```
# IMPORTANT: RUN THIS CELL IN ORDER TO IMPORT YOUR KAGGLE DATA SOURCES
# TO THE CORRECT LOCATION (/kaggle/input) IN YOUR NOTEBOOK,
# THEN FEEL FREE TO DELETE THIS CELL.
# NOTE: THIS NOTEBOOK ENVIRONMENT DIFFERS FROM KAGGLE'S PYTHON
# ENVIRONMENT SO THERE MAY BE MISSING LIBRARIES USED BY YOUR
# NOTEBOOK.

import os
import sys
from tempfile import NamedTemporaryFile
from urllib.request import urlopen
from urllib.parse import unquote, urlparse
from urllib.error import HTTPError
from zipfile import ZipFile
import tarfile
import shutil

CHUNK_SIZE = 40960
DATA_SOURCE_MAPPING = 'spam-text-message-classification:https%3A%2F%2Fstorage.googleapis.com%2Fkaggle-data-sets%2F2050%2F3494%2Fbundle%2Far

KAGGLE_INPUT_PATH='/kaggle/input'
KAGGLE_WORKING_PATH='/kaggle/working'
KAGGLE_SYMLINK='kaggle'

!umount /kaggle/input/ 2> /dev/null
shutil.rmtree('/kaggle/input', ignore_errors=True)
os.makedirs(KAGGLE_INPUT_PATH, 0o777, exist_ok=True)
os.makedirs(KAGGLE_WORKING_PATH, 0o777, exist_ok=True)

try:
  os.symlink(KAGGLE_INPUT_PATH, os.path.join("..", 'input'), target_is_directory=True)
except FileExistsError:
  pass
try:
  os.symlink(KAGGLE_WORKING_PATH, os.path.join("..", 'working'), target_is_directory=True)
except FileExistsError:
  pass

for data_source_mapping in DATA_SOURCE_MAPPING.split(','):
    directory, download_url_encoded = data_source_mapping.split(':')
    download_url = unquote(download_url_encoded)
    filename = urlparse(download_url).path
    destination_path = os.path.join(KAGGLE_INPUT_PATH, directory)
    try:
        with urlopen(download_url) as fileres, NamedTemporaryFile() as tfile:
            total_length = fileres.headers['content-length']
            print(f'Downloading {directory}, {total_length} bytes compressed')
            dl = 0
            data = fileres.read(CHUNK_SIZE)
            while len(data) > 0:
                dl += len(data)
                tfile.write(data)
                done = int(50 * dl / int(total_length))
                sys.stdout.write(f"\r[{'=' * done}{' ' * (50-done)}] {dl} bytes downloaded")
                sys.stdout.flush()
                data = fileres.read(CHUNK_SIZE)
            if filename.endswith('.zip'):
              with ZipFile(tfile) as zfile:
                zfile.extractall(destination_path)
            else:
              with tarfile.open(tfile.name) as tarfile:
                tarfile.extractall(destination_path)
            print(f'\nDownloaded and uncompressed: {directory}')
    except HTTPError as e:
        print(f'Failed to load (likely expired) {download_url} to path {destination_path}')
        continue
    except OSError as e:
        print(f'Failed to load {download_url} to path {destination_path}')
        continue

print('Data source import complete.')
```

```
Failed to load (likely expired) https://storage.googleapis.com/kaggle-data-sets/2050/3494/bundle/archive.zip?X-Goog-Algorithm=GOOG4-RSA
Data source import complete.
```

```
# Common imports
import numpy as np
import tensorflow as tf
from tensorflow import keras

# Data processing and visualization imports
import string
import pandas as pd
import plotly.express as px
import tensorflow.data as tfd
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split

# Model building imports
from sklearn.utils import class_weight
from tensorflow.keras import callbacks
from tensorflow.keras import Model, layers
```

```
# Define hyperparameters
num_heads = 4
embed_dim = 256
ff_dim = 128
vocab_size = 10000
max_seq_len = 40

# Set constants
learning_rate = 1e-3
epochs = 100
batch_size = 32

# Define training callbacks
callbacks = [
    keras.callbacks.EarlyStopping(patience=3, restore_best_weights=True),
    keras.callbacks.ModelCheckpoint("SpamDetector.h5", save_best_only=True)
]
```

```
# Set up random seed for reproducibility
random_seed = 123
np.random.seed(random_seed)
tf.random.set_seed(random_seed)
```

