1.importing the dependancies

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import gradio as gr
import joblib
from sklearn.preprocessing import LabelEncoder
from imblearn.over_sampling import SMOTE
from sklearn.model_selection import train_test_split,cross_val_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
import pickle
from sklearn.linear_model import LogisticRegression
```

2.Data Loading and Understanding

```
#load teh csv data to a pandas dataframe
df = pd.read_csv("/content/WA_Fn-UseC_-Telco-Customer-Churn.csv")
```

```
# Display first few rows
df.head()
```

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	Onl
0	7590-VHVEG	Female	0	Yes	No	1	No	No phone service	DSL	No	
1	5575-GNVDE	Male	0	No	No	34	Yes	No	DSL	Yes	
2	3668-QPYBK	Male	0	No	No	2	Yes	No	DSL	Yes	
3	7795-CFOCW	Male	0	No	No	45	No	No phone service	DSL	Yes	
4	9237-HQITU	Female	0	No	No	2	Yes	No	Fiber optic	No	

```
# Shape of the dataset
print("Shape:", df.shape)
# Column names
print("Columns:", df.columns.tolist())
# Data types and non-null values
df.info()
# Summary statistics for numeric features
df.describe()
```

```
Shape: (7043, 21)
Columns: ['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents', 'tenure', 'PhoneService', 'MultipleLines', 'InternetServ
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7043 entries, 0 to 7042
Data columns (total 21 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   customerID        7043 non-null   object 
 1   gender            7043 non-null   object 
 2   SeniorCitizen     7043 non-null   int64  
 3   Partner           7043 non-null   object 
 4   Dependents        7043 non-null   object 
 5   tenure            7043 non-null   int64  
 6   PhoneService      7043 non-null   object 
 7   MultipleLines     7043 non-null   object 
 8   InternetService   7043 non-null   object 
 9   OnlineSecurity    7043 non-null   object 
 10  OnlineBackup      7043 non-null   object 
 11  DeviceProtection  7043 non-null   object 
 12  TechSupport       7043 non-null   object 
 13  StreamingTV       7043 non-null   object 
 14  StreamingMovies   7043 non-null   object 
 15  Contract          7043 non-null   object 
 16  PaperlessBilling  7043 non-null   object 
 17  PaymentMethod     7043 non-null   object 
 18  MonthlyCharges    7043 non-null   float64
 19  TotalCharges      7043 non-null   object 
 20  Churn             7043 non-null   object 
dtypes: float64(1), int64(2), object(18)
memory usage: 1.1+ MB
```

	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

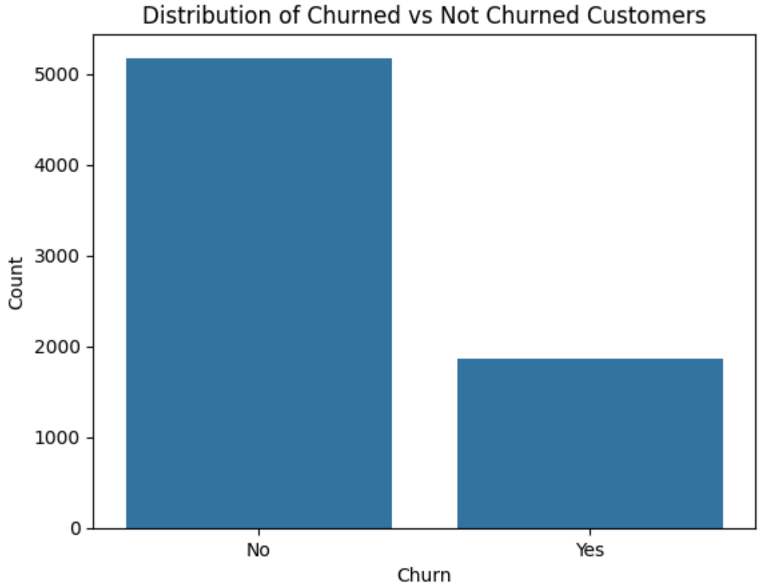
3.Check for Missing Values and Duplicates

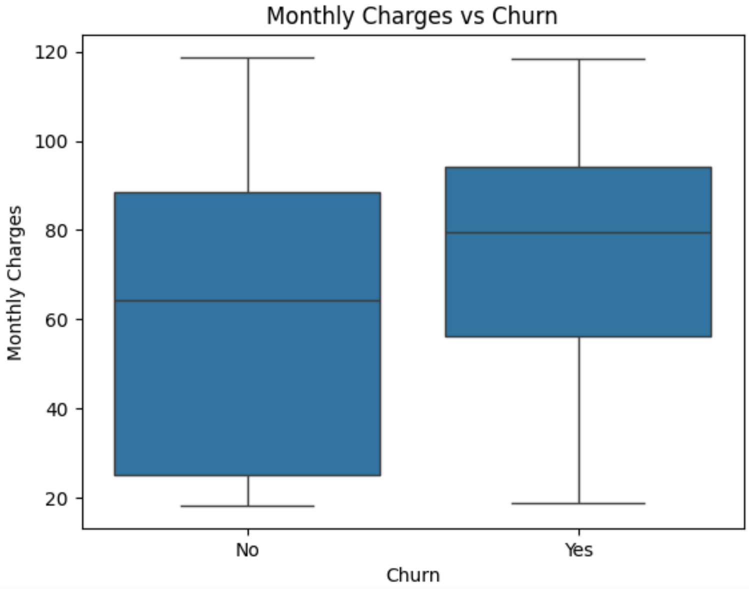
```
# Check for missing values
print(df.isnull().sum())
# Check for duplicates
print("Duplicate rows:", df.duplicated().sum())
```

