 false negatives (FN), and 1842 true positives (TP). The high number of true positives and true negatives indicates that the model is effective in both identifying instances where the median house value is below 200,000 and where it is above 200,000.

Training Efficiency: Notably, the model attained a significant level of accuracy a small number of epochs (100), highlighting its efficiency in learning from the dataset.