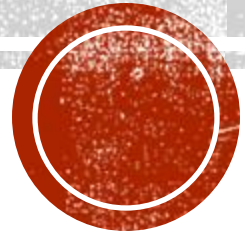
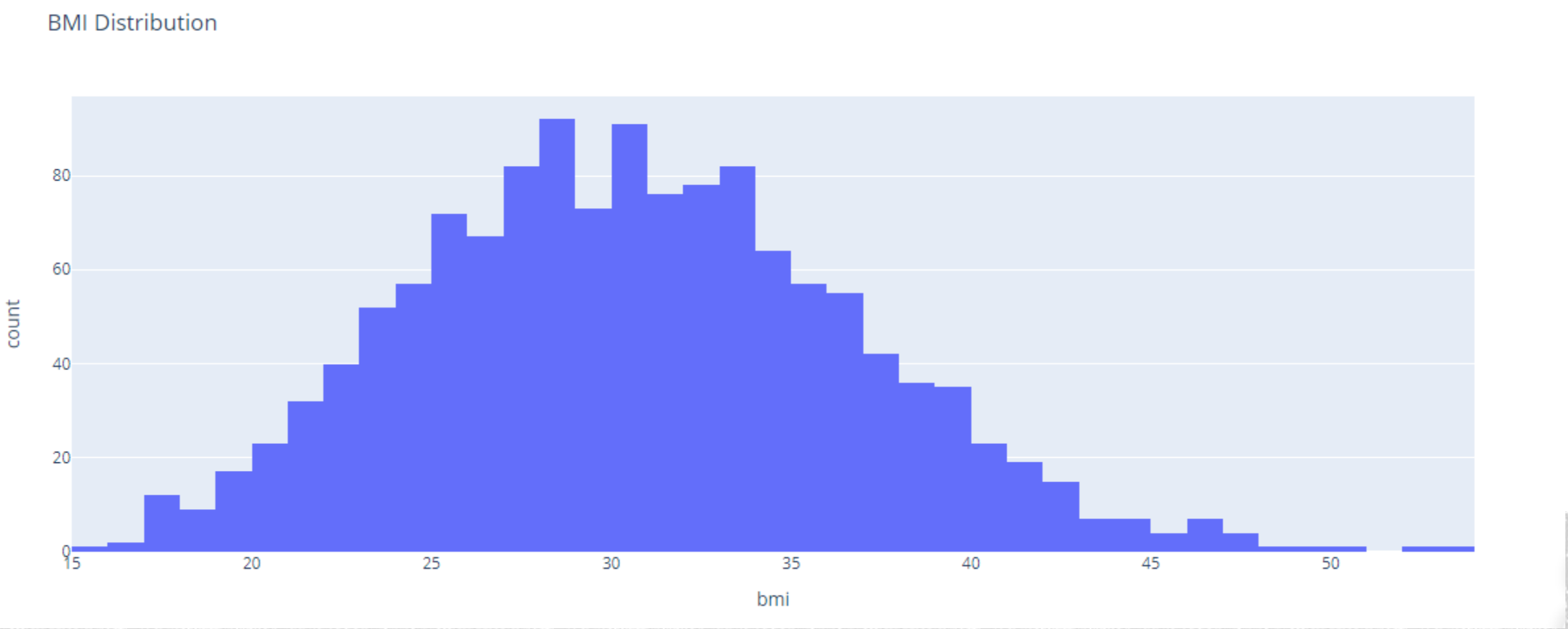
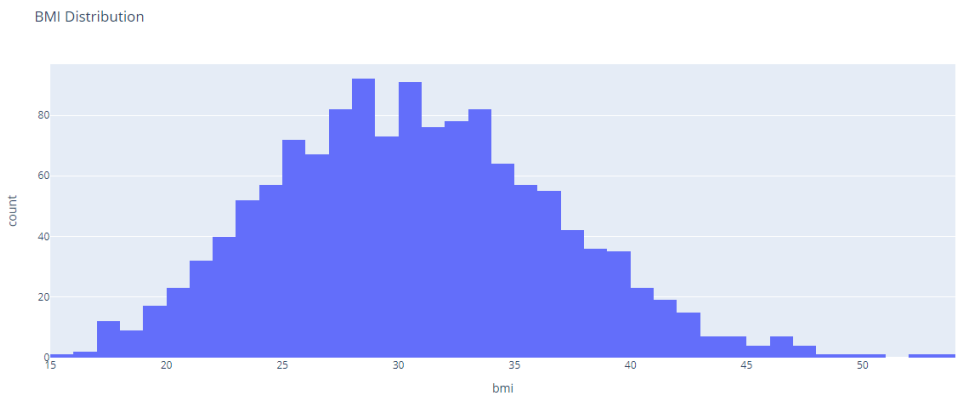
PREDICTING MEDICAL INSURANCE COSTS USING KERAS SEQUENTIAL