```
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
Duplicate rows: 0
```

4.Visualize a Few Features

```
# Distribution of Churn
sns.countplot(x='Churn', data=df)
plt.title('Distribution of Churned vs Not Churned Customers')
plt.xlabel('Churn')
plt.ylabel('Count')
plt.show()

# Relationship between Monthly Charges and Churn
sns.boxplot(x='Churn', y='MonthlyCharges', data=df)
plt.title('Monthly Charges vs Churn')
plt.xlabel('Churn')
plt.ylabel('Monthly Charges')
plt.show()
```





5.Identify Target and Features

```
#Identify target and features for churn prediction
target = 'Churn'
features = df.columns.drop(target)
print("Features:", features)
```

```
Features: Index(['customerID', 'gender', 'SeniorCitizen', 'Partner', 'Dependents',
       'tenure', 'PhoneService', 'MultipleLines', 'InternetService',
       'OnlineSecurity', 'OnlineBackup', 'DeviceProtection', 'TechSupport',
       'StreamingTV', 'StreamingMovies', 'Contract', 'PaperlessBilling',
       'PaymentMethod', 'MonthlyCharges', 'TotalCharges'],
      dtype='object')
```

6.Convert Categorical Columns to Numerical

```
# Identify categorical columns
categorical_cols = df.select_dtypes(include=['object']).columns
print("Categorical Columns:", categorical_cols.tolist())
```

```
# Convert binary categorical columns using LabelEncoder
label_encoder = LabelEncoder()
for col in categorical_cols:
    if df[col].nunique() == 2:
        df[col] = label_encoder.fit_transform(df[col])
    else:
        df = pd.get_dummies(df, columns=[col], drop_first=True)
```

```
Categorical Columns: ['customerID', 'gender', 'Partner', 'Dependents', 'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSe
```

7.One-Hot Encoding

1.Separate features and target first:

```
# Save target variable separately
target = 'Churn'
y = df[target]

# Drop target from features
X = df.drop(columns=[target])
```

2.One-hot encode only the features:

```
# One-hot encode features
X_encoded = pd.get_dummies(X, drop_first=True)

# If needed, encode the target (binary label)
from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)  # "Yes"/"No" → 1/0
```

8.Feature Scaling

```
# Separate target variable
target = 'Churn'
y = df[target]

# Drop target from features
X = df.drop(columns=[target])

# One-hot encode features
X_encoded = pd.get_dummies(X, drop_first=True)

# Encode the target ("Yes"/"No") to 1/0
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
```

9.Train-Test Split

```
# Split data
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y_encoded, test_size=0.2, random_state=42)
```

10.Model Building

```
# Train model
model = LogisticRegression(max_iter=1000)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_test)
```

11.Evaluation

