CUSTOMER CHURN PREDICTION USING LLMs WITH SHAP AND BEHAVIOR SUMMARIES:

Import libraries:

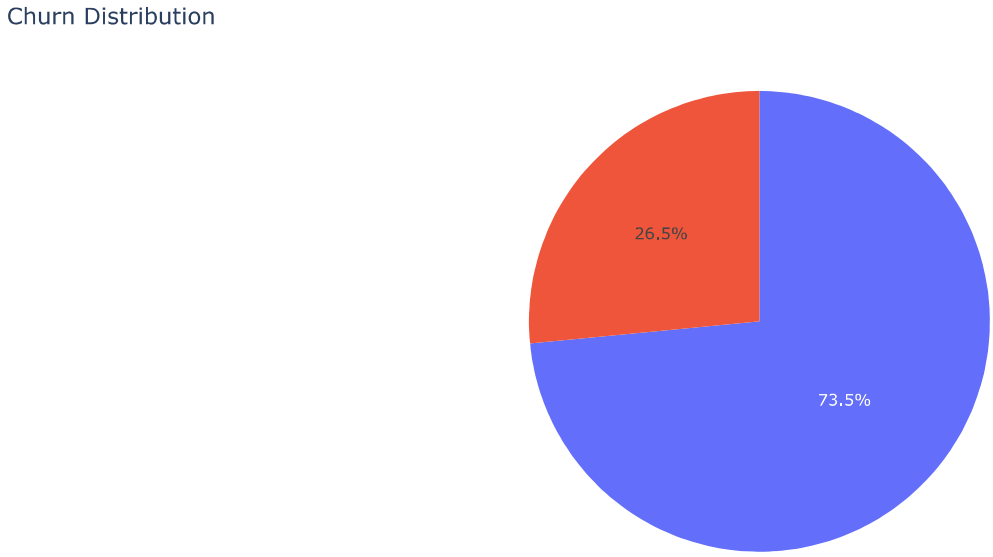
```
import pandas as pd
import numpy as np
import random
import plotly.express as px
import plotly.graph_objects as go
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from transformers import T5Tokenizer, T5ForConditionalGeneration, Trainer, TrainingArguments
from torch.utils.data import Dataset
import torch


# Load and Inspect Data
df = pd.read_csv("/content/drive/MyDrive/PDS_FINAL/Customer_churn.csv")
original_df = df.copy()


original_df = df.copy()


# Initial inspection
print("Initial shape:", df.shape)
print("\nData Types:\n", df.dtypes)
print("\nMissing values:\n", df.isnull().sum())
print("\nUnique values in Churn column:", df['Churn'].unique())
```