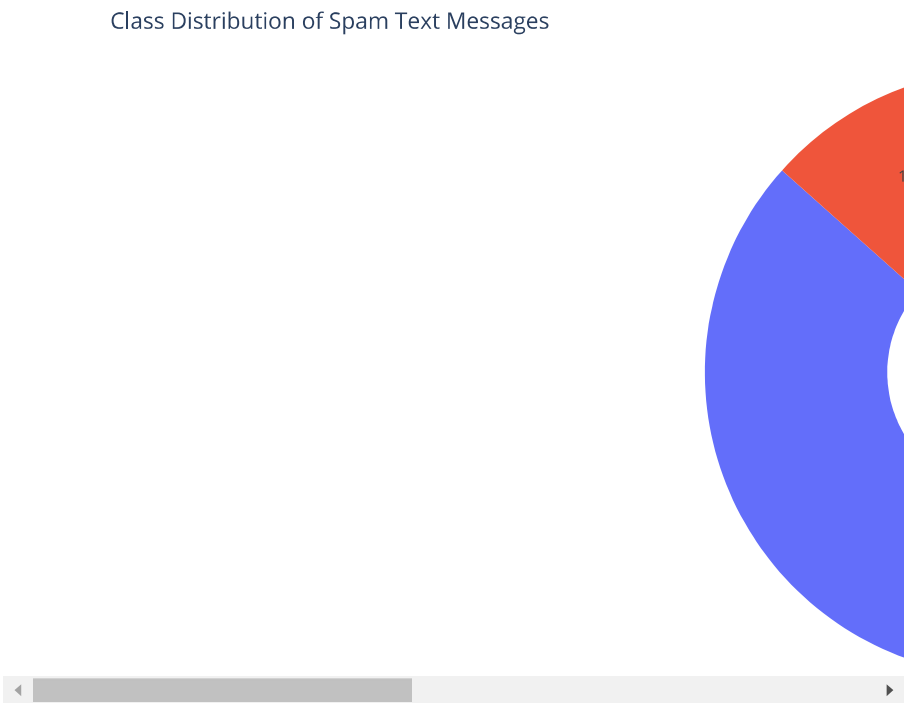
```
# Specify the path to the SPAM text message dataset
data_path = '/content/SPAM text message 20170820 - Data.csv'

# Load the dataset using the load_data function
data_frame = pd.read_csv(data_path)

# Print the first five rows of the dataset
data_frame.head()
```

	Category	Message
0	ham	Go until jurong point, crazy.. Available only ...
1	ham	Ok lar... Joking wif u oni...
2	spam	Free entry in 2 a wkly comp to win FA Cup fina...
3	ham	U dun say so early hor... U c already then say...
4	ham	Nah I don't think he goes to usf, he lives aro...

