```
# Script download and cleaning function (limit to 100 characters)
import urllib.request
def dt(url, limit=20000):
    """Downloads script from a URL, cleans it, and limits to a specified number of
characters."""
    response = urllib.request.urlopen(url)
    script = response.read().decode('utf-8')[:limit]  # Limit to the first 'limit'
characters
    script = script.lower()
    import re
    script = re.sub(r'[^a-zA-Z\s.,!?;\'\"]', '', script)  # Remove non-alphabetical
characters
    return script

# Download Text
links = [
    'https://www.gutenberg.org/files/26184/26184-0.txt',
    'https://www.gutenberg.org/files/84/84-0.txt',
    'https://www.gutenberg.org/files/2701/2701-0.txt'
]

scripts = []
j = 0

while j < len(links):
    scripts.append(dt(links[j]))
    j += 1

script = ' '.join(scripts)  # Scripts join, but now it will be limited to the first
100 characters

# Character frequency analysis
eccentric = sorted(set(script))
symbol_conv = {}
conv = 0

# Calculate character frequency
char_freq = {char: script.count(char) for char in eccentric}

# Bar chart for character frequency
import matplotlib.pyplot as plt
import seaborn as sns
plt.figure(figsize=(12, 6))
sns.barplot(x=list(char_freq.keys()), y=list(char_freq.values()))
plt.title('Character Frequency')
plt.xticks(rotation=45)
plt.show()

# Scatter plot for character distribution
x_vals = [i for i, char in enumerate(script)]
y_vals = [char_freq.get(char, 0) for char in script]

plt.figure(figsize=(12, 6))
plt.scatter(x_vals, y_vals, alpha=0.5)
plt.title('Character Distribution - Scatter Plot')
plt.xlabel('Character Position in Script')
plt.ylabel('Character Frequency')
plt.show()
```

```
# Histogram for character frequency distribution
plt.figure(figsize=(12, 6))
plt.hist(list(char_freq.values()), bins=30, color='blue', edgecolor='black')
plt.title('Character Frequency Distribution (Histogram)')
plt.xlabel('Character Frequency')
plt.ylabel('Count of Characters')
plt.show()

# Symbol conversion mapping
import numpy as np
while conv < len(eccentric):
    symbol_conv[eccentric[conv]] = conv
    conv += 1
conv_symbol = np.array(eccentric)

queue_segment = 100

# Convert script to numeric
script_numeric = np.zeros(len(script), dtype=np.int32)
i = 0

while i < len(script):
    script_numeric[i] = symbol_conv[script[i]]
    i += 1
conv_symbol = np.array(eccentric)

# Training Data Generation
input_chron = []
intent_chron = []

l = 0
while l < len(script_numeric) - queue_segment:
    input_chron.append(script_numeric[l:l + queue_segment])
    intent_chron.append(script_numeric[l + 1:l + queue_segment + 1])
    l += 1
input_chron = np.array(input_chron)
intent_chron = np.array(intent_chron)

# Neural Network Model
import tensorflow as tf
class scriptModel(tf.keras.Model):
    def __init__(idn, term_margin):
        super().__init__()
        idn.embedding = tf.keras.layers.Embedding(term_margin, 128)
        idn.gru = tf.keras.layers.GRU(512, return_sequences=True)
        idn.dense = tf.keras.layers.Dense(term_margin)

    def call(idn, inputs):
        x = idn.embedding(inputs)
        x = idn.gru(x)
        return idn.dense(x)

# Train the prototype
term_margin = len(eccentric)
prot = scriptModel(term_margin)

# Compilation
prot.compile(optimizer=tf.keras.optimizers.Adam(),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
```

```
# Create validation data by splitting the dataset
val_input_chron = input_chron[int(len(input_chron) * 0.8):]  # 20% for validation
val_intent_chron = intent_chron[int(len(intent_chron) * 0.8):]

# Train the prototype with validation data
history = prot.fit(input_chron[:int(len(input_chron) * 0.8)],  # 80% for training
                   intent_chron[:int(len(intent_chron) * 0.8)],
                   epochs=10,  # Increase epochs for better training
                   batch_size=64,
                   validation_data=(val_input_chron, val_intent_chron))

# Plot the training and validation loss
plt.plot(history.history['loss'], label='Training Loss', color='purple')
plt.plot(history.history['val_loss'], label='Validation Loss', color='black')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Script initiation function for text generation
def script_generation(prot, initiate_scr, num_chars=100, temp=0.5):
    """The trained prototype generates string."""

    # Filter the input to ensure it contains only valid characters present in
'symbol_conv'
    valid_chars = [c for c in initiate_scr if c in symbol_conv]  # Only keep valid
characters
    if not valid_chars:
        raise ValueError("Input string contains no valid characters present in the
training data.")

    initiate_scr = ''.join(valid_chars)  # Rebuild input string using only valid
characters
    input_analysis = tf.expand_dims([symbol_conv[c] for c in initiate_scr], 0)
    script_generated = []

    count = 0
    while count < num_chars:
        assum = prot(input_analysis)[:, -1, :] / temp
        followup_char_conv = tf.random.categorical(assum, num_samples=1)[-1,
0].numpy()  # Corrected here

        count += 1
        val = 0 <= followup_char_conv < len(conv_symbol)

        script_generated.append(conv_symbol[followup_char_conv]) if val else None

        input_analysis = tf.expand_dims([followup_char_conv], 0)

    return initiate_scr + ''.join(script_generated)

# Test the model by generating text
initial_text = "The quick brown fox"
try:
    generated_text = script_generation(prot, initial_text, num_chars=100, temp=0.5)
    print("Generated Text:\n", generated_text)
except ValueError as e:
```

```
print(e)
```