Explanation Of Deployment File Code

This Streamlit app is a **Churn Prediction App** that uses a **Random Forest Classifier** to predict whether a customer will churn (leave the service) based on multiple features. Let's break down the code into logical sections and explain each part.

---

1. Import Required Libraries

```
import pandas as pd
import streamlit as st
import time
import plotly.graph_objects as go
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE
```

Purpose

- `pandas`: For data manipulation and loading the dataset.
- `streamlit`: For creating the web-based user interface.
- `time`: To add delays for better UI/UX.
- `plotly.graph_objects`: For creating visual elements like gauge charts.
- `sklearn.pipeline.Pipeline`: Helps in structuring the model workflow.
- `sklearn.preprocessing.StandardScaler`: Scales numeric features for better model performance.

- `sklearn.preprocessing.LabelEncoder`: Encodes categorical values into numeric form.
- `sklearn.ensemble.RandomForestClassifier`: The machine learning model used for classification.
- `sklearn.impute.SimpleImputer`: Handles missing values.
- `sklearn.model_selection.train_test_split`: Splits the data into training and testing sets.
- `imblearn.over_sampling.SMOTE`: Addresses class imbalance in the dataset.

---

2. Load and Preprocess Data

```
data = pd.read_excel(r"C:\Users\shoai\OneDrive\Desktop\Least Used Apps\Python Programs\DS Project 1\Churn (1) (2).xlsx", sheet_name="Churn (1)")
data = data.drop(columns=["Unnamed: 0"], errors='ignore')
```

Purpose

- Loads the dataset from an **Excel file**.
- Drops an unnecessary column `"Unnamed: 0"`, if it exists, to clean the dataset.

---

3. Define Features and Target

```
selected_features = [
    "intl.plan", "voice.plan", "customer.calls", "day.charge", "intl.charge",
    "eve.charge", "night.charge", "day.mins", "eve.mins", "night.mins", "state", "area.code"
]
X = data[selected_features]
y = data["churn"].map({"no": 0, "yes": 1})
```

Purpose

- `selected_features`: Defines the features used for prediction.
- `X = data[selected_features]`: Extracts these features as the input data.

- `y = data["churn"].map({"no": 0, "yes": 1})`: Converts `"yes"` to `1` and `"no"` to `0` for the target variable.

---

4. Encode Categorical Features

```
categorical_cols = ["intl.plan", "voice.plan", "state", "area.code"]
label_encoders = {}

for col in categorical_cols:
    X[col] = X[col].astype(str)
    le = LabelEncoder()
    X[col] = le.fit_transform(X[col])
    label_encoders[col] = le
```

Purpose

- Identifies categorical columns: `"intl.plan"`, `"voice.plan"`, `"state"`, and `"area.code"`.
- Converts them into **string format** before encoding.
- Uses `LabelEncoder` to convert categorical values into **numerical values**.
- Saves encoders in a dictionary (`label_encoders`) for later use in user input transformation.

---

5. Handle Missing Values

```
imputer = SimpleImputer(strategy="median")
X_imputed = pd.DataFrame(imputer.fit_transform(X), columns=X.columns)
```

Purpose

- Uses **median imputation** to fill missing values in the dataset.

---

6. Handle Imbalanced Data with SMOTE

```
smote = SMOTE(sampling_strategy='auto', random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_imputed, y)
```

Purpose

- **SMOTE (Synthetic Minority Over-sampling Technique)** is applied to **balance the dataset** by creating synthetic samples for the minority class (`churn = 1`).

---

7. Train-Test Split

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test
_size=0.2, random_state=42)
```

Purpose

- Splits the resampled dataset into **80% training** and **20% testing**.

---

8. Build and Train the Model

```
pipeline = Pipeline([
    ("scaler", StandardScaler()),
    ("classifier", RandomForestClassifier(n_estimators=150, random_state=42))
])
pipeline.fit(X_train, y_train)
```

Purpose

- Uses a `Pipeline` to:
    1. **Standardize** numeric data using `StandardScaler`.
    2. **Train a Random Forest Classifier** with `150` decision trees.

---

9. Streamlit UI Enhancements

```
st.set_page_config(page_title="Churn Prediction", layout="wide")
```

- Sets up the Streamlit page title and layout.

Dark Mode Toggle