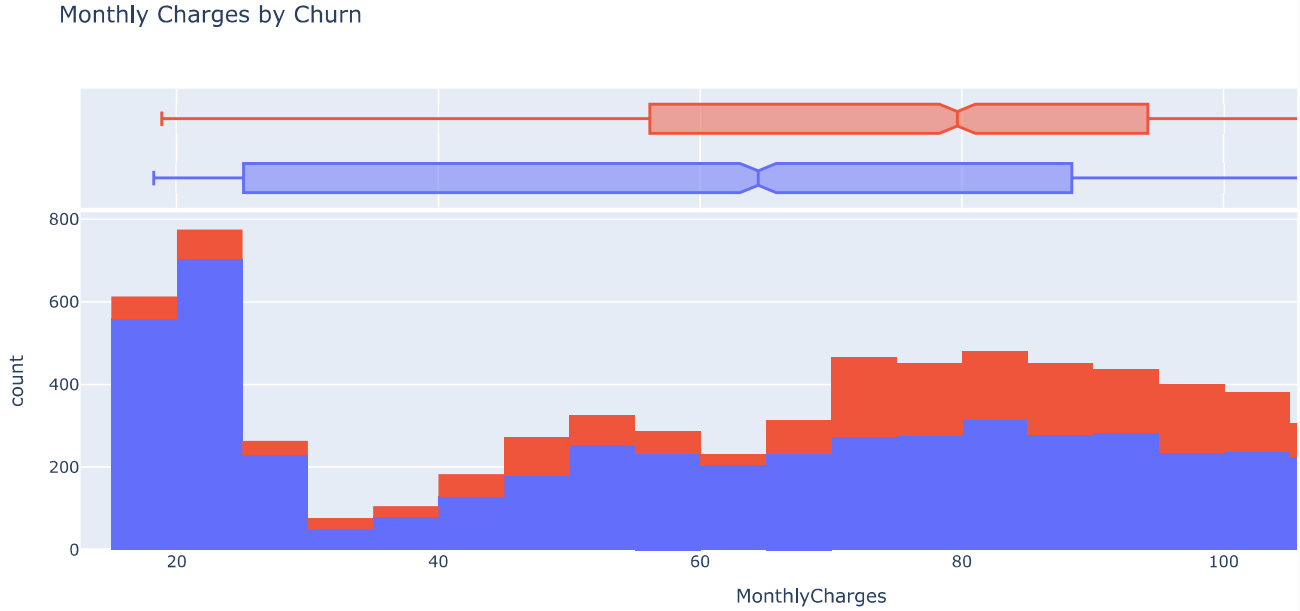
```
Initial shape: (7043, 21)

Data Types:
 customerID          object
gender              object
SeniorCitizen        int64
Partner             object
Dependents          object
tenure               int64
PhoneService        object
MultipleLines       object
InternetService     object
OnlineSecurity      object
OnlineBackup        object
DeviceProtection    object
TechSupport         object
StreamingTV         object
StreamingMovies     object
Contract            object
PaperlessBilling    object
PaymentMethod       object
MonthlyCharges     float64
TotalCharges        object
Churn               object
dtype: object

Missing values:
 customerID         0
gender              0
SeniorCitizen       0