Next steps: Generate code with `data_frame` View recommended plots

Let's gather some deeper data informations.

```
# Get the counts of each class and their names
class_dis = data_frame.Category.value_counts()
class_names = class_dis.index

# Create the Pie Chart
fig = px.pie(names=class_names,
             values=class_dis,
             color=class_names,
             hole=0.4,
             labels={'value': 'Count', 'names': 'Class'},
             title='Class Distribution of Spam Text Messages')

# Customize the layout
fig.update_layout(
    margin=dict(l=10, r=10, t=60, b=10),
    legend=dict(orientation="h", yanchor="bottom", y=1.02, xanchor="right", x=1),
)

# Show the plot
fig.show()
```




```
# Data set size
N_SAMPLES = len(data_frame)

print(f"Total Number of Samples : {N_SAMPLES}")
```

```
Total Number of Samples : 5572
```

```
max_len = max([len(text) for text in data_frame.Message])
print(f"Maximum Length Of Input Sequence(Chars) : {max_len}")
```

```
Maximum Length Of Input Sequence(Chars) : 910
```

```
# Extract X and y from the data frame
X = data_frame['Message'].tolist()
y = data_frame['Category'].tolist()


# Initialize label encoder
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)

# Print the first 5 elements of X and y
print(f'X[:5]: \n{X[:5]}\n')
print(f'y[:5]: {y[:5]}\n')
print(f"Label Mapping : {label_encoder.inverse_transform(y[:5])}")
```

```
X[:5]:
['Go until jurong point, crazy.. Available only in bugis n great world la e buffet... Cine there got amore wat...', 'Ok lar... Joking w

y[:5]: [0 0 1 0 0]

Label Mapping : ['ham' 'ham' 'spam' 'ham' 'ham']
```

Text Vectorization

```
# Compute class weights
class_weights = class_weight.compute_class_weight(class_weight='balanced', classes=data_frame.Category.unique(), y=label_encoder.inverse_tr
class_weights = {number: weight for number, weight in enumerate(class_weights)}
# Show
print(f"Associated class weights: {class_weights}")
```

```
Associated class weights: {0: 0.5774093264248704, 1: 3.7295850066934406}
```

```
# Define a function to preprocess the text
def preprocess_text(text: str) -> str:
    """
    Preprocesses the text by removing punctuation, lowercasing, and stripping whitespace.
    """
    # Replace punctuation with spaces
    text = tf.strings.regex_replace(text, f"[{string.punctuation}]", " ")

    # Lowercase the text
    text = tf.strings.lower(text)

    # Strip leading/trailing whitespace
    text = tf.strings.strip(text)

    return text


# Create a TextVectorization layer
text_vectorizer = layers.TextVectorization(
    max_tokens=vocab_size,                      # Maximum vocabulary size
    output_sequence_length=max_seq_len,         # Maximum sequence length
    standardize=preprocess_text,                # Custom text preprocessing function
    pad_to_max_tokens=True,                     # Pad sequences to maximum length
    output_mode='int'                           # Output integer-encoded sequences
)

# Adapt the TextVectorization layer to the data
text_vectorizer.adapt(X)
```

Let's see the Text Vectorization working.

```
for _ in range(5):
    # Send a text to randomly.
    text_temp = X[np.random.randint(N_SAMPLES)]

    # Apply text to vectorization.
    text_vec_temp = text_vectorizer(text_temp)

    # Show the results
    print(f"Original Text: {text_temp}")
    print(f"Vectorized Text: {text_vec_temp}\n")
```

```
Original Text: Ard 4 lor...
Vectorized Text: [569  44  86   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0]

Original Text: Nowadays people are notixiquating the laxinorficated opportunity for bambling of entropication.... Have you ever oblisin
Vectorized Text: [3435  271   24 6074    6 6479 1767   14 8098   16 7302   19    4  372
 6045 5987   35 2822   14    6 6314 8267   16 7193   13   10  176 7823
    2  112   13   10 8162 4482   35 1233 6582    0    0    0]

Original Text: Que pases un buen tiempo or something like that
Vectorized Text: [5637 5901  831 7911 4868   31  200   59   18    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0]

Original Text: Moby Pub Quiz.Win a £100 High Street prize if u know who the new Duchess of Cornwall will be? Txt her first name to 8227
Vectorized Text: [2063  657  753  183    5  512 1412 1022  158   38    7   58  119    6
  106 3871   16 2762   37   39   77  110  211  272    3 1465 1318   90
  324  242 4374 1206    0    0    0    0    0    0    0    0]

Original Text: I accidentally brought em home in the box
Vectorized Text: [   2 2249 2204 1071   83    9    6  349    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0]
```

```
# Get the vocabulary
VOCAB = text_vectorizer.get_vocabulary()

# Let's have a look at the tokens present in the vocabulary
print(f"Vocabulary size: {len(VOCAB)}")
print(f"Vocabulary: {VOCAB[150:200]}")
```

```
Vocabulary size: 8841
Vocabulary: ['number', 'message', 'e', 've', 'tomorrow', 'say', 'won', 'right', 'prize', 'already', 'after', 'said', 'ask', 'doing', 'c
```