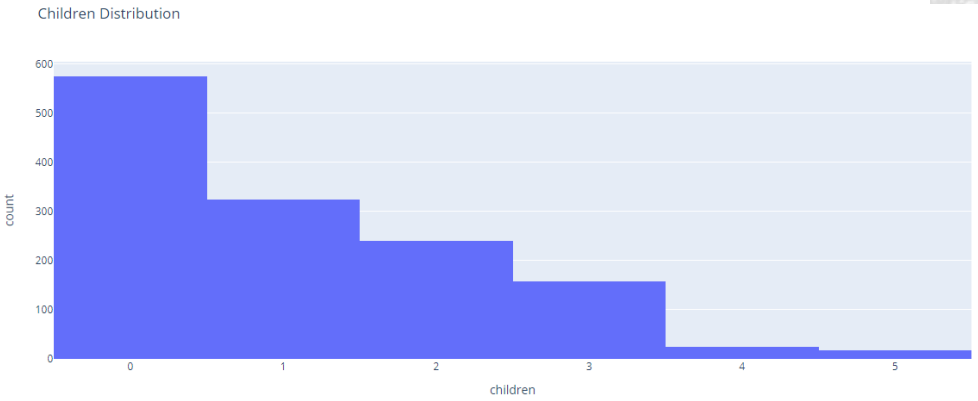
Assignment 3




BMI Distribution
count
0
20
40
60
80
15
20
25
30
35
40
45
50
bmi

UNIVARIATE ANALYSIS

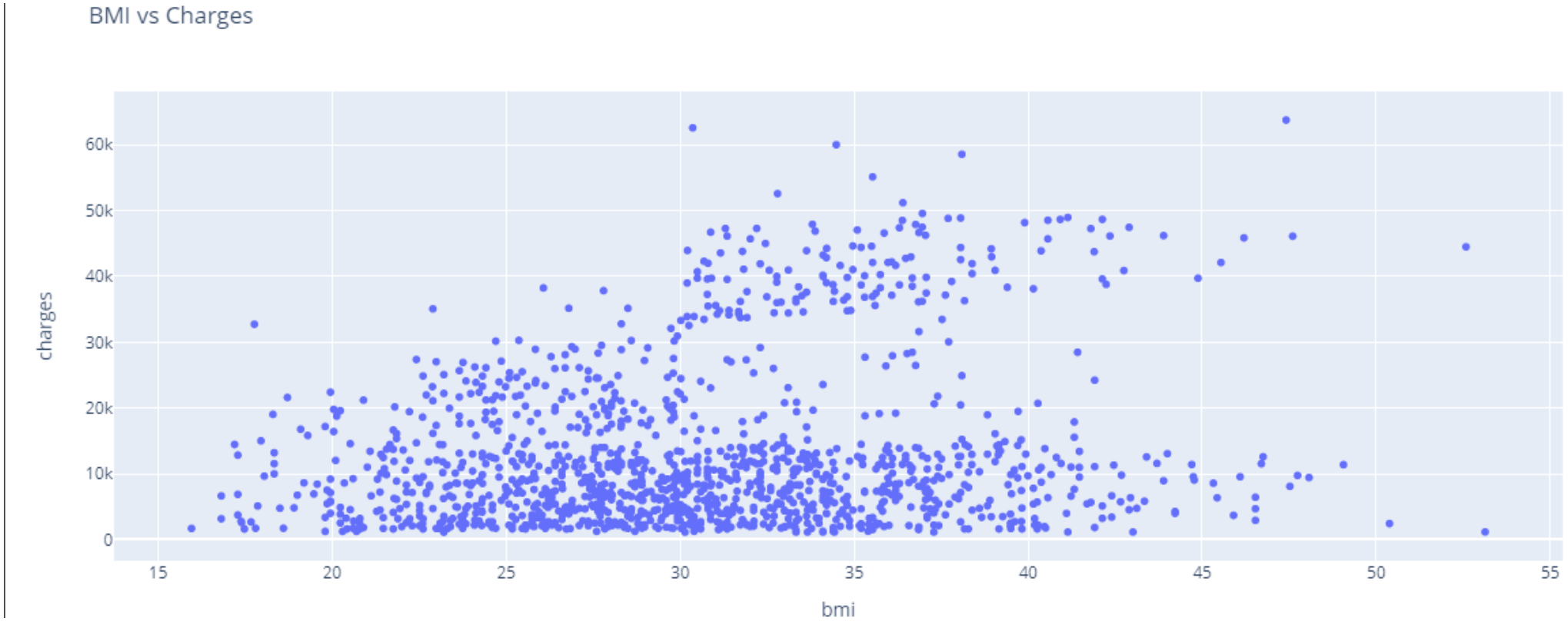


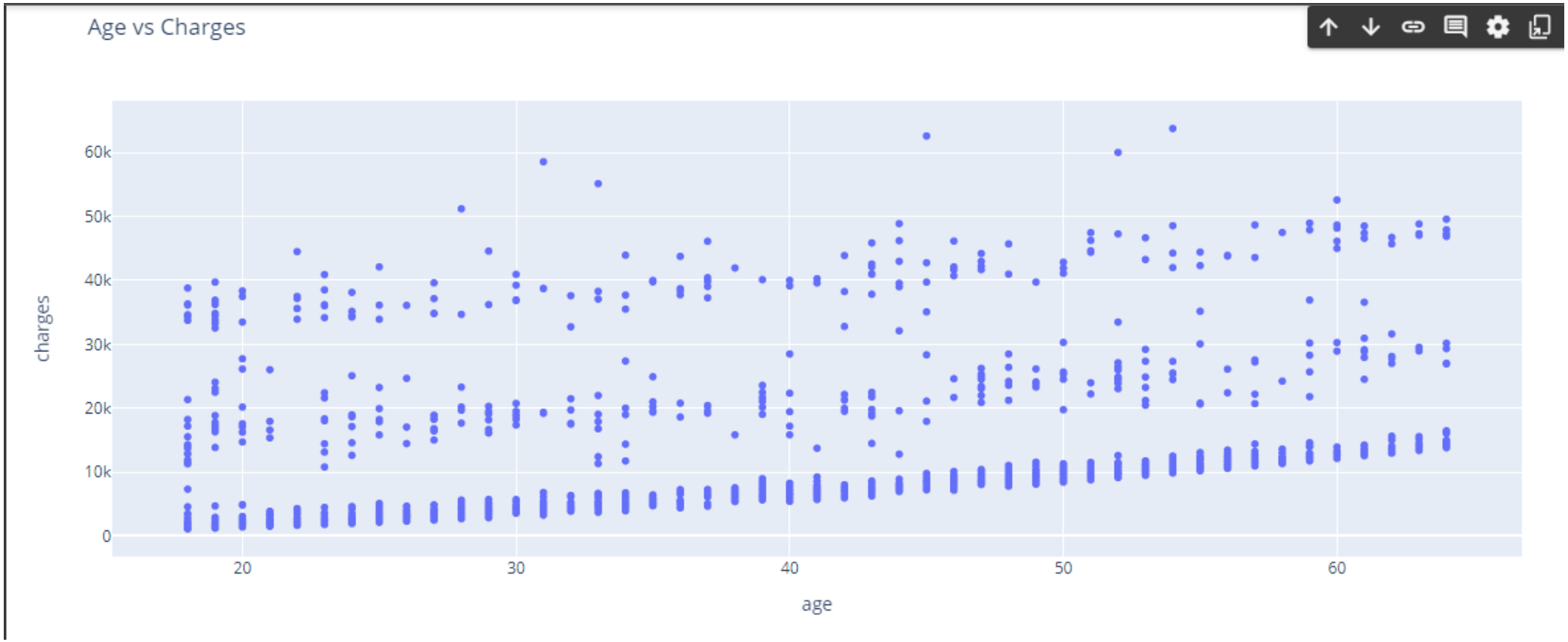


1. **BMI Distribution**:
    1. The first histogram represents the distribution of BMI values in the dataset.
    2. The x-axis shows the BMI value ranges, while the y-axis shows the count of individuals (frequency) within each BMI range.
    3. It seems to have a bell-shaped distribution, which suggests a normal distribution, indicating that most individuals have a BMI around the center of the range, with fewer individuals at the lower and higher ends.
    4. The distribution has a slight right skew, indicating that there are more individuals with a BMI slightly above the average.

2. **Children Distribution**:
    1. The second histogram shows how many children the individuals in the dataset have.
    2. The x-axis categorizes the number of children, and the y-axis shows the frequency of individuals with that number of children.
    3. The distribution suggests that most individuals have fewer children, with the majority having none or one child, and progressively fewer individuals have two, three, four, or five children.
    4. This could imply that, within this dataset, individuals without children or with only one child are more common, potentially reflecting demographic trends in the population sample.