Partner             0
Dependents          0
tenure              0
PhoneService        0
MultipleLines       0
InternetService     0
OnlineSecurity      0
OnlineBackup        0
DeviceProtection    0
TechSupport         0
StreamingTV         0
StreamingMovies     0
Contract            0
PaperlessBilling    0
PaymentMethod       0
MonthlyCharges      0
TotalCharges        0
Churn               0
dtype: int64

Unique values in Churn column: ['No' 'Yes']
```

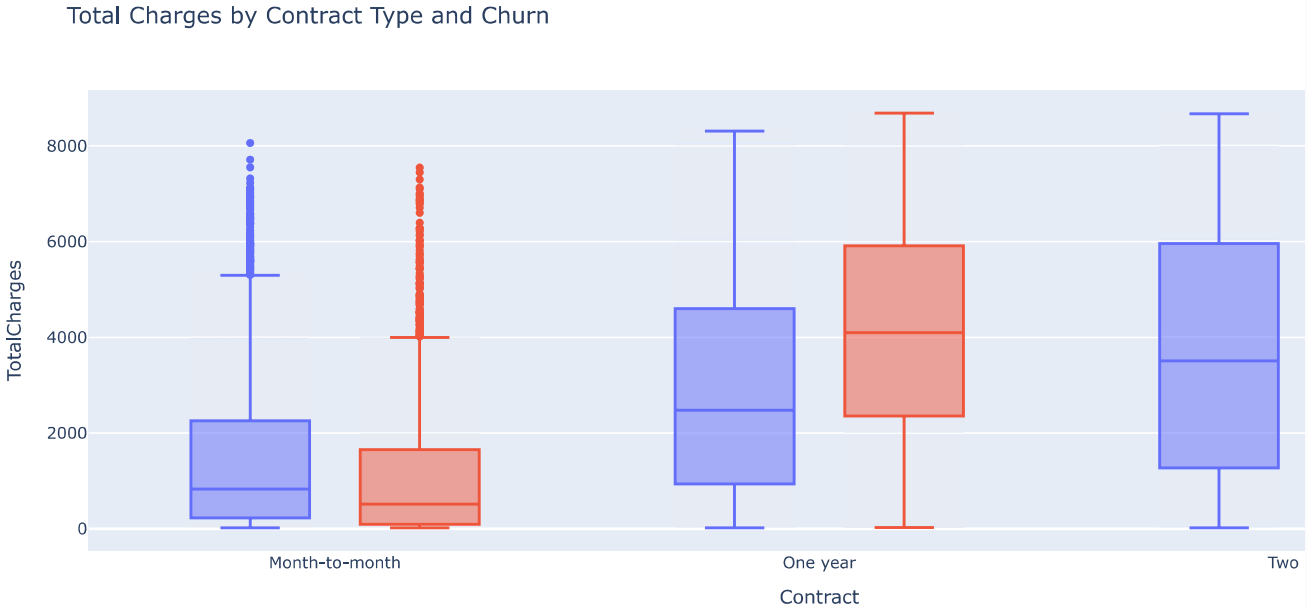
```
# Data Cleaning
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df['TotalCharges'] = df['TotalCharges'].fillna(df['TotalCharges'].median())
df.drop(columns=['customerID'], inplace=True)
df['Churn'] = df['Churn'].map({'No': 'no', 'Yes': 'yes'})
```