Data Splitting

As we have our processing functions ready, let's split the data into **training and testing**, and also apply the **Text Vectorization**.

```
# Split the data into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42, shuffle=True)

# Apply the Text Vectorization
X_train = text_vectorizer(X_train)
X_test = text_vectorizer(X_test)

# One Hot Vectors
Xoh_train = tf.one_hot(X_train, depth=10000)
Xoh_test  = tf.one_hot(X_test, depth=10000)
```

```
class TokenAndPositionalEmbedding(layers.Layer):

    def __init__(self, embedding_dims, vocab_size, seq_len, **kwargs):
        super(TokenAndPositionalEmbedding, self).__init__(**kwargs)

        # Initialize parameters
        self.seq_len = seq_len
        self.vocab_size = vocab_size
        self.embedding_dims = embedding_dims
        self.embed_scale = tf.math.sqrt(tf.cast(embedding_dims, tf.float32))

        # Define layers
        self.token_embedding = layers.Embedding(
            input_dim=vocab_size,
            output_dim=embedding_dims,
            name="token_embedding"
        )

        self.positional_embedding = layers.Embedding(
            input_dim=seq_len,
            output_dim=embedding_dims,
            name="positional_embedding"
        )

    def call(self, inputs):
        seq_len = tf.shape(inputs)[1]

        # Token Embedding
        token_embedding = self.token_embedding(inputs)
        token_embedding *= self.embed_scale

        # Positional Embedding
        positions = tf.range(start=0, limit=seq_len, delta=1)
        positional_embedding = self.positional_embedding(positions)

        # Add Token and Positional Embedding
        embeddings = token_embedding + positional_embedding

        return embeddings


    def get_config(self):
        config = super(TokenAndPositionalEmbedding, self).get_config()
        config.update({
            'embedding_dims': self.embedding_dims,
            'vocab_size': self.vocab_size,
            'seq_len': self.seq_len,
        })
        return config
```

```
# Let's look what the layer do.
temp_embeds = TokenAndPositionalEmbedding(embed_dim, vocab_size, max_seq_len)(X_train[:1])
temp_embeds
```

```
<tf.Tensor: shape=(1, 40, 256), dtype=float32, numpy=
array([[[ 0.3891715 ,  0.38913023, -0.47816482, ..., -0.32396913,
          0.47747707, -0.4279145 ],
        [ 0.79614294, -0.48260328, -0.1703661 , ...,  0.04113927,
          0.06693406,  0.14104116],
        [-0.5814726 , -0.5571734 ,  0.534974  , ...,  0.7113527 ,
          0.23286858, -0.4773966 ],
        ...,
        [ 0.08254623, -0.5658101 ,  0.70171624, ...,  0.01840043,
         -0.4055661 , -0.30367637],
        [ 0.04067282, -0.59197736,  0.7193675 , ..., -0.00395709,
         -0.3587655 , -0.21139917],
        [ 0.06172895, -0.6155349 ,  0.70183825, ..., -0.03303777,
         -0.38995034, -0.2888692 ]]], dtype=float32)>
```