BIVARIATE ANALYSIS




Age vs Charges
charges
60k
50k
40k
30k
20k
10k
0
20
30
40
50
60
age

1. **Age vs. Charges Scatter Plot:**
    1. This plot shows individual data points representing the relationship between age (on the x-axis) and insurance charges (on the y-axis).
    2. A general upward trend can be observed, suggesting that as age increases, the insurance charges also tend to increase. This reflects the common understanding that older individuals are often charged more for health insurance due to a higher risk of health issues.
    3. The spread of charges also seems to increase with age, indicated by the vertical dispersion of points becoming broader as age increases. This might suggest that factors other than age begin to play a more significant role in the variability of charges for older age groups.
2. **BMI vs. Charges Scatter Plot:**
    1. This plot displays the relationship between BMI (on the x-axis) and insurance charges (on the y-axis).
    2. The distribution is more scattered compared to the Age vs. Charges plot, indicating a less clear relationship between BMI and insurance charges.
    3. While there is a slight upward trend suggesting that higher BMI may be associated with higher insurance charges, the correlation seems weaker than that with age.
    4. There are several outliers, especially at higher BMI levels, which could indicate cases where a high BMI leads to significantly higher insurance costs, possibly due to associated health risks.

CORRELATION MAP

- The smoker variable has a strong positive correlation with charges (approximately 0.79), suggesting that smokers tend to have higher medical charges.
- The age variable has a moderate positive correlation with charges (approximately 0.30), indicating that as age increases, medical charges tend to be higher.
- Other variables have much weaker correlations with charges.
- In this case, smoker and age might be especially useful features for predicting medical insurance costs

ONE HOT ENCODING

```
# Create pipelines for both numeric and categorical preprocessing
numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(drop='first'))  # drop='first' is used to avoid collinearity
])
```

When the OneHotEncoder is applied, each categorical feature is transformed into multiple binary features, each representing a unique category value for that feature. The drop='first' parameter is used to drop the first level of each categorical feature to avoid the dummy variable trap, which can cause multicollinearity issues in certain models (though this is less of an issue with models like neural networks). After encoding, for instance, the sex feature with categories female and male will be transformed into a single binary feature (since one of the binary features is dropped), which will be 1 for male and 0 for female (or the reverse, depending on which category is encoded as 1).

TRANSFORMING TO NUMPY ARRAYS

```
# Combine preprocessing steps into one column transformer
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```
[25] # Split your data into features and target variable
X = insurance_data.drop('charges', axis=1)
y = insurance_data['charges']

# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
[26] # Apply the ColumnTransformer to the feature set
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)
```

X_train_preprocessed and X_test_preprocessed are now numpy arrays ready to be used in Keras model.

