End-to-End Full-Stack ML Project (Customer Churn Example)

1. Project Overview

An end-to-end ML project goes beyond model training — it needs **data ingestion, preprocessing, ML models, evaluation, feature engineering, and a deployed full-stack interface**.
This guide demonstrates a simple setup for **customer churn prediction** with:

- Dataset ingestion (CSV/Excel)
- ML pipeline (sklearn, XGBoost, SMOTE)
- Frontend (Streamlit + Plotly + custom HTML/CSS/JS)
- Localized state/session management (cookies, JSON dumps, URI params)
- Deployment on Streamlit Cloud

---

2. Data Ingestion

- Users **upload CSV/Excel files** directly in the Streamlit app.
- Dataset stored in **session state** for further processing.

```
import pandas as pd
import streamlit as st

uploaded = st.file_uploader("Upload customer dataset (CSV/Excel)")
if uploaded:
    df = pd.read_csv(uploaded) if uploaded.name.endswith(".csv") else pd.read_excel(uploaded)
    st.session_state['raw_data'] = df
```

---

3. Preprocessing & Class Balancing

- Missing values handled (`fillna`, imputation).
- Feature encoding (LabelEncoder / OneHotEncoder).
- Scaling (StandardScaler).
- **SMOTE** applied to balance churn vs non-churn customers.

---

4. Model Training & Evaluation

- **SVM (84.13%)** and **XGBoost (84.25%)** trained/tested.
- Hyperparameter tuning with GridSearchCV.

```
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2)
model = XGBClassifier()
model.fit(X_train, y_train)
acc = accuracy_score(y_test, model.predict(X_test))
st.write(f"Model Accuracy: {acc:.2%}")
```

---

5. Feature Engineering

- Domain-specific features: tenure groups, transaction buckets, customer loyalty tiers.
- Stored in `session_state` for consistency across app tabs.

---

6. Session State & Persistence

6.1 Cookies & JSON Dumps

- Predictions and user inputs saved into **JSON dumps** for reload across sessions.
- **Cookies/localStorage** used to persist session state after refresh/login.

```
import json

# Save session
with open("session_dump.json", "w") as f:
    json.dump(st.session_state, f)

# Load session
with open("session_dump.json") as f:
    st.session_state.update(json.load(f))
```

6.2 URI Parameters

- Streamlit supports **query parameters** to keep state in the URL (useful for sharing filtered dashboards).

```
# Save filter state into URI
params = st.experimental_get_query_params()
st.experimental_set_query_params(view="high_risk", page="dashboard")
```

---

7. Frontend Visualization

- Interactive dashboards with **Plotly** and **Seaborn**.
- HTML + CSS + JS snippets embedded using `st.components.v1.html`.
- Example: customer churn probability bar chart.

---

8. Deployment

- **Streamlit Cloud**: Easy push to GitHub → auto-deploy.
- Persistence:
    - `session_state` → in-memory state while active.
    - `cookies/json dumps` → reload session after app restart.
    - `URI params` → shareable links with filters applied.

---

9. Example User Workflow

1. User uploads dataset (CSV/Excel).
2. Data ingested → preprocessing + SMOTE → ML prediction.
3. Model outputs churn risk with visual insights.
4. Session automatically saved:
    - `session_state` → active navigation.
    - JSON dumps → reload later.
    - Cookies/localStorage → keep auth/session tokens.
    - URI parameters → share dashboard view with a link.
5. Streamlit app deployed to cloud → accessible via browser.

---

10. Tech Stack

- **ML/EDA**: pandas, numpy, sklearn, xgboost, imblearn, seaborn, plotly
- **Frontend/Deployment**: Streamlit, HTML, CSS, JS
- **Persistence**: session_state, cookies, JSON dumps, URI parameters