Transformer Layer

```
class TransformerLayer(layers.Layer):

    def __init__(self, num_heads: int, dropout_rate: float, embedding_dims: int, ff_dim: int, **kwargs):
        super(TransformerLayer, self).__init__(**kwargs)

        # Initialize Parameters
        self.num_heads = num_heads
        self.dropout_rate = dropout_rate
        self.embedding_dims = embedding_dims
        self.ff_dim = ff_dim

        # Initialize Layers
        self.mha = layers.MultiHeadAttention(num_heads=num_heads, key_dim=embedding_dims, dropout=dropout_rate)
        self.ln1 = layers.LayerNormalization(epsilon=1e-6)

        self.ffn = keras.Sequential([
            layers.Dense(ff_dim, activation='relu', kernel_initializer='he_normal'),
            layers.Dense(embedding_dims)
        ])
        self.ln2 = layers.LayerNormalization(epsilon=1e-6)

    def call(self, inputs):
        """Forward pass of the Transformer Layer.

        Args:
            inputs: Tensor with shape `(batch_size, seq_len, embedding_dims)` representing the input sequence.

        Returns:
            Tensor with shape `(batch_size, seq_len, embedding_dims)` representing the output sequence after applying the Transformer Layer
        """

        # Multi-Head Attention
        attention = self.mha(inputs, inputs, inputs)

        # Layer Normalization and Residual Connection
        normalized1 = self.ln1(attention + inputs)

        # Feedforward Network
        ffn_out = self.ffn(normalized1)

        # Layer Normalization and Residual Connection
        normalized2 = self.ln2(ffn_out + normalized1)

        return normalized2

    def get_config(self):
        """Get the configuration of the Transformer Layer.

        Returns:
            Dictionary with the configuration of the layer.
        """
        config = super(TransformerLayer, self).get_config()
        config.update({
            "num_heads": self.num_heads,
            "dropout_rate": self.dropout_rate,
            "embedding_dims": self.embedding_dims,
            "ff_dim": self.ff_dim
        })
        return config
```

```
# Transformer layers execution
TransformerLayer(num_heads=num_heads, embedding_dims=embed_dim, ff_dim=ff_dim, dropout_rate=0.1)(temp_embeds)
```

```
<tf.Tensor: shape=(1, 40, 256), dtype=float32, numpy=
array([[[-0.00509165, -0.2659131 , -0.9958785 , ..., -0.01139773,
          0.6033171 , -1.1317413 ],
        [ 1.2146025 , -1.6704727 , -1.2876291 , ...,  0.15295842,
         -0.01600631,  0.3441215 ],
        [-1.1809858 , -1.7543097 ,  0.47527918, ...,  1.6960139 ,
          0.58979076, -2.0903115 ],
        ...,
        [-0.03364862, -1.3067727 ,  1.8767515 , ...,  0.43261683,
         -1.6341335 , -1.185184  ],
        [-0.14125913, -1.3276275 ,  1.8349987 , ...,  0.43384203,
         -1.5241923 , -1.0942626 ],
```

```
[-0.11927039, -1.3352348 ,  1.8447485 , ...,  0.34539884,
 -1.6663549 , -1.2077408 ]]], dtype=float32)>
```

Transformer Text Classification Model

It's time to combine the **Token and Positional Embedding l**ayer and the **Transformer layer** to make a **Transformer Network architecture** for **text classification**.

```
# Input layer
InputLayer = layers.Input(shape=(max_seq_len,), name="InputLayer")

# Embedding Layer
embeddings = TokenAndPositionalEmbedding(embed_dim, vocab_size, max_seq_len, name="EmbeddingLayer")(InputLayer)

# Transformer Layer
encodings = TransformerLayer(num_heads=num_heads, embedding_dims=embed_dim, ff_dim=ff_dim, dropout_rate=0.1, name="TransformerLayer")(embeddi

# Classifier
gap = layers.GlobalAveragePooling1D(name="GlobalAveragePooling")(encodings)
drop = layers.Dropout(0.5, name="Dropout")(gap)
OutputLayer = layers.Dense(1, activation='sigmoid', name="OutputLayer")(drop)

# Model
model = keras.Model(InputLayer, OutputLayer, name="TransformerNet")

# Model Architecture Summary
model.summary()
```