SECTION A

```
The training data has 8 features.
Model: "Model_seq_Insurance_arch1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 Model1_HiddenLayer_0 (Dens  (None, 64)                576
 e)

 Model1_HiddenLayer_1 (Dens  (None, 32)                2080
 e)

 Model1_OutputLayer (Dense)  (None, 1)                 33

=================================================================
Total params: 2689 (10.50 KB)
Trainable params: 2689 (10.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

1.First Hidden Layer (Model1_HiddenLayer_0): This is the first hidden layer in the network. It contains 64 neurons (or units). Each neuron in this layer will receive input from all the neurons in the input layer. The activation function for this layer is not specified in the provided information, but typically ReLU (Rectified Linear Unit) is used in such architectures.

2.Second Hidden Layer (Model1_HiddenLayer_1): This is the second hidden layer in the network and contains 32 neurons. As with the first hidden layer, each neuron here receives input from all the neurons in the previous layer.

3.Output Layer (Model1_OutputLayer): The output layer has a single neuron since this seems to be a regression model (evident from the continuous nature of the output, which is likely to be the insurance cost). The activation function for this output neuron is typically linear (or no activation function), which is suitable for regression tasks.

MATH ON THE PARAMETERS

- **Number of Parameters = (Number of Neurons in Previous Layer * Number of Neurons in Current Layer) + Number of Neurons in Current Layer (for biases)**

1. **Input Layer to First Hidden Layer**:
    1. Each feature connects to each neuron in the first hidden layer.
    2. Parameters = Number of input features * Number of neurons in the first hidden layer + Number of biases in the first hidden layer.
    3. Parameters = 8 * 64 + 64.

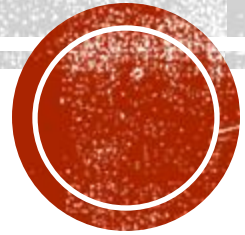
2. **First Hidden Layer to Second Hidden Layer**:
    1. Each neuron in the first hidden layer connects to each neuron in the second hidden layer.
    2. Parameters = Number of neurons in the first hidden layer * Number of neurons in the second hidden layer + Number of biases in the second hidden layer.
    3. Parameters = 64 * 32 + 32.

