**Exp 5**

**CREATE A LINEAR REGRESSION MODEL USING KERAS API**

**AIM:**

To create a linear regression model using the Keras API for predicting a target variable based

on a single input feature.

**INTEGRATED DEVELOPMENT ENVIRONMENT (IDE) REQUIRED:**

JUPYTER NOTEBOOK

**REQUIRED LIBRARIES FOR PYTHON:**

• Tensorflow

• Numpy

• Matplotlib

**PROCEDURE:**

• **Step 1:** Import necessary libraries including NumPy for numerical operations,

TensorFlow for building and training the model, and Matplotlib for visualization.

• **Step 2:** Generate random data for demonstration, where **X** is the input feature and **y** is

the target variable with some random noise.

• **Step 3:** Build a sequential model (**model**) using Keras and add a single Dense layer

with a linear activation function.

• **Step 4:** Compile the model using stochastic gradient descent (**'sgd'**) as the optimizer

and mean squared error (**'mean\_squared\_error'**) as the loss function.

• **Step 5:** Train the model using the generated data (**X** and **y**) for 50 epochs with a batch

size of 1.

• **Step 6:** Make predictions using the trained model on the input data.

• **Step 7:** Plot the training data points and the regression line to visualize the model's

performance.

• **Step 8:** Print the weights and bias learned by the model during training.

• **Step 9:** Make predictions on new input data.

• **Step 10:** Display the input values and corresponding predicted outputs.

**PROGRAM:**

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

# Generate random data

np.random.seed(42)

tf.random.set\_seed(42)

# Generate random data

y = 4 + 3 \* X + np.random.randn(100, 1)

# Create a linear model model

= tf.keras.Sequential ([

tf.keras.layers.Input(shape=(1,), name='input\_layer'),

tf.keras.layers.Dense(1, name='output\_layer') ])

# Compile the model

model.compile(optimizer='sgd', loss='mse') # sgd stands for Stochastic Gradient Descent

# Display the model summary

model.summary()

# Train the model

history = model.fit(X, y, epochs=100, verbose=0)

# Plot the training loss

plt.plot(history.history['loss'])

plt.title('Model Training Loss')

plt.xlabel('Epoch')

plt.ylabel('Mean Squared

Error') plt.show()

# Get the learned weight and bias

weight, bias = model.layers[0].get\_weights()

print("Weight:", weight[0, 0])

print("Bias:", bias[0])# Make predictions

X\_new = np.array([[0], [2]])

predictions = model.predict(X\_new)

# Display predictions

for i in range(len(X\_new)):

print(f"Input: {X\_new[i][0]}, Predicted Output: {predictions[i][0]}")

**OUTPUT:**