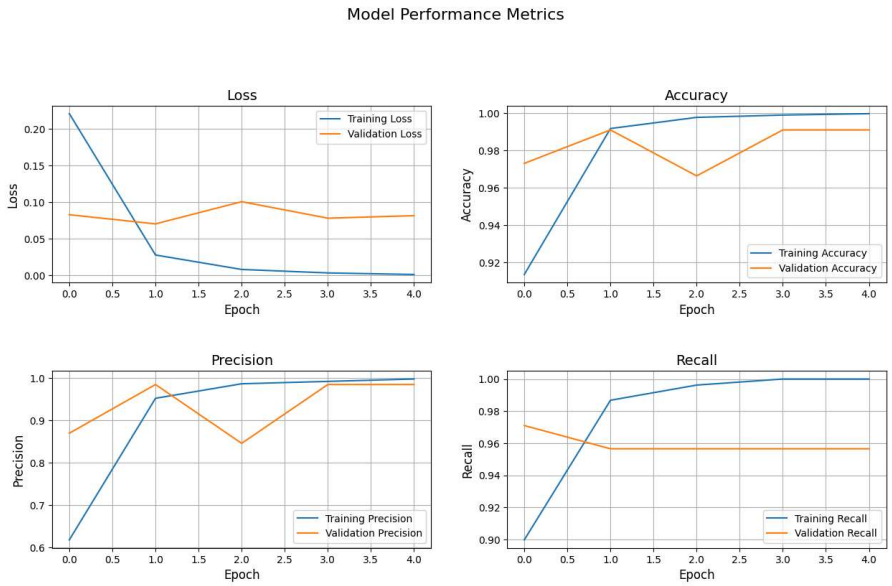
```
Model: "TransformerNet"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 InputLayer (InputLayer)     [(None, 40)]              0

 EmbeddingLayer (TokenAndPo  (None, 40, 256)           2570240
 sitionalEmbedding)

 TransformerLayer (Transfor  (None, 40, 256)           1118848
 merLayer)

 GlobalAveragePooling (Glob  (None, 256)               0
 alAveragePooling1D)

 Dropout (Dropout)           (None, 256)               0

 OutputLayer (Dense)         (None, 1)                 257

=================================================================
Total params: 3689345 (14.07 MB)
Trainable params: 3689345 (14.07 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Transformer Training

```
# Compile the Model
model.compile(
    loss='binary_crossentropy',
    optimizer='adam',
    metrics=[
        keras.metrics.BinaryAccuracy(name='accuracy'),
        keras.metrics.Precision(name='precision'),
        keras.metrics.Recall(name='recall'),
        keras.metrics.AUC(name='auc'),
    ]
)

# Train Model
history = model.fit(
    X_train, y_train,
    validation_split=0.1,
    batch_size=batch_size,
    epochs=epochs,
    callbacks=callbacks,
    class_weight=class_weights
)
```

```
Epoch 1/100
126/126 [==============================] - 47s 342ms/step - loss: 0.2212 - accuracy: 0.9135 - precision: 0.6182 - recall: 0.8998 - auc:
Epoch 2/100
/usr/local/lib/python3.10/dist-packages/keras/src/engine/training.py:3103: UserWarning:

You are saving your model as an HDF5 file via `model.save()`. This file format is considered legacy. We recommend using instead the nat

126/126 [==============================] - 47s 370ms/step - loss: 0.0276 - accuracy: 0.9918 - precision: 0.9526 - recall: 0.9868 - auc:
Epoch 3/100
126/126 [==============================] - 50s 400ms/step - loss: 0.0078 - accuracy: 0.9978 - precision: 0.9869 - recall: 0.9962 - auc:
Epoch 4/100
126/126 [==============================] - 44s 351ms/step - loss: 0.0031 - accuracy: 0.9990 - precision: 0.9925 - recall: 1.0000 - auc:
Epoch 5/100
126/126 [==============================] - 43s 341ms/step - loss: 9.5540e-04 - accuracy: 0.9998 - precision: 0.9981 - recall: 1.0000 -
```