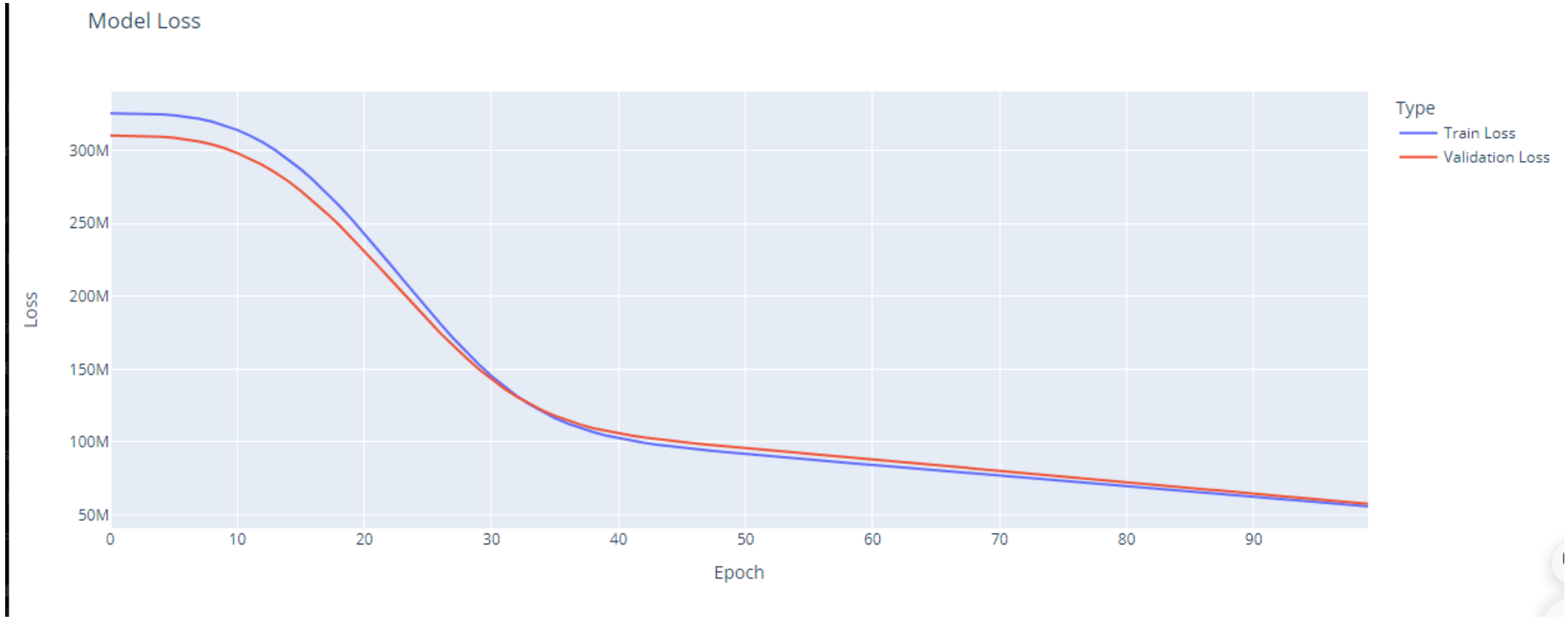
3. **Second Hidden Layer to Output Layer**:
    1. Each neuron in the second hidden layer connects to the single neuron in the output layer.
    2. Parameters = Number of neurons in the second hidden layer * Number of neurons in the output layer + Number of biases in the output layer.
    3. Parameters = 32 * 1 + 1.

- Adding these up:
- $Total\ Parameters$=(8×64+64)+(64×32+32)+(32×1+1)
- 2689 =(8×64+64)+(64×32+32)+(32×1+1)

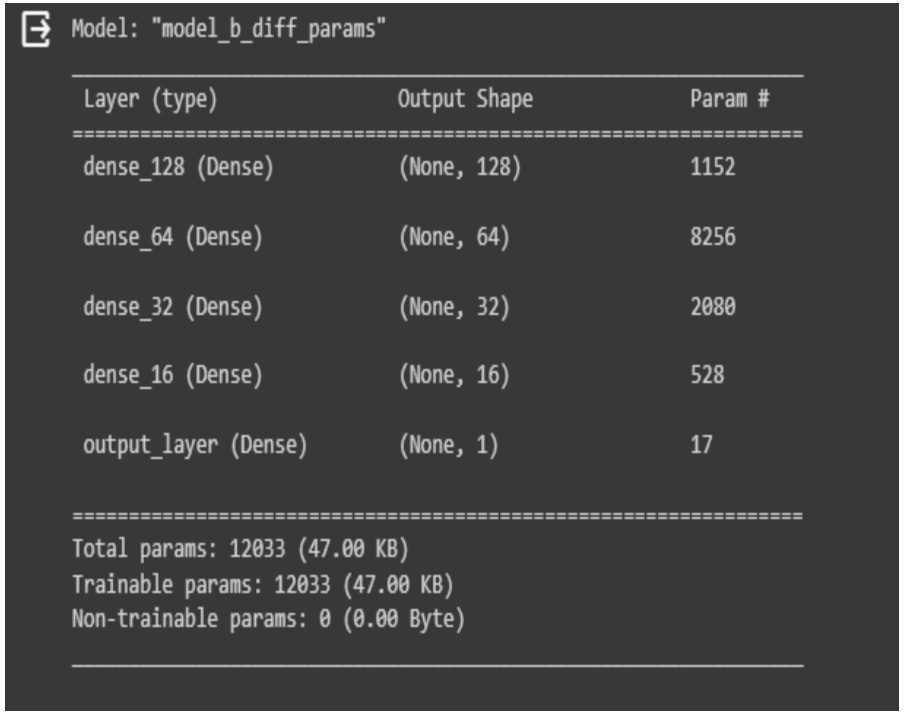
R2 SCORE OF MODEL-1

```
9/9 [==============================] - 0s 2ms/step
R2 score: 0.6393441670448308
```




Model Loss
Loss
300M
250M
200M
150M
100M
50M
0
10
20
30
40
50
60
70
80
90
Epoch
Type
Train Loss
Validation Loss

SECTION-B



Model: "model_b_diff_params"

Layer (type)	Output Shape	Param #
dense_128 (Dense)	(None, 128)	1152
dense_64 (Dense)	(None, 64)	8256
dense_32 (Dense)	(None, 32)	2080
dense_16 (Dense)	(None, 16)	528
output_layer (Dense)	(None, 1)	17

Total params: 12033 (47.00 KB)
Trainable params: 12033 (47.00 KB)
Non-trainable params: 0 (0.00 Byte)

1.First Hidden Layer (dense_128):

- This layer has 128 neurons.
- If the input layer size is **n_input**, the parameters for this layer would be calculated as **n_input * 128 + 128** (for weights and biases).

2.Second Hidden Layer (dense_64):

- This layer has 64 neurons.
- The parameters for this layer would be **128 * 64 + 64**.

3.Third Hidden Layer (dense_32):

- This layer has 32 neurons.
- The parameters for this layer would be **64 * 32 + 32**.

4.Fourth Hidden Layer (dense_16):

- This layer has 16 neurons.
- The parameters for this layer would be **32 * 16 + 16**.

5.Output Layer (output_layer):

- The output layer has a single neuron, which is typical for a regression task.
- The parameters for this layer would be **16 * 1 + 1**.

Adding all these parameters together gives us the total number of parameters for the model:

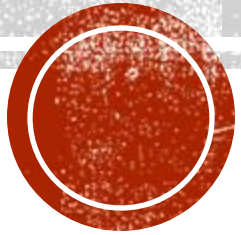
Total Parameters=(128+64)+(128∗64+64)+(64∗32+32)+(32∗16+16)+(16∗1+1)

Total Parameters=1152+8256+2080+528+17Total Parameters=1152+8256+2080+528+17

Total Parameters=12033Total Parameters=12033

R2 SCORE OF MODEL-2

```
9/9 [==============================] - 0s 2ms/step
R2 score for model B with different parameters: 0.8772264124931615
```



SECTION-C