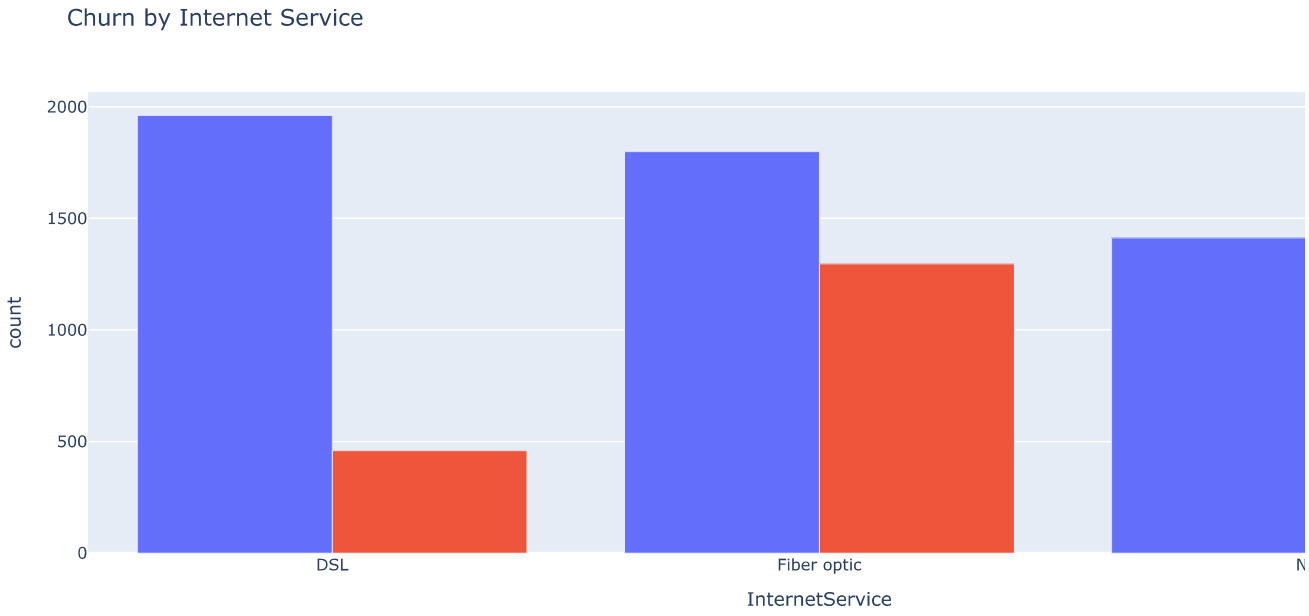
EDA visualizations

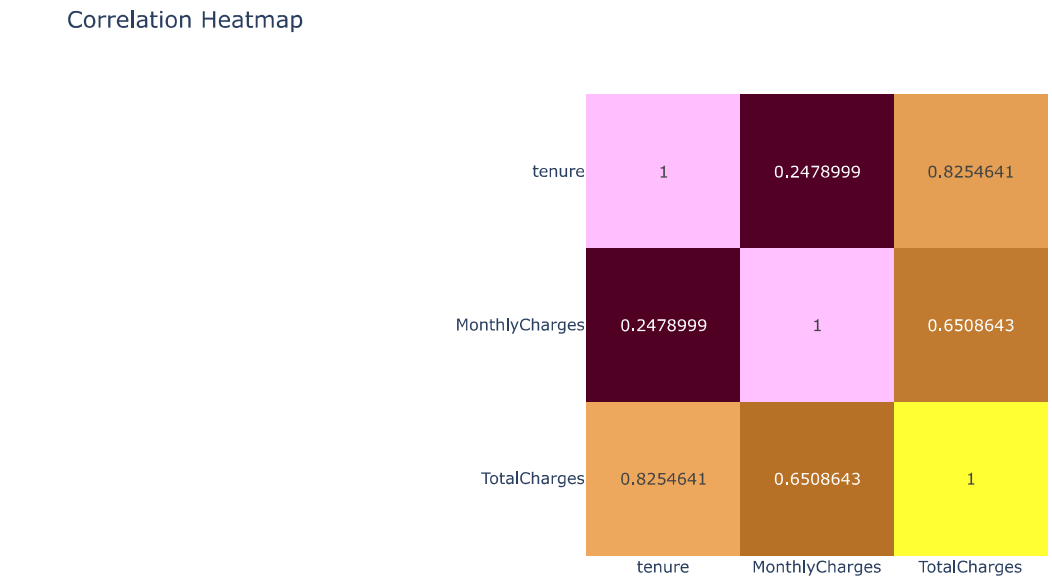
```
px.pie(df, names='Churn', title='Churn Distribution').show()
px.histogram(df, x='MonthlyCharges', color='Churn', marginal='box', nbins=50, title='Monthly Charges by Churn').show()
px.box(df, x='Contract', y='TotalCharges', color='Churn', title='Total Charges by Contract Type and Churn').show()
px.bar(df.groupby(['InternetService', 'Churn']).size().reset_index(name='count'),
       x='InternetService', y='count', color='Churn', barmode='group', title='Churn by Internet Service').show()
px.imshow(df[['tenure', 'MonthlyCharges', 'TotalCharges']].corr(), text_auto=True, title='Correlation Heatmap').show()
```


Churn Distribution
26.5%
73.5%


Monthly Charges by Churn
800
600
400
200
0
count
20
40
60
80
100
MonthlyCharges


Total Charges by Contract Type and Churn
8000
6000
4000
2000
0
TotalCharges
Month-to-month
One year
Two
Contract


Churn by Internet Service
2000
1500
1000
500
0
count
DSL
Fiber optic
N
InternetService


Correlation Heatmap
tenure
MonthlyCharges
TotalCharges
1
0.2478999
0.8254641
0.2478999
1
0.6508643
0.8254641
0.6508643
1
tenure
MonthlyCharges
TotalCharges

```
from transformers import (
    T5Tokenizer, T5ForConditionalGeneration, Trainer, TrainingArguments,
    BertTokenizer, BertForSequenceClassification
)


!pip install --upgrade transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.51.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from transformers) (3.18.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.30.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2.0.2)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (from transformers) (2024.11.6)
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.21.1)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (from transformers) (0.5.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.30.0->trans
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub<1.0,>=0.3
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests->transformers) (2025.4.2
```