```
dark_mode = st.sidebar.checkbox("🌙 Dark Mode")
if dark_mode:
    st.markdown("""
        <style>
        body { background-color: #121212; color: white; }
        .stButton>button { background-color: #FF5733; color: white; }
        </style>
    """, unsafe_allow_html=True)
```

- Allows users to toggle **dark mode** for better readability.

---

10. UI for User Input

```
st.sidebar.header("🔧 Adjust Inputs")
user_inputs = {}

for col in selected_features:
    if col in categorical_cols:
        user_inputs[col] = st.sidebar.selectbox(f"{col}", label_encoders[col].class
es_)
    else:
        default_value = float(X[col].mean()) if col in X.columns else None
        user_inputs[col] = st.sidebar.slider(f"{col}", float(X[col].min()), float(X[co
l].max()), default_value)
```

Purpose

- Creates a **sidebar with user input fields**.

- Uses `selectbox` for categorical features and `slider` for numerical features.

11. Prediction Logic

```
if st.button("🚀 Predict Churn"):
    with st.spinner('🔍 Analyzing customer data...'):
        time.sleep(2)

    input_df = pd.DataFrame([user_inputs])

    for col in categorical_cols:
        input_df[col] = label_encoders[col].transform([input_df[col][0]])[0]

    input_df = input_df.astype(float)
    input_df = pd.DataFrame(imputer.transform(input_df), columns=selected_fe
atures)

    prediction = pipeline.predict(input_df)[0]
    prediction_proba = pipeline.predict_proba(input_df)[0]
```

Purpose

- Prepares user input for model prediction.
- Encodes categorical inputs using **pre-trained label encoders**.
- Handles missing values using **median imputation**.
- Uses the **trained model** to make predictions.

12. Display Prediction Results

```
churn_text = "Yes" if prediction == 1 else "No"
churn_class = "churn-yes" if prediction == 1 else "churn-no"

st.markdown(f"""
```

```
    <div class="prediction-box {churn_class}">
        🔥 The Predicted Churn is: <strong>{churn_text}</strong>!
    </div>
""", unsafe_allow_html=True)
```

Purpose

- Displays the churn prediction as **"Yes" or "No"** with a color-coded background.

---

13. Gauge Chart for Churn Probability

```
fig = go.Figure(go.Indicator(
    mode="gauge+number",
    value=prediction_proba[1] * 100,
    title={'text': "Churn Probability (%)"},
    gauge={'axis': {'range': [None, 100]},
           'bar': {'color': "red" if prediction == 1 else "green"},
           'steps': [
               {'range': [0, 50], 'color': "lightgreen"},
               {'range': [50, 80], 'color': "orange"},
               {'range': [80, 100], 'color': "red"}
           ]}))