```
from kerastuner.tuners import RandomSearch
import keras_tuner as kt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.optimizers import Adam
import keras_tuner as kt
def build_model(hp):
    model = Sequential()
    model.add(Dense(hp.Int('input_units', min_value=32, max_value=256, step=32),
                    activation='relu', input_shape=(n_features,)))
    for i in range(hp.Int('n_layers', 1, 3)):
        model.add(Dense(hp.Int(f'dense_{i}_units', min_value=32, max_value=256, step=32), activation='relu'))
    model.add(Dense(1, activation='linear'))
    model.compile(
        optimizer=Adam(hp.Choice('learning_rate', values=[1e-2, 1e-3, 1e-4])),
        loss='mse',
        metrics=['mae']
    )
    return model
```

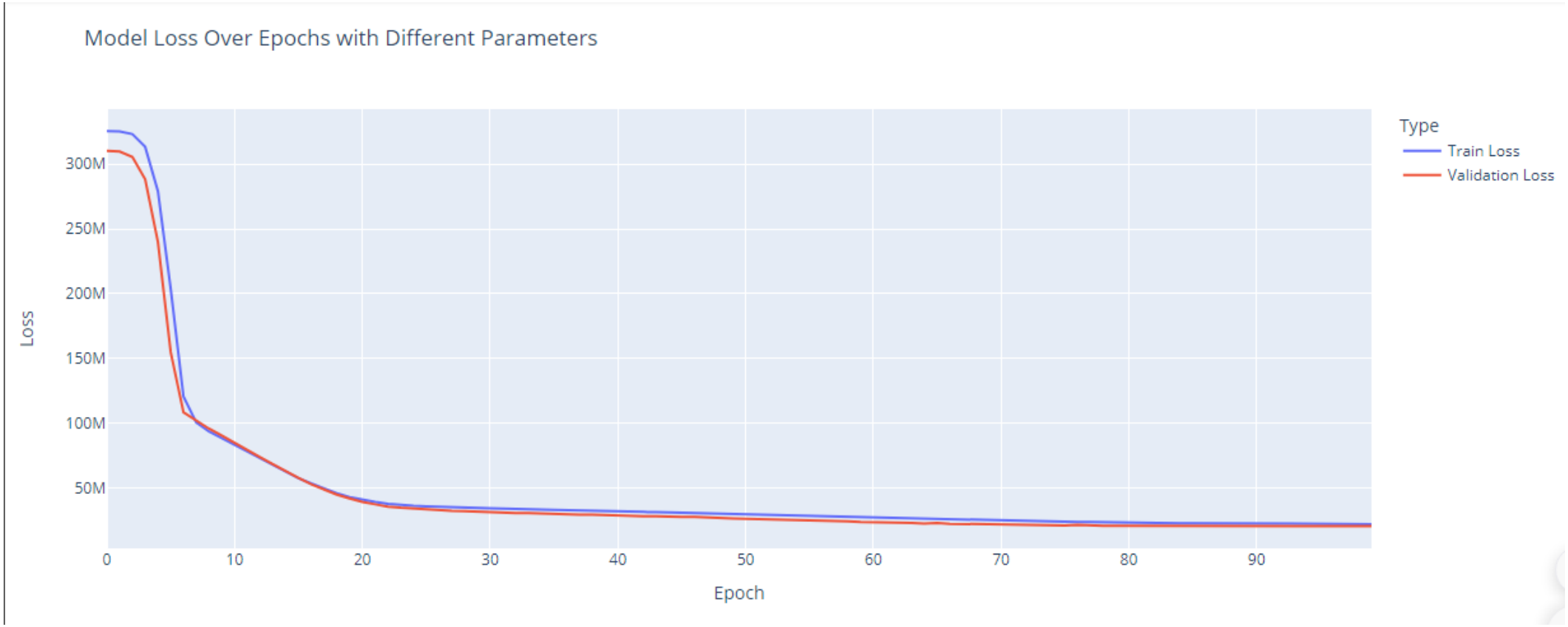
```
[56]
    tuner = kt.RandomSearch(
        build_model,
        objective='val_loss',
        max_trials=5,
        executions_per_trial=3,
        directory='keras_tuner_dir',
        project_name='insurance_cost_prediction'
    )

    Reloading Tuner from keras_tuner_dir/insurance_cost_prediction/tuner0.json

[57] tuner.search(X_train_preprocessed, y_train, epochs=10, validation_split=0.2)
```

```
best_hps = tuner.get_best_hyperparameters(num_trials=1)[0]
print(best_hps.values)
```

```
{'input_units': 192, 'n_layers': 3, 'dense_0_units': 64, 'learning_rate': 0.01, 'dense_1_units': 128, 'dense_2_units': 160}
```

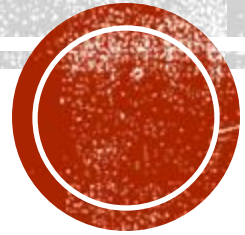
```
model_c = tuner.hypermodel.build(best_hps)
history_c = model.fit(
    X_train_preprocessed, y_train,
    epochs=100,
    validation_split=0.2
)
```

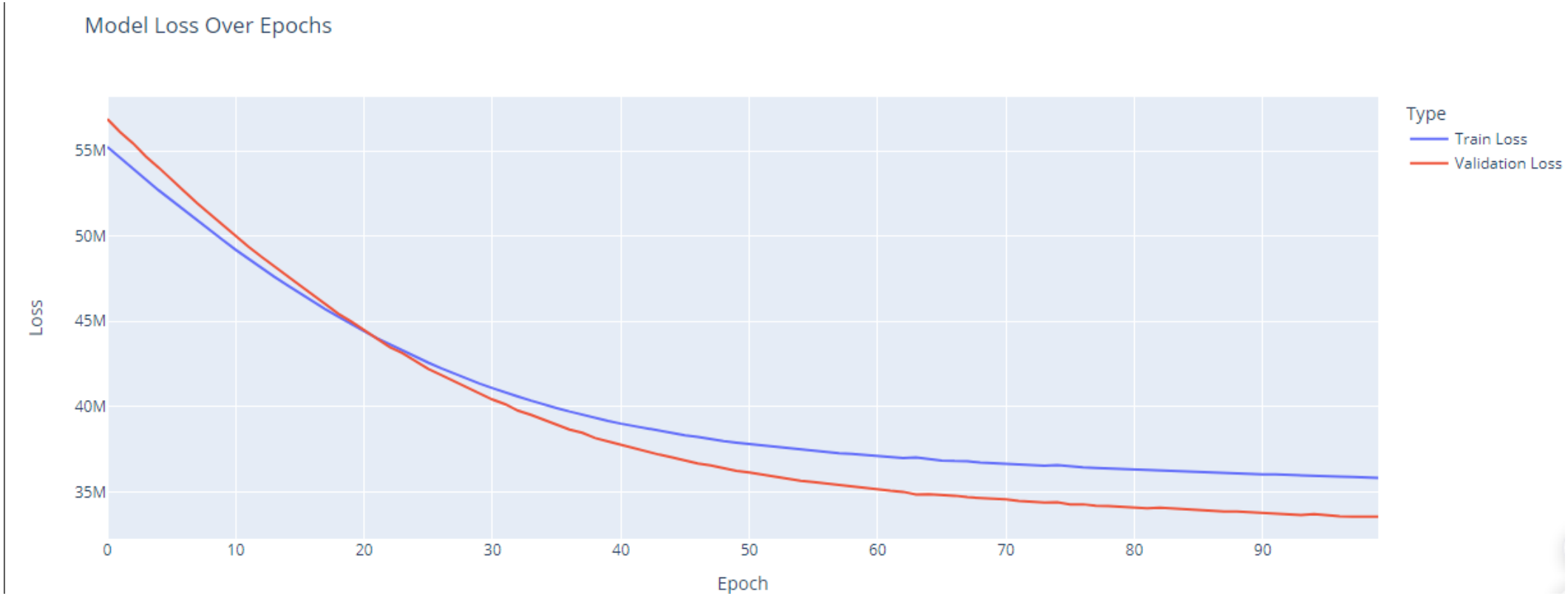
- Model Initialization: A new Sequential model is created, which is a linear stack of layers.
- Input Layer: The first Dense layer added to the model serves as the input layer. The number of units (neurons) in this layer is a hyperparameter that will be searched by the tuner within the range of 32 to 256, with a step size of 32.
- The activation function for this layer is 'relu' (rectified linear unit), and it expects input data with a shape that has a number of features equal to n_features.
- Hidden Layers: The code sets up a loop to add a variable number of hidden Dense layers, with the number of layers being a hyperparameter n_layers to be tuned from 1 to 3. Each of these layers also has a variable number of units, just like the input layer, and uses the 'relu' activation function.
- Output Layer: A single-unit Dense layer with a 'linear' activation function is added. This is typical for a regression problem where the output is a single continuous value.
- Model Compilation: The optimizer is Adam, a popular choice for many types of neural networks. The learning rate for Adam is also a hyperparameter to be tuned, with possible values of 1e-2, 1e-3, or 1e-4.
- The loss function is set to 'mse' (mean squared error), which is common for regression problems. The metric used to evaluate the model's performance during training and validation is 'mae' (mean absolute error).


Model Loss Over Epochs with Different Parameters
Loss
300M
250M
200M
150M
100M
50M
0
10
20
30
40
50
60
70
80
90
Epoch
Type
Train Loss
Validation Loss

MODEL-3 R2 SCORE