BERT-Based Classification

```
df_bert = df.copy()
df_bert['Churn_Label'] = df_bert['Churn'].map({'no': 0, 'yes': 1})

def row_to_prompt(row):
    return (
        f"{row['gender']} customer, {'Senior' if row['SeniorCitizen'] else 'Non-senior'}, "
        f"Contract: {row['Contract']}, MonthlyCharges: ${row['MonthlyCharges']}, "
        f"InternetService: {row['InternetService']}, TechSupport: {row['TechSupport']}, "
        f"TotalCharges: ${row['TotalCharges']}"
    )

df_bert['text'] = df_bert.apply(row_to_prompt, axis=1)
X_train_bert, X_val_bert, y_train_bert, y_val_bert = train_test_split(
    df_bert['text'], df_bert['Churn_Label'], test_size=0.2, stratify=df_bert['Churn_Label'], random_state=42
)

bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

class BertChurnDataset(Dataset):
    def __init__(self, texts, labels):
        self.encodings = bert_tokenizer(texts.tolist(), truncation=True, padding=True, max_length=128)
        self.labels = labels.tolist()
    def __getitem__(self, idx):
        return {
            'input_ids': torch.tensor(self.encodings['input_ids'][idx]),
            'attention_mask': torch.tensor(self.encodings['attention_mask'][idx]),
            'labels': torch.tensor(self.labels[idx])
        }
    def __len__(self): return len(self.labels)

train_dataset_bert = BertChurnDataset(X_train_bert, y_train_bert)
val_dataset_bert = BertChurnDataset(X_val_bert, y_val_bert)

bert_model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

bert_args = TrainingArguments(
    output_dir='./bert_results',
    num_train_epochs=3,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    logging_dir='./bert_logs',
    logging_steps=10
)

bert_trainer = Trainer(
    model=bert_model,
    args=bert_args,
    train_dataset=train_dataset_bert,
    eval_dataset=val_dataset_bert,
    tokenizer=bert_tokenizer
)
```

```
bert_trainer.train()

bert_preds_output = bert_trainer.predict(val_dataset_bert)
bert_preds = np.argmax(bert_preds_output.predictions, axis=1)

print("BERT Accuracy:", accuracy_score(y_val_bert, bert_preds))
print("\nBERT Classification Report:\n", classification_report(y_val_bert, bert_preds))
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
<ipython-input-12-bb345fb68838>:45: FutureWarning:

`tokenizer` is deprecated and will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class` instead.
```

[1059/1059 3:03:52, Epoch 3/3]

Step	Training Loss
10	0.575500
20	0.628200
30	0.454000
40	0.412700
50	0.515400
60	0.472000
70	0.524300
80	0.470600
90	0.445100
100	0.429200
110	0.500600
120	0.495500
130	0.506700
140	0.528200
150	0.408400
160	0.455500
170	0.494400
180	0.490900
190	0.409700
200	0.497800
210	0.502200
220	0.425300
230	0.362200
240	0.447200
250	0.429100
260	0.465900
270	0.452400
370	0.525600
380	0.468300
390	0.494400
400	0.415900

```
# Convert numeric predictions (0/1) to labels (no/yes)
bert_pred_labels = ['yes' if pred == 1 else 'no' for pred in bert_preds]
actual_labels = ['yes' if y == 1 else 'no' for y in y_val_bert]

# first 20 entries only
bert_results_df = pd.DataFrame({
    'Text Prompt': X_val_bert.values[:20],
    'Actual Churn': actual_labels[:20],
    'Predicted Churn': bert_pred_labels[:20]
})


print("📋 First 20 BERT Classification Results:")
display(bert_results_df)
```

```
📋 First 20 BERT Classification Results:
```

