MoP - Build Your First Churn Prediction Model (Decision Tree)

GOAL:

Train a simple model that predicts whether a customer will churn (1) or stay (0) using easy-to-understand features (age, data usage, complaints, tenure, plan type). You will:

1. Clean & fix missing data (no dropping),
2. Split into train/test,
3. Train a Decision Tree,
4. Evaluate with accuracy + precision/recall/F1 + confusion matrix,
5. (Optional) Tune hyperparameters and save the model.

This MOP helps you to generate random dataset, splits it into training and testing parts, builds a **Decision Tree model** that learns patterns from the training data, and prepares it for future prediction and evaluation.

1) Setup

- Open Anaconda Navigator → Jupyter notebook
- In the browser tab, navigate into churn_p/.
- Create a new notebook -> churn_training.ipynb (use kernel – Python Conda)

```
pip install pandas scikit-learn matplotlib
```
(# we installing python libraries – use anaconda prompt or jupyter notebook you just created)

Restart Python Kernel in case a note comes (#**kernel (Python process)** doesn't automatically reload new libraries — hence you must restart it so it loads the updated packages.)

2) Create a demo dataset (so no one is blocked)

As we don't have any dataset we are creating one for practice. Either we can extract the data from the sources like MYSQL, CSV or json or we create dumpy data.

```
import pandas as pd
import numpy as np

# Reproducibility
rng = np.random.default_rng(42)

n = 300
df = pd.DataFrame({
    "age": rng.integers(18, 70, size=n),
    "usage_gb": np.round(rng.normal(12, 6, size=n).clip(0, None), 1),
    "complaints": rng.integers(0, 6, size=n),
    "tenure_months": rng.integers(1, 60, size=n),
    "plan_type": rng.choice(["Prepaid", "Postpaid"], size=n, p=[0.6, 0.4])
})

# Simple churn signal (not perfect): high complaints + low tenure + low usage more likely to churn
logit = (
    -2.0
    + 0.35*df["complaints"]
    - 0.03*df["tenure_months"]
    - 0.04*df["usage_gb"]
    + (df["plan_type"] == "Prepaid").astype(int)*0.4
)
prob = 1/(1+np.exp(-logit))
df["churn"] = (rng.random(n) < prob).astype(int)

# Intentionally add a few missing/odd values to practice fixing (no row drops)
df.loc[rng.choice(df.index, 5, replace=False), "usage_gb"] = np.nan
df.loc[rng.choice(df.index, 3, replace=False), "age"] = np.nan
df.loc[rng.choice(df.index, 3, replace=False), "plan_type"] = None

df.to_csv("churn_data.csv", index=False)
print("Saved churn_data.csv with shape:", df.shape)
df.head()
```

Your data is ready

3) Load & fix the data **(no dropping) – ETL**

```
import pandas as pd
from sklearn.impute import SimpleImputer

df = pd.read_csv("churn_data.csv")
print("Original shape:", df.shape)
display(df.head())

# --- Numeric imputation: fill NaNs with median (robust to outliers) ---
num_cols = ["age", "usage_gb", "complaints", "tenure_months"]
num_imputer = SimpleImputer(strategy="median")
df[num_cols] = num_imputer.fit_transform(df[num_cols])

# --- Categorical imputation: fill missing with most frequent ---
cat_cols = ["plan_type"]
cat_imputer = SimpleImputer(strategy="most_frequent")
df[cat_cols] = cat_imputer.fit_transform(df[cat_cols])

# --- Encode plan_type to numeric (0=Prepaid, 1=Postpaid) ---
```

```
df["plan_type_enc"] = (df["plan_type"] == "Postpaid").astype(int)

# --- Feature matrix (X) and target (y) ---
X = df[["age", "usage_gb", "complaints", "tenure_months", "plan_type_enc"]]
y = df["churn"].astype(int)

print("Any remaining NaNs? ->", X.isna().sum().sum(), y.isna().sum())

# --- ✅ Save cleaned dataset to CSV ---
df.to_csv("updated_churn_data.csv", index=False)
print("✅ Saved cleaned dataset as 'updated_churn_data.csv' with", len(df), "rows.")
```

Why this is safe: we impute (fill) missing values instead of removing rows, keeping your dataset intact.