```
9/9 [==============================] - 0s 5ms/step
R2 score for best model: 0.79333303538546 51
```

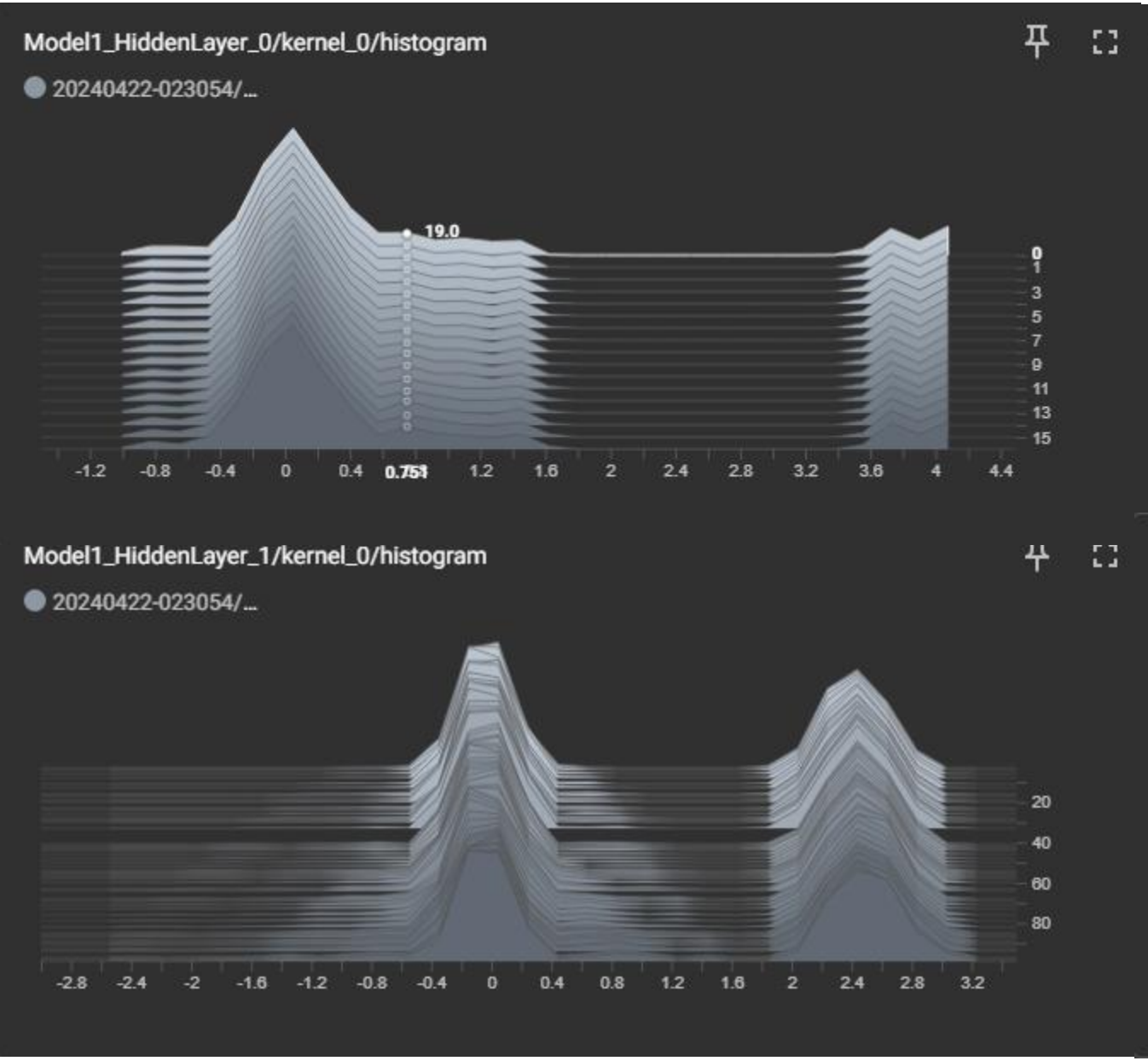


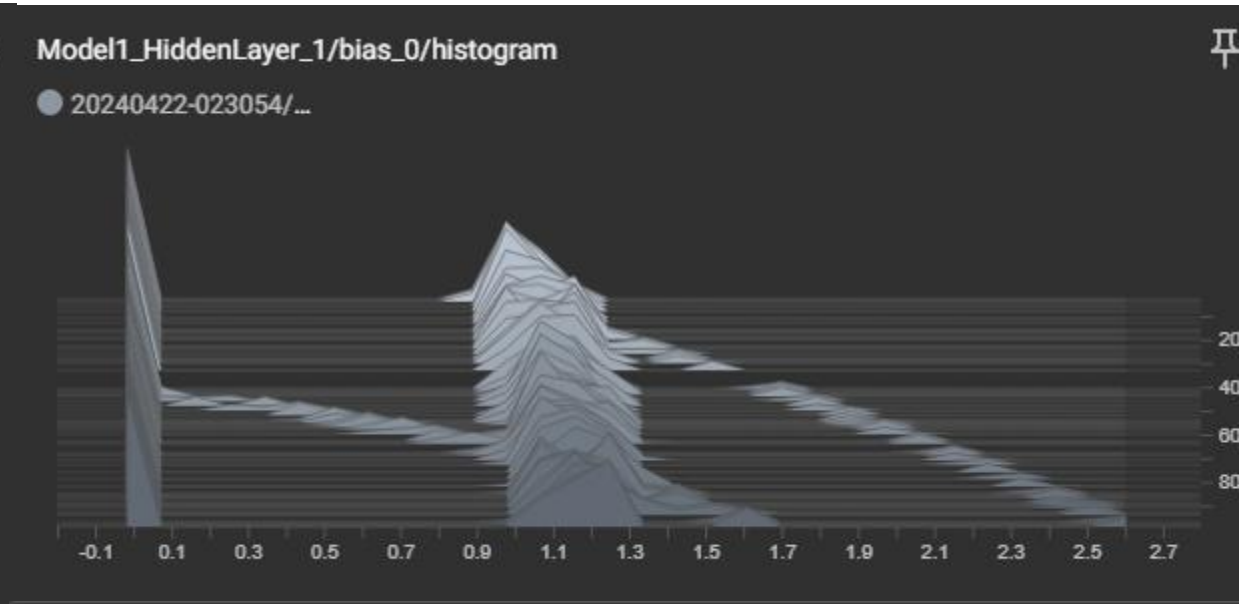

Model Loss Over Epochs
Loss
55M
50M
45M
40M
35M
0
10
20
30
40
50
60
70
80
90
Epoch
Type
Train Loss
Validation Loss

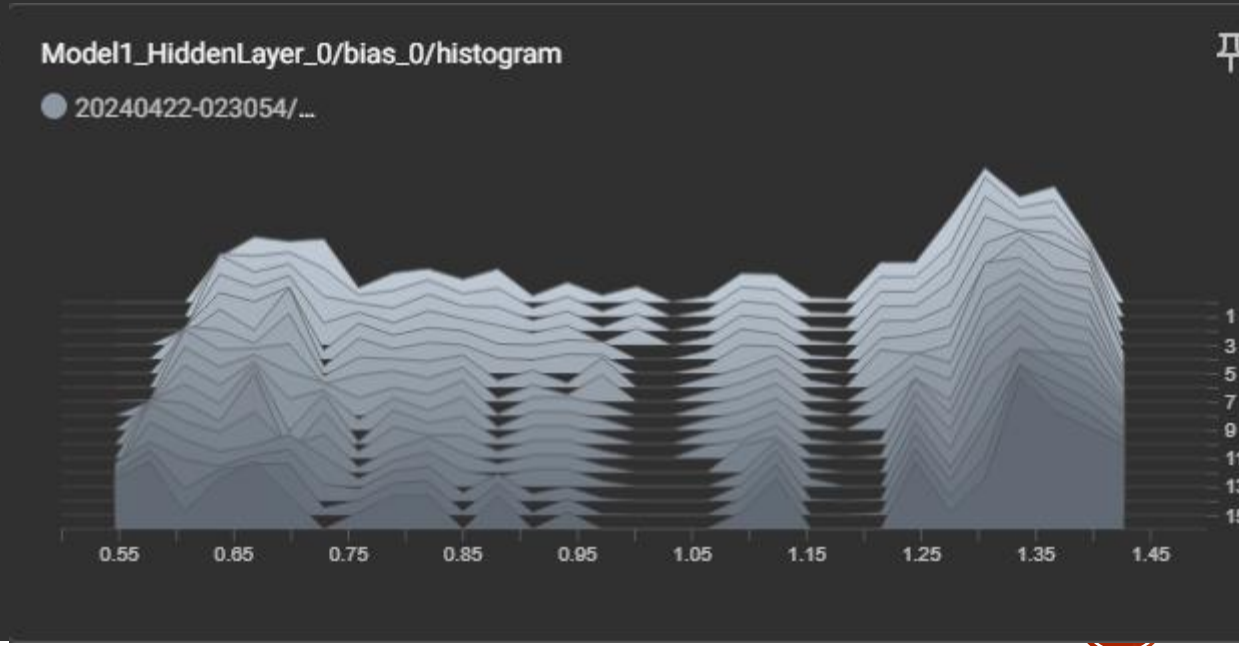
INVOKING TENSOR BOARD CALLBACK AND EMBEDDING TENSORBOARD