	Text Prompt	Actual Churn	Predicted Churn
0	Male customer, Non-senior, Contract: Two year,...	no	no
1	Female customer, Senior, Contract: Month-to-mo...	no	yes
2	Female customer, Non-senior, Contract: One yea...	no	no
3	Male customer, Non-senior, Contract: Month-to-...	no	no
4	Female customer, Non-senior, Contract: Two yea...	no	no
5	Female customer, Non-senior, Contract: Month-t...	no	yes
6	Female customer, Non-senior, Contract: Month-t...	no	no
7	Male customer, Non-senior, Contract: Month-to-...	no	no
8	Female customer, Non-senior, Contract: Two yea...	no	no
9	Male customer, Senior, Contract: Month-to-mont...	yes	no
10	Male customer, Non-senior, Contract: Month-to-...	no	no
11	Female customer, Senior, Contract: Two year, M...	no	no
12	Male customer, Non-senior, Contract: Month-to-...	no	no
13	Female customer, Senior, Contract: Month-to-mo...	yes	yes
14	Female customer, Non-senior, Contract: Month-t...	no	no
15	Male customer, Non-senior, Contract: Two year,...	no	no
16	Female customer, Non-senior, Contract: Two yea...	no	no
17	Male customer, Non-senior, Contract: Month-to-...	no	yes
18	Female customer, Non-senior, Contract: One yea...	no	no
19	Female customer, Non-senior, Contract: One yea...	no	no

```
# -------------------- Convert to Natural Language Prompt --------------------
def row_to_text(row):
    return (
        f"A {row['gender'].lower()} customer, "
        f"{'a senior' if row['SeniorCitizen'] else 'not a senior'}, "
        f"on a {row['Contract'].lower()} contract, paying ${row['MonthlyCharges']} per month, "
        f"uses {row['InternetService'].lower()} internet, {'has' if row['TechSupport'] == 'Yes' else 'no'} tech support, "
        f"total charges ${row['TotalCharges']}. Customer support says: '{row['SupportInteraction']}' "
        f"(Sentiment: {row['Sentiment']})."
    )

df['text'] = df.apply(row_to_text, axis=1)
df['label'] = df['Churn']


# -------------------- Train-Test Split --------------------
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['text'].tolist(), df['label'].tolist(), test_size=0.2, random_state=42, stratify=df['label']
)
```

T5 Model Setup

```
#Tokenizer and Model
tokenizer = T5Tokenizer.from_pretrained('t5-small')
model = T5ForConditionalGeneration.from_pretrained('t5-small')

class ChurnDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len=128):
        self.inputs = tokenizer(texts, padding=True, truncation=True, max_length=max_len, return_tensors="pt")
        self.labels = tokenizer(labels, padding=True, truncation=True, max_length=10, return_tensors="pt")
    def __len__(self): return len(self.inputs['input_ids'])
    def __getitem__(self, idx):
        return {
            'input_ids': self.inputs['input_ids'][idx],
            'attention_mask': self.inputs['attention_mask'][idx],
            'labels': self.labels['input_ids'][idx]
        }

train_dataset = ChurnDataset(train_texts, train_labels, tokenizer)
val_dataset = ChurnDataset(val_texts, val_labels, tokenizer)

args = TrainingArguments(
```

```
    output_dir='./results', num_train_epochs=3,
    per_device_train_batch_size=8, per_device_eval_batch_size=8,
    logging_dir='./logs', logging_steps=10
)

trainer = Trainer(model=model, args=args, train_dataset=train_dataset, eval_dataset=val_dataset)
```

```
tokenizer_config.json: 100%  2.32k/2.32k [00:00<00:00, 162kB/s]
spiece.model: 100%  792k/792k [00:00<00:00, 9.02MB/s]
tokenizer.json: 100%  1.39M/1.39M [00:00<00:00, 9.80MB/s]
You are using the default legacy behaviour of the <class 'transformers.models.t5.tokenization_t5.T5Tokenizer'>. This is expected, ar
config.json: 100%  1.21k/1.21k [00:00<00:00, 82.4kB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better p
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back
model.safetensors: 100%  242M/242M [00:01<00:00, 193MB/s]
generation_config.json: 100%  147/147 [00:00<00:00, 11.3kB/s]
```

```
#Train Model
trainer.train()
```

```
Passing a tuple of `past_key_values` is deprecated and will be removed in Transformers v4.48.0. You should pass an instance of `E
[2115/2115 08:24, Epoch 3/3]
```