```
# Evaluate
print("Accuracy:", accuracy_score(y_test, y_pred))
print("\nClassification Report:\n", classification_report(y_test, y_pred))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred))
```

```
Accuracy: 0.8246983676366217

Classification Report:
               precision    recall  f1-score   support

           0       0.86      0.91      0.88      1036
           1       0.70      0.59      0.64       373

    accuracy                           0.82      1409
   macro avg       0.78      0.75      0.76      1409
weighted avg       0.82      0.82      0.82      1409


Confusion Matrix:
 [[942  94]
 [153 220]]
```

12.Make Predictions from New Input

```
#new inputs values
new_customer = {
    'gender': 'Female',
    'SeniorCitizen': 0,
    'Partner': 'Yes',
    'Dependents': 'No',
    'tenure': 5,
    'PhoneService': 'Yes',
    'MultipleLines': 'No',
    'InternetService': 'DSL',
    'OnlineSecurity': 'Yes',
    'OnlineBackup': 'No',
    'DeviceProtection': 'Yes',
    'TechSupport': 'No',
    'StreamingTV': 'No',
    'StreamingMovies': 'No',
    'Contract': 'Month-to-month',
    'PaperlessBilling': 'Yes',
    'PaymentMethod': 'Electronic check',
    'MonthlyCharges': 70.35,
    'TotalCharges': 350.5
}
```

13.Convert to DataFrame and Encode

```
# Convert to DataFrame
new_df = pd.DataFrame([new_customer])

# Combine with original df to match columns
df_temp = pd.concat([df.drop('Churn', axis=1), new_df], ignore_index=True)

# One-hot encode the combined DataFrame
df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)

# Match the encoded feature order (use df_encoded which is the encoded training features)
df_temp_encoded = df_temp_encoded.reindex(columns=X_encoded.columns, fill_value=0)
```

14.Predict the Churn

```
# Predict churn for new customer input
predicted_churn = model.predict(df_temp_encoded)