```
# Create a TensorBoard callback instance for new training runs
tensorboard_callback = TensorBoard(log_dir=new_logdir, histogram_freq=1)

# Fit the model with the TensorBoard callback for new training runs
history_c = model.fit(
    X_train_preprocessed, y_train,
    epochs=100,
    validation_split=0.2,
    callbacks=[early_stopper, tensorboard_callback]
)
```

Tensor Boards- Hidden layers

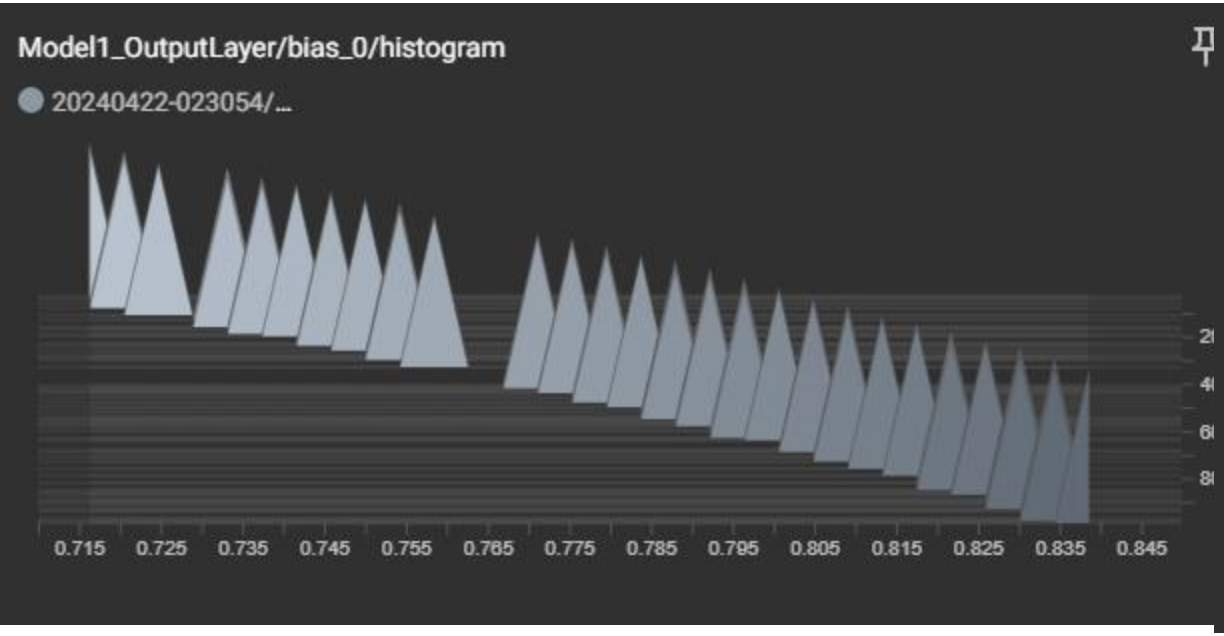


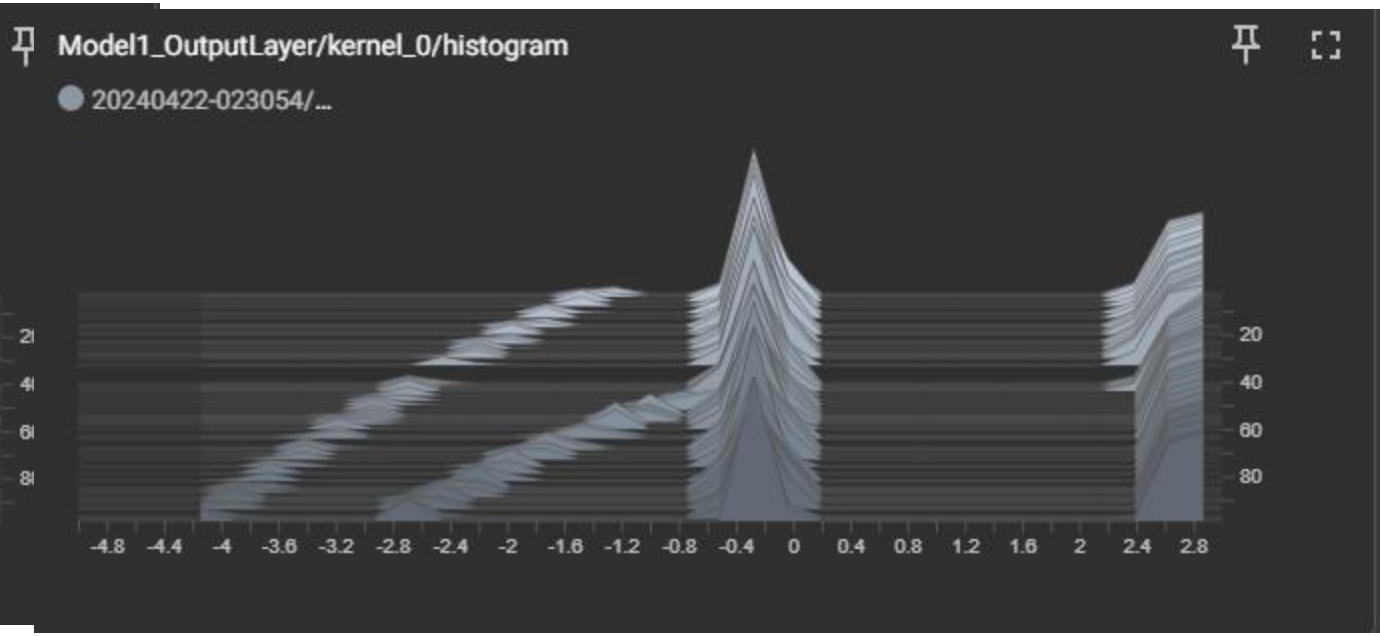




1. **First Hidden Layer Bias (Model1_HiddenLayer_0/bias_0)**:
    1. The distribution of bias values in the first hidden layer seems to be concentrated around values greater than 1, with the peak around 1.25.
    2. Over time, as the epochs progress (from back to front in the 3D plot), the bias values shift slightly, indicating that the learning process is adjusting these biases.
2. **First Hidden Layer Weights (Model1_HiddenLayer_0/kernel_0)**:
    1. The weight values in the first hidden layer show a more diverse distribution, with multiple peaks and a range from about -1 to 4.5.
    2. This wider distribution suggests that different neurons are learning to detect various features by adjusting their weights differently.
    3. The changes over epochs suggest ongoing learning and adjustments as the model attempts to minimize the loss function.
3. **Second Hidden Layer Bias (Model1_HiddenLayer_1/bias_0)**:
    1. The second layer's bias values are mostly concentrated around 0.5 to 1.5, with a clear peak around 1.
    2. This concentration indicates a common adjustment for the neurons in this layer, leaning towards positive bias values which influence the activation coming into this layer.
4. **Second Hidden Layer Weights (Model1_HiddenLayer_1/kernel_0)**:
    1. The weight distribution for the second hidden layer displays a bimodal pattern, suggesting two main groupings of weight values that the neurons are adjusting around.
    2. This layer is likely refining the features detected by the first layer, as indicated by the bimodal distribution which could be representing different kinds of feature interactions.

TENSOR BOARDS-OUTPUT LAYER





1. **Output Layer Bias (Model1_OutputLayer/bias_0)**:
    1. This histogram shows the distribution of bias values in the output layer.
    2. The values are fairly stable and clustered around 0.8, which may suggest a slight positive adjustment that the neuron in the output layer is consistently applying to its inputs.
    3. Over the course of the epochs (represented by the layers of the histogram, with the frontmost layer being the latest epoch), there appears to be little variation, indicating that the bias has reached a relatively stable value early in the training process.
2. **Output Layer Weights (Model1_OutputLayer/kernel_0)**:
    1. This histogram shows the distribution of weights connecting the last hidden layer to the output neuron.
    2. The distribution of weights appears to be approximately symmetric around zero and has multiple peaks, suggesting a complex relationship where some inputs are weighted positively and others negatively, reflecting the diversity in how different features affect the output.
    3. Over time, the distribution remains centered but spreads slightly wider, indicating that the model is fine-tuning the influence of the input features on the prediction.

THANK YOU ☺