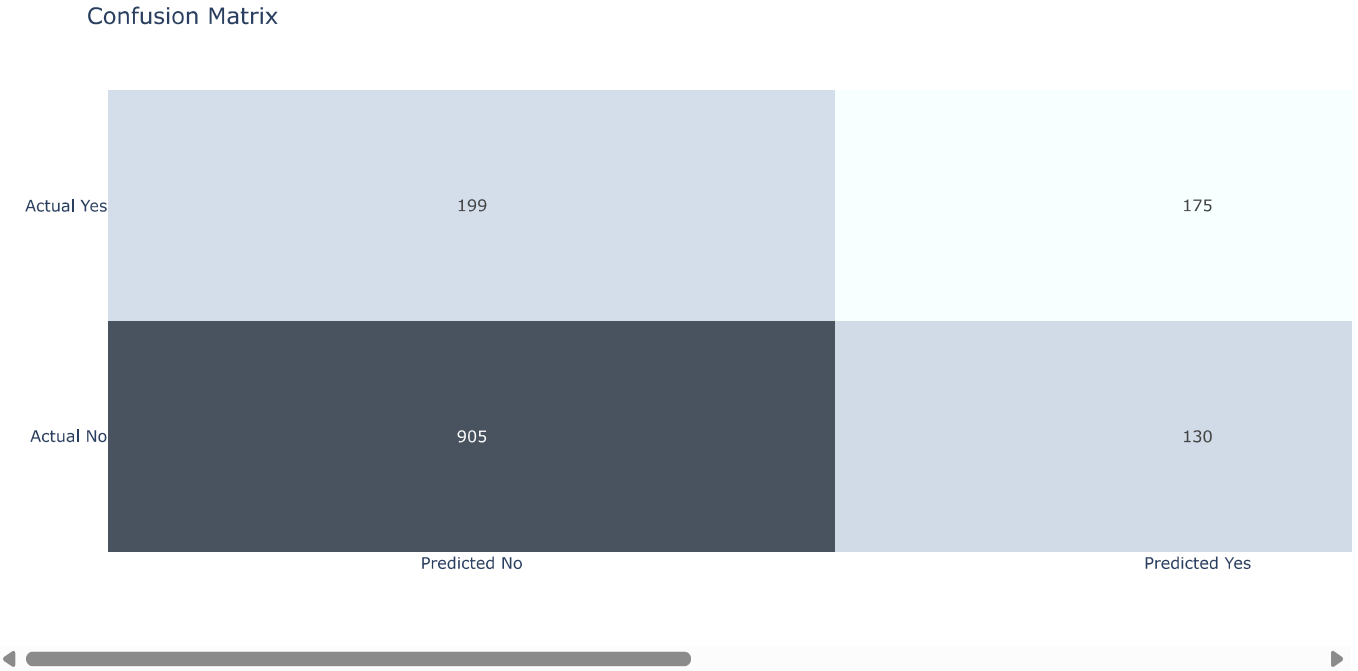
Step	Training Loss
10	9.169800
20	4.044100
30	2.057200
40	1.365100
50	0.837600
60	0.505400
70	0.340600
80	0.339900
90	0.409800
100	0.345900
110	0.310400
120	0.306600
130	0.265600
140	0.265900
150	0.295000
160	0.216300
170	0.273200
180	0.287300
190	0.287200
200	0.262700
210	0.315000
220	0.278800
230	0.288300
240	0.327400
250	0.330300
260	0.217500
270	0.311600
280	0.272600
290	0.252100
300	0.232000