# Output result
print("🔮 Churn Prediction:", "Yes" if predicted_churn[0] == 1 else "No")
```

```
🔮 Churn Prediction: No
```

15.Deployment-Building an Interactive App

```
!pip install gradio
```

```
Collecting starlette<1.0,>=0.40.0 (from gradio)
Collecting starlette<1.0,>=0.40.0 (from gradio)
  Downloading starlette-0.46.2-py3-none-any.whl.metadata (6.2 kB)
Collecting tomlkit<0.14.0,>=0.12.0 (from gradio)
  Downloading tomlkit-0.13.2-py3-none-any.whl.metadata (2.7 kB)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.11/dist-packages (from gradio) (0.15.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.11/dist-packages (from gradio) (4.13.2)
Collecting uvicorn>=0.14.0 (from gradio)
  Downloading uvicorn-0.34.2-py3-none-any.whl.metadata (6.5 kB)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gradio) (2025.3.2)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.11/dist-packages (from gradio-client==1.10.0->gra
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.11/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (2025.4.26)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.11/dist-packages (from httpx>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.11/dist-packages (from httpcore==1.*->httpx>=0.24.1->gradio) (
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (3.18.0
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (2.32.3
Requirement already satisfied: tqdm>=4.42.1 in /usr/local/lib/python3.11/dist-packages (from huggingface-hub>=0.28.1->gradio) (4.
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradi
Requirement already satisfied: pydantic-core==2.33.1 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gradio
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.11/dist-packages (from pydantic<2.12,>=2.0->gra
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (8.1.8)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (1.5
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.11/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,>=
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.11/dist-packages (from rich>=10.11.0->typer<1.0,
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hu
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests->huggingface-hub>=0.2
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.11/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->
Downloading gradio-5.29.0-py3-none-any.whl (54.1 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 54.1/54.1 MB 13.6 MB/s eta 0:00:00
Downloading gradio_client-1.10.0-py3-none-any.whl (322 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.9/322.9 kB 22.4 MB/s eta 0:00:00
Downloading aiofiles-24.1.0-py3-none-any.whl (15 kB)
Downloading fastapi-0.115.12-py3-none-any.whl (95 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 95.2/95.2 kB 9.1 MB/s eta 0:00:00
Downloading groovy-0.1.2-py3-none-any.whl (14 kB)
Downloading python_multipart-0.0.20-py3-none-any.whl (24 kB)
Downloading ruff-0.11.8-py3-none-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.5/11.5 MB 115.8 MB/s eta 0:00:00
Downloading safehttpx-0.1.6-py3-none-any.whl (8.7 kB)
Downloading semantic_version-2.10.0-py2.py3-none-any.whl (15 kB)
Downloading starlette-0.46.2-py3-none-any.whl (72 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 72.0/72.0 kB 6.6 MB/s eta 0:00:00
Downloading tomlkit-0.13.2-py3-none-any.whl (37 kB)
Downloading uvicorn-0.34.2-py3-none-any.whl (62 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.5/62.5 kB 5.5 MB/s eta 0:00:00
Downloading ffmpy-0.5.0-py3-none-any.whl (6.0 kB)
Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub, uvicorn, tomlkit, semantic-version, ruff, python-multipart, groovy, ffmpy, aiofiles, starle
Successfully installed aiofiles-24.1.0 fastapi-0.115.12 ffmpy-0.5.0 gradio-5.29.0 gradio-client-1.10.0 groovy-0.1.2 pydub-0.25.1
```

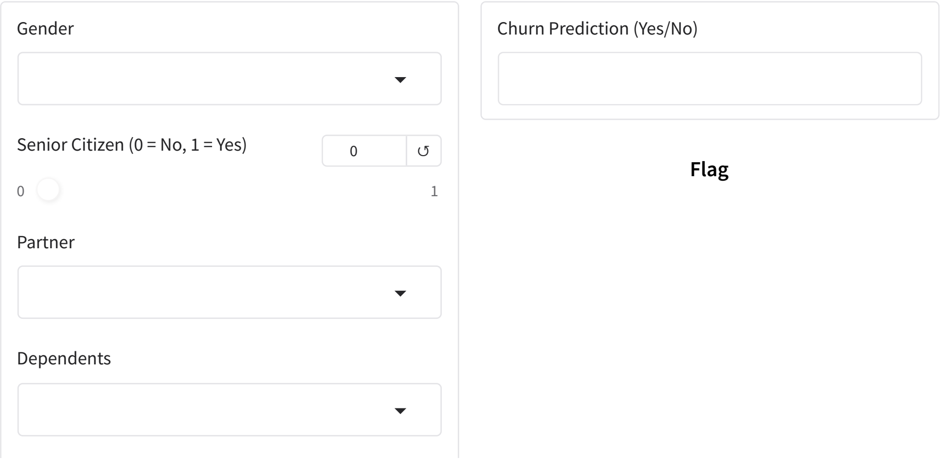
16.Create a Prediction Function

```python
def predict_churn(gender, senior_citizen, partner, dependents, tenure, monthly_charges, total_charges,
                  phone_service, multiple_lines, internet_service, online_security, online_backup,
                  device_protection, tech_support, streaming_tv, streaming_movies, contract,
                  paperless_billing, payment_method):

    # Create input dictionary
    input_data = {
        'gender': gender,
        'SeniorCitizen': int(senior_citizen),
        'Partner': partner,
        'Dependents': dependents,
        'tenure': int(tenure),
        'MonthlyCharges': float(monthly_charges),
        'TotalCharges': float(total_charges),
        'PhoneService': phone_service,
        'MultipleLines': multiple_lines,
        'InternetService': internet_service,
        'OnlineSecurity': online_security,
        'OnlineBackup': online_backup,
        'DeviceProtection': device_protection,
        'TechSupport': tech_support,
        'StreamingTV': streaming_tv,
        'StreamingMovies': streaming_movies,
        'Contract': contract,
        'PaperlessBilling': paperless_billing,
        'PaymentMethod': payment_method
    }
    # Convert the input data into DataFrame
    input_df = pd.DataFrame([input_data])

    # Combine the new input with the original DataFrame (except for 'Churn' target column)
    df_temp = pd.concat([df.drop('Churn', axis=1), input_df], ignore_index=True)

    # One-hot encode the combined DataFrame
    df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)

    # Reindex to match the training dataset's encoded features
    df_temp_encoded = df_temp_encoded.reindex(columns=df_encoded.drop('Churn', axis=1).columns, fill_value=0)