Your Dataset is updated

4) Train/Test split & model training

# This will import functions from the scikit-learn (machine learning) library. Helps split your dataset into two parts: one for training the model and one for testing it. DecisionTreeClassifier is a machine learning algorithm that learns to make predictions by building a tree-like structure of decisions.

```
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Split
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42
)

# Train Decision Tree (simple baseline)
clf = DecisionTreeClassifier(max_depth=5, random_state=42)
clf.fit(X_train, y_train)
```

# This is where the **learning happens**. The model looks at all the feature values in X_train and the correct answers in y_train, and learns **patterns** that connect them. After this step, the model can make predictions on unseen data.

```
print("Model trained.")
```

Your MODEL is trained

5) Evaluate (accuracy + precision/recall/F1 + confusion matrix)

This code helps you to check how well your trained Decision Tree model performed. It compares the model’s predictions with the real answers, prints accuracy and other metrics, and then shows a confusion matrix plot to visualize the results.

```
from sklearn.metrics import accuracy_score, precision_recall_fscore_support,
classification_report, confusion_matrix
import matplotlib.pyplot as plt
import numpy as np

# Predict
y_pred = clf.predict(X_test)

# Accuracy
acc = accuracy_score(y_test, y_pred)
print(f"Accuracy: {acc:.2%}")

# Detailed metrics
print("\nClassification report:")
print(classification_report(y_test, y_pred, digits=3))

# Confusion matrix
cm = confusion_matrix(y_test, y_pred)
print("Confusion matrix:\n", cm)

# Top-left (50) -> correctly predicted “Stay”
# Top-right (10) -> false alarms (predicted churn, actually stayed)
# Bottom-right (35) -> correctly predicted “Churn”
# Bottom-left (5) -> missed churns


# Plot nicely
fig, ax = plt.subplots()
ax.imshow(cm, cmap="Blues")
ax.set_title("Confusion Matrix")
ax.set_xlabel("Predicted"); ax.set_ylabel("Actual")
for (i, j), v in np.ndenumerate(cm):
    ax.text(j, i, str(v), ha='center', va='center', fontsize=12)
ax.set_xticks([0,1]); ax.set_yticks([0,1])
ax.set_xticklabels(["Stay (0)", "Churn (1)"]); ax.set_yticklabels(["Stay (0)", "Churn (1)"])
plt.show()
```

CHECK Accuracy of Your MODEL

- Makes predictions using your trained model.
- Calculates how accurate those predictions are.
- Shows detailed metrics like precision, recall, and F1-score.
- Displays a confusion matrix — both as text and as a colored chart — to help you visualize model performance.

6) Explain model insights (feature importance)

```
import pandas as pd
import numpy as np

fi = pd.Series(best_clf.feature_importances_, index=X.columns) if 'best_clf' in globals() else
pd.Series(clf.feature_importances_, index=X.columns)
print("Feature importance (higher = more influence):")
display(fi.sort_values(ascending=False).to_frame("importance"))
```

Typical pattern: complaints and tenure are strong churn drivers; usage & plan type may also matter.

7) Save the model & use it for a new prediction

```
import joblib
model_to_save = best_clf if 'best_clf' in globals() else clf
joblib.dump(model_to_save, "churn_decision_tree.joblib")
print("Saved churn_decision_tree.joblib")

# Example: predict churn for a new customer
new_customer = pd.DataFrame([{
    "age": 28, "usage_gb": 6.0, "complaints": 3, "tenure_months": 4, "plan_type_enc": 0  #
Prepaid
}])
print("Predicted churn (1=churn):", int(model_to_save.predict(new_customer)[0]))
```

8) Deliverables (students submit on google drive)

- churn_data.csv and churn_data.csv

- Notebook churn_training.ipynb with:
- Imputation steps (no row drops)
- Train/test split
- Model training + evaluation + confusion matrix
- Saved model: churn_decision_tree.joblib
- 3–5 line business insight (e.g., "High complaints + short tenure drive churn—offer retention pack in first 6 months.")