st.plotly_chart(fig, use_container_width=True)
```

Purpose

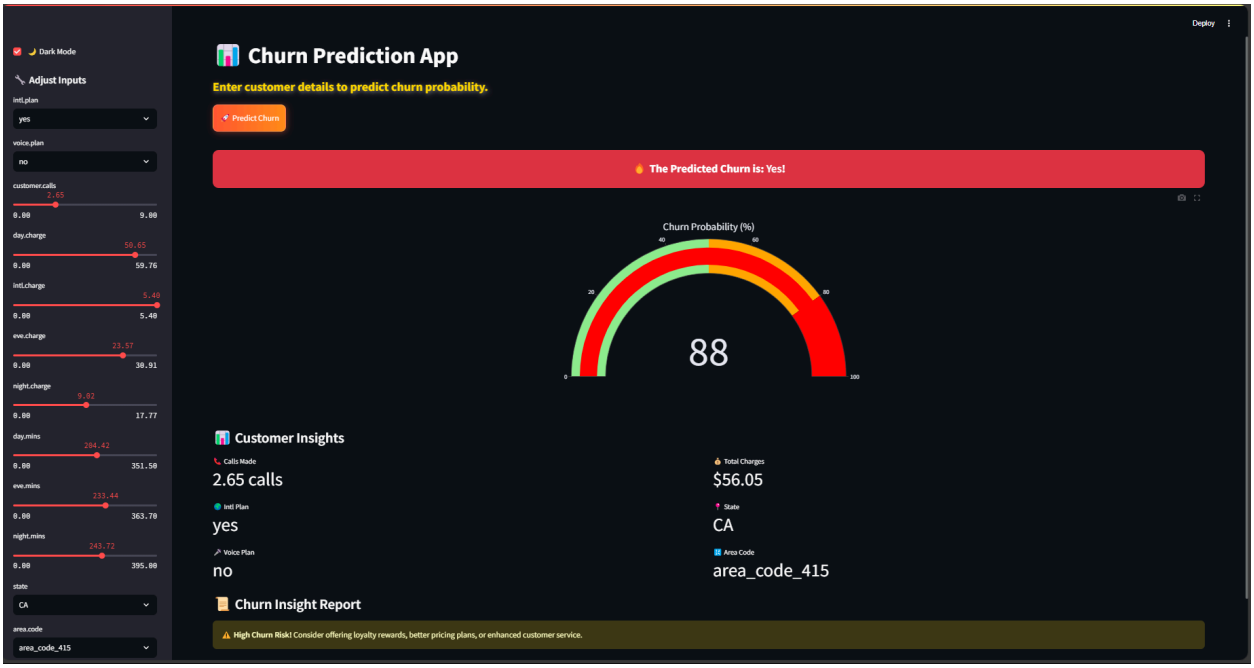
- **Visualizes churn probability** using a **gauge chart**.

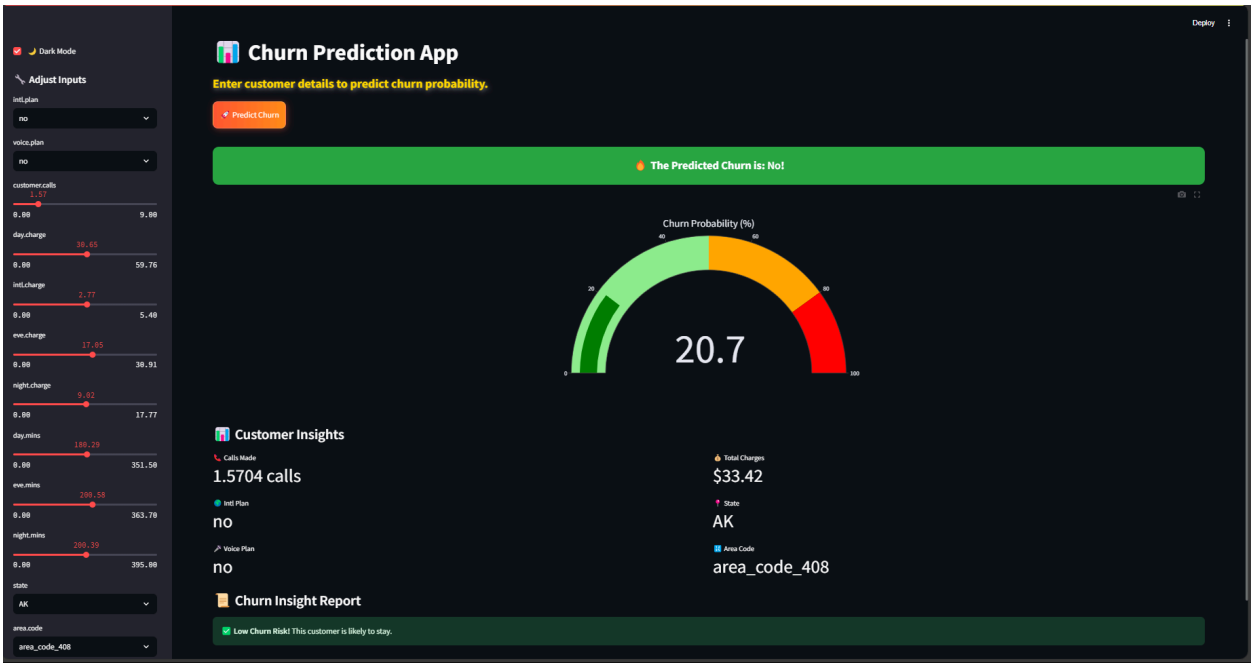
Conclusion

- This **Churn Prediction App** provides **an interactive UI**, **real-time predictions**, and **data visualization**.

- It effectively **handles missing values, imbalanced data, and categorical features**.
- The UI includes **dark mode, color-coded results, and a smooth user experience**. 🚀

Prediction As Yes & No





Command For Running The File in VS Code/Cursor :-

```
cd "your file path" #without the file name
# example :-
cd "C:\Users\shoai\OneDrive\Desktop\Least Used Apps\Python Programs\DS Pro

streamlit run "File's name"

# example :-
streamlit run Churn_UI_Change.py
```

You Can Run The File Using The Above Commands.

It will open as a Local-Host on your Default Browser (e.g., Chrome, Microsoft Edge, Mozirilla Firefox, etc.)