Experiment-1: Working with TensorFlow , NumPy, Kera, Pandas, Matplotlib, etc.

Importing Nessary Library and Module

```
#@title Importing Nessary Library and Module
import tensorflow as tf
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

np.random.rand(100, 1).flatten(): This generates an array of shape (100, 1) with random numbers between 0 and 1 using NumPy's rand function. The flatten() method is then used to convert this 2D array into a 1D array.

Generate Random Data for Simple Linear Regression using Numpy

```
#@title Generate Random Data for Simple Linear Regression using Numpy
np.random.seed(42)

data = {'X': np.random.rand(100, 1).flatten()}
data['y']= 3 * data['X'] + 2 + 0.1 * np.random.randn(100)
```

Create Pandas DataFrame from Random Data using Pandas

```
#@title Create Pandas DataFrame from Random Data using Pandas
df = pd.DataFrame(data)
print(df.info())
```

Split Data into Training and Testing Sets using Sklearn

```
#@title Split Data into Training and Testing Sets using Sklearn
X_train, X_test, y_train, y_test = train_test_split(df['X'], df['y'], test_size=0.2, random_state=42)
```

Build and Display Summary of Neural Network with Keras

```
#@title Build and Display Summary of Neural Network with Keras
model = Sequential()

# Add a hidden layer with 5 units and ReLU activation
model.add(Dense(units=2, activation='linear'))

# Add the output layer with 1 unit and linear activation
model.add(Dense(units=1, activation='relu'))

# Build the model
model.build(input_shape=(None, 1))

# Compile the model
model.compile(optimizer='sgd', loss='mean_squared_error')

# Display the model summary
model.summary()
```

Explanation of the code:

model = Sequential() : This line creates a Sequential model using Keras. The Sequential model is a linear stack of layers, and you can add layers to it in a step-by-step fashion.

model.add(Dense(units=2, activation='linear')) : This line adds a dense (fully connected) hidden layer to the model. The layer has 6 units (neurons) and uses the ReLU (Rectified Linear Unit) activation function. The Dense layer connects every input neuron to every output neuron.

model.add(Dense(units=1, activation='relu')) : This line adds another dense layer to the model, which serves as the output layer. It has 1 unit and uses a linear activation function. For regression tasks, a linear activation is often used for the output layer.

model.build(input_shape=(None, 1)) : This line builds the model, specifying the input shape. The input_shape is set to (None, 1), indicating that the model can accept input data with any batch size (None) and one feature (1).

model.compile(optimizer='sgd', loss='mean_squared_error') : This line compiles the model. It specifies the optimizer (Stochastic Gradient Descent - 'sgd') and the loss function (Mean Squared Error - 'mean_squared_error'). Compiling the model is a necessary step before training.

history = model.fit(X_train, y_train, epochs=100, verbose=0) : This line trains the model using the training data (X_train and y_train). It specifies the number of training epochs (100) and sets verbose=0 to suppress training progress updates.

model.summary() : This line prints a summary of the model's architecture, including the type of layers, the number of parameters, and the output shapes. The summary provides a concise overview of the neural network.

Train, Predict, and Evaluate Neural Network

```
#@title Train, Predict, and Evaluate Neural Network
# Train the model
history = model.fit(X_train, y_train, epochs=100, verbose=0)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error on Test Set: {mse:.4f}')
```

Visualize Results of Simple Linear Regression with Neural Network

```
#@title Visualize Results of Simple Linear Regression with Neural Network
plt.scatter(X_test, y_test, color='black', label='Actual')
plt.plot(X_test, y_pred, color='blue', linewidth=3, label='Predicted')
plt.xlabel('X')
plt.ylabel('y')
plt.title('Scatter Plot of Actual vs Predicted Values (Simple Linear Regression)')
plt.legend()
plt.show()
```