```
# Plot metrics
fig, axes = plt.subplots(nrows=2, ncols=2, figsize=(15, 8))
plt.subplots_adjust(hspace=0.5)

axes[0, 0].plot(history.history['loss'], label='Training Loss')
axes[0, 0].plot(history.history['val_loss'], label='Validation Loss')
axes[0, 0].set_title('Loss', fontsize=14)
axes[0, 0].set_xlabel('Epoch', fontsize=12)
axes[0, 0].set_ylabel('Loss', fontsize=12)
axes[0, 0].grid(True)
axes[0, 0].legend(fontsize=10)

axes[0, 1].plot(history.history['accuracy'], label='Training Accuracy')
axes[0, 1].plot(history.history['val_accuracy'], label='Validation Accuracy')
axes[0, 1].set_title('Accuracy', fontsize=14)
axes[0, 1].set_xlabel('Epoch', fontsize=12)
axes[0, 1].set_ylabel('Accuracy', fontsize=12)
axes[0, 1].grid(True)
axes[0, 1].legend(fontsize=10)

axes[1, 0].plot(history.history['precision'], label='Training Precision')
axes[1, 0].plot(history.history['val_precision'], label='Validation Precision')
axes[1, 0].set_title('Precision', fontsize=14)
axes[1, 0].set_xlabel('Epoch', fontsize=12)
axes[1, 0].set_ylabel('Precision', fontsize=12)
axes[1, 0].grid(True)
axes[1, 0].legend(fontsize=10)

axes[1, 1].plot(history.history['recall'], label='Training Recall')
axes[1, 1].plot(history.history['val_recall'], label='Validation Recall')
axes[1, 1].set_title('Recall', fontsize=14)
axes[1, 1].set_xlabel('Epoch', fontsize=12)
axes[1, 1].set_ylabel('Recall', fontsize=12)
axes[1, 1].grid(True)
axes[1, 1].legend(fontsize=10)

fig.suptitle('Model Performance Metrics', fontsize=16, y=1.05)
plt.show()
```




```
# Evaluate model performance on test data
loss, acc, precision, recall, auc = model.evaluate(X_test, y_test, verbose=0)

# Show the model performance
print('Test loss      :', loss)
print('Test accuracy  :', acc*100)
print('Test precision :', precision*100)
print('Test recall    :', recall*100)
print('Test AUC       :', auc*100)
```

```
Test loss      : 0.05889171361923218
Test accuracy  : 98.65471124649048
Test precision : 95.89040875434875
Test recall    : 93.95973086357117
Test AUC       : 99.31496381759644
```

Transformer Predictions

```
def decode_tokens(tokens):
    """
    This function takes in a list of tokenized integers and returns the corresponding text based on the provided vocabulary.

    .
for _ in range(10):
    # Randomly select a text from the testing data.
    index = np.random.randint(1,len(X_test))
    tokens = X_test[index-1:index]
    label = y_test[index]

    # Feed the tokens to the model
    print(f"\nModel Prediction\n{'-'*100}")
    proba = 1 if model.predict(tokens, verbose=0)[0][0]>0.5 else 0
    pred = label_encoder.inverse_transform([proba])
    print(f"Message: '{decode_tokens(tokens[0])}' | Prediction: {pred[0].title()} | True : {label_encoder.inverse_transform([label])[0].title
```

```
Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'well thats nice too bad i cant eat it' | Prediction: Ham | True : Ham


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'height of oh shit situation a guy throws a luv letter on a gal but falls on her brothers head whos a gay d' | Prediction: Ham


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'on the way to office da' | Prediction: Ham | True : Ham


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'i had a good time too its nice to do something a bit different with my weekends for a change see ya soon' | Prediction: Ham |


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'thank u it better work out cause i will feel used otherwise' | Prediction: Ham | True : Ham


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'don know this week i m going to tirunelvai da' | Prediction: Ham | True : Ham


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'never blame a day in ur life good days give u happiness bad days give u experience both are essential in life all are gods bl


Model Prediction
----------------------------------------------------------------------------------------------------
Message: 'don t make life too stressfull always find time to laugh it may not add years to your life but surely adds more life to ur ye
```