BERT Classification Report:

	precision	recall	f1-score	support
0	0.85	0.87	0.86	1035
	62	0.57	0.59	374
accuracy			0.79	1409
macro avg	0.73	0.72	0.73	1409
	79	0.79	0.79	1409

```
#Evaluate Model
# Tokenize validation texts
val_inputs = tokenizer(val_texts, return_tensors="pt", padding=True, truncation=True).to(model.device)

# Generate predictions
generated_ids = model.generate(
    input_ids=val_inputs["input_ids"],
    attention_mask=val_inputs["attention_mask"],
    max_length=10
)

# Decode predictions
preds_text = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)

# Evaluate
print("Accuracy:", accuracy_score(val_labels, preds_text))
print(classification_report(val_labels, preds_text))

# Confusion Matrix
cm = confusion_matrix(val_labels, preds_text, labels=['no', 'yes'])
fig_cm = go.Figure(data=go.Heatmap(
    z=cm, x=['Predicted No', 'Predicted Yes'], y=['Actual No', 'Actual Yes'],
    colorscale='Blues', text=cm, texttemplate="%{text}"
))
```

```
Accuracy: 0.7665010645848119
              precision    recall  f1-score   support

          no       0.82      0.87      0.85      1035
         yes       0.57      0.47      0.52       374

    accuracy                           0.77      1409
   macro avg       0.70      0.67      0.68      1409
weighted avg       0.75      0.77      0.76      1409
```



SHAP + XGBoost

```
# SHAP Explainability with XGBoost

import shap
import xgboost as xgb
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

# Prepare tabular data (exclude text columns used for T5)
df_xgb = df.copy()
```

```
# Encode categorical columns
categorical_cols = df_xgb.select_dtypes(include=['object']).columns.tolist()
categorical_cols.remove("Churn")  # target column

label_encoders = {}
for col in categorical_cols:
    le = LabelEncoder()
    df_xgb[col] = le.fit_transform(df_xgb[col])
    label_encoders[col] = le
```

```
880	0.252400
```

```
# Drop only columns that exist
columns_to_drop = [col for col in ["Churn", "text", "label", "summary", "customerID", "SupportInteraction"] if col in df_xgb.columns]
X = df_xgb.drop(columns=columns_to_drop)
y = df_xgb["Churn"].map({'no': 0, 'yes': 1})
```

```
940	0.240000
```

```
# Train XGBoost model
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
xgb_model = xgb.XGBClassifier(use_label_encoder=False, eval_metric='logloss')
xgb_model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/xgboost/core.py:158: UserWarning:
[21:34:10] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.
```

```
950	0.293300
960	0.219200
970	0.268100
980	0.191100
```

XGBClassifier

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric='logloss',
              feature_types=None, gamma=None, grow_policy=None,
              importance_type=None, interaction_constraints=None,
              learning_rate=None, max_bin=None, max_cat_threshold=None,
              max_cat_to_onehot=None, max_delta_step=None, max_depth=None,
              max_leaves=None, min_child_weight=None, missing=nan,
              monotone_constraints=None, multi_strategy=None, n_estimators=None,
              n_jobs=None, num_parallel_tree=None, random_state=None, ...)
```

```
1060	0.254400
```

```
# Apply SHAP
explainer = shap.Explainer(xgb_model, X_train)
shap_values = explainer(X_test)
```

```
# Global feature importance
shap.summary_plot(shap_values, X_test)
```

```
1120	0.326500
1130	0.192000
1140	0.225500
1150	0.239600
1160	0.222700
1170	0.227300
1180	0.236600
1190	0.213800
1200	0.226500
1210	0.281800
1220	0.204000
1230	0.209800
1240	0.221900
1250	0.281300
1260	0.296700
1270	0.310200
1280	0.234100
1290	0.230800
1300	0.283600
1310	0.292000
```