    # Scale the features (use the same scaler as during training)
    scaled
```

17.Create the Gradio Interface

```
# Define the prediction function (assuming `predict_churn` is already defined)
def predict_churn(gender, senior_citizen, partner, dependents, tenure, monthly_charges, total_charges,
                  phone_service, multiple_lines, internet_service, online_security, online_backup,
                  device_protection, tech_support, streaming_tv, streaming_movies, contract,
                  paperless_billing, payment_method):

    # Create input dictionary
    input_data = {
        'gender': gender,
        'SeniorCitizen': int(senior_citizen),
        'Partner': partner,
        'Dependents': dependents,
        'tenure': int(tenure),
        'MonthlyCharges': float(monthly_charges),
        'TotalCharges': float(total_charges),
        'PhoneService': phone_service,
        'MultipleLines': multiple_lines,
        'InternetService': internet_service,
        'OnlineSecurity': online_security,
        'OnlineBackup': online_backup,
        'DeviceProtection': device_protection,
        'TechSupport': tech_support,
        'StreamingTV': streaming_tv,
        'StreamingMovies': streaming_movies,
        'Contract': contract,
        'PaperlessBilling': paperless_billing,
        'PaymentMethod': payment_method
    }

    # Convert the input data into DataFrame
    input_df = pd.DataFrame([input_data])

    # Combine the new input with the original DataFrame (except for 'Churn' target column)
    df_temp = pd.concat([df.drop('Churn', axis=1), input_df], ignore_index=True)

    # One-hot encode the combined DataFrame
    df_temp_encoded = pd.get_dummies(df_temp, drop_first=True)

    # Reindex to match the training dataset's encoded features
    df_temp_encoded = df_temp_encoded.reindex(columns=df_encoded.drop('Churn', axis=1).columns, fill_value=0)

    # Scale the features (use the same scaler as during training)
    scaled_input = scaler.transform(df_temp_encoded.tail(1))

    # Predict churn using the trained model
    prediction = model.predict(scaled_input)

    return "Yes" if prediction[0] == 1 else "No"

# Create the Gradio interface inputs
inputs = [
    gr.Dropdown(['Female', 'Male'], label="Gender"),
    gr.Slider(0, 1, step=1, label="Senior Citizen (0 = No, 1 = Yes)"),
    gr.Dropdown(['Yes', 'No'], label="Partner"),
    gr.Dropdown(['Yes', 'No'], label="Dependents"),
    gr.Slider(0, 72, step=1, label="Tenure (Months)"),
    gr.Slider(20.0, 120.0, step=0.1, label="Monthly Charges"),
    gr.Slider(0.0, 1000.0, step=0.1, label="Total Charges"),
    gr.Dropdown(['Yes', 'No'], label="Phone Service"),
    gr.Dropdown(['Yes', 'No'], label="Multiple Lines"),
    gr.Dropdown(['DSL', 'Fiber optic', 'No'], label="Internet Service"),
    gr.Dropdown(['Yes', 'No'], label="Online Security"),
    gr.Dropdown(['Yes', 'No'], label="Online Backup"),
    gr.Dropdown(['Yes', 'No'], label="Device Protection"),
    gr.Dropdown(['Yes', 'No'], label="Tech Support"),
    gr.Dropdown(['Yes', 'No'], label="Streaming TV"),
    gr.Dropdown(['Yes', 'No'], label="Streaming Movies"),
    gr.Dropdown(['Month-to-month', 'One year', 'Two year'], label="Contract"),
    gr.Dropdown(['Yes', 'No'], label="Paperless Billing"),
    gr.Dropdown(['Electronic check', 'Mailed check', 'Bank transfer', 'Credit card'], label="Payment Method")
]

# Output for churn prediction
output = gr.Textbox(label="Churn Prediction (Yes/No)")

# Launch the Gradio interface
gr.Interface(
    fn=predict_churn,               # Prediction function
    inputs=inputs,                  # Input features
    outputs=output,                 # Output (Churn Prediction)
    title="📊 Customer Churn Prediction",
    description="Enter customer details to predict whether the customer will churn (Yes) or stay (No)."
).launch()
```

```
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatica

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ee838531387124050a.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working
```


Gender
Senior Citizen (0 = No, 1 = Yes)
0
↺
0
1
Partner
Dependents
Churn Prediction (Yes/No)
Flag