Import Libraries

```
# 0) Imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV,
StratifiedKFold
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, roc_auc_score, confusion_matrix, roc_curve
import joblib
import warnings
warnings.filterwarnings("ignore")
```

Load Dataset

```
# Step 3: Load Dataset
import pandas as pd
from google.colab import files

# Upload file
uploaded = files.upload()
df = pd.read_csv("Customer_data - customer_data.csv")

# Display basic dataset info
print("Dataset Shape:", df.shape)
print("\nFirst 5 rows:\n", df.head())
print("\nDataset Info:\n")
print(df.info())
print("\nMissing values:\n", df.isnull().sum())
```

```
<IPython.core.display.HTML object>

Saving Customer_data - customer_data.csv to Customer_data - customer_data.csv
Dataset Shape: (7043, 21)

First 5 rows:
    customerID  gender  SeniorCitizen Partner Dependents  tenure PhoneService
\
0  7590-VHVEG  Female              0     Yes         No       1           No
```

```
1  5575-GNVDE    Male              0      No         No      34          Yes
2  3668-QPYBK    Male              0      No         No       2          Yes
3  7795-CFOCW    Male              0      No         No      45           No
4  9237-HQITU  Female              0      No         No       2          Yes

      MultipleLines InternetService OnlineSecurity  ... DeviceProtection  \
0  No phone service             DSL             No  ...               No
1                No             DSL            Yes  ...              Yes
2                No             DSL            Yes  ...               No
3  No phone service             DSL            Yes  ...              Yes
4                No     Fiber optic             No  ...               No

  TechSupport StreamingTV StreamingMovies        Contract PaperlessBilling  \
0          No          No              No  Month-to-month              Yes
1          No          No              No        One year               No
2          No          No              No  Month-to-month              Yes
3         Yes          No              No        One year               No
4          No          No              No  Month-to-month              Yes

               PaymentMethod MonthlyCharges  TotalCharges  Churn
0           Electronic check          29.85         29.85     No
1               Mailed check          56.95       1889.50     No
2               Mailed check          53.85        108.15    Yes
3  Bank transfer (automatic)          42.30       1840.75     No
4           Electronic check          70.70        151.65    Yes

[5 rows x 21 columns]

Dataset Info:

<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      7032 non-null   float64
 20  Churn             7043 non-null   object
dtypes: float64(2), int64(2), object(17)
memory usage: 1.1+ MB
None

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        11
Churn                0
dtype: int64
```

Preprocessing

```
# Made minimal, necessary transformations so the pipeline is deterministic.
# Replaced spaces-only entries with NaN
df.replace(" ", np.nan, inplace=True)

# Identify the target column and ensure binary numeric encoding.
# This assumes target column is named 'Churn' with values like 'Yes'/'No' or
1/0.
if "Churn" not in df.columns:
    raise KeyError("Target column 'Churn' not found in dataset. Rename the
target column to 'Churn'.")

# Convert common textual target encodings to 0/1
if df["Churn"].dtype == object or df["Churn"].dtype.name == "category":
```

```
    df["Churn"] = df["Churn"].map(lambda x: 1 if str(x).strip().lower() in
["yes", "1", "true", "y"] else 0)

# Ensure type is int
df["Churn"] = df["Churn"].astype(int)

# Print class balance
print("Churn distribution:\n", df["Churn"].value_counts(normalize=True))
```

```
Churn distribution:
 Churn
0    0.73463
1    0.26537
Name: proportion, dtype: float64
```

Encoding Categorical Variables

```
# numerical vs categorical
# Excluding the target column
features = df.drop(columns=["Churn"]).columns.tolist()
num_cols = df[features].select_dtypes(include=["int64",
"float64"]).columns.tolist()
cat_cols = df[features].select_dtypes(include=["object", "category",
"bool"]).columns.tolist()

# If any numeric-like columns are in object dtype (like 'TotalCharges'),
attempt conversion
for c in features:
    if c not in num_cols and c not in cat_cols:
        # infer by trying to convert
        try:
            df[c] = pd.to_numeric(df[c])
            num_cols.append(c)
        except Exception:
            cat_cols.append(c)

print("Numeric columns:", num_cols)
print("Categorical columns:", cat_cols)
```

```
Numeric columns: ['SeniorCitizen', 'tenure', 'MonthlyCharges',
'TotalCharges']
Categorical columns: ['customerID', 'gender', 'Partner', 'Dependents',
'PhoneService', 'MultipleLines', 'InternetService', 'OnlineSecurity',
'OnlineBackup', 'DeviceProtection', 'TechSupport', 'StreamingTV',
'StreamingMovies', 'Contract', 'PaperlessBilling', 'PaymentMethod']
```

```
# 4) Reproducible preprocessing pipeline with imputation (required)
from sklearn.impute import SimpleImputer
```

```
# Numeric pipeline: impute missing with median, then scale
numeric_transformer =  Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="median")),
    ("scaler", StandardScaler())
])

# Categorical pipeline: impute missing with most frequent, then one-hot
encode
categorical_transformer = Pipeline(steps=[
    ("imputer", SimpleImputer(strategy="most_frequent")),
    ("onehot", OneHotEncoder(handle_unknown="ignore", sparse_output=False))
])

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, num_cols),
        ("cat", categorical_transformer, cat_cols)
    ],
    remainder="drop"
)

joblib.dump({"num_cols": num_cols, "cat_cols": cat_cols}, "cols_info.joblib")
print("Preprocessor with imputation created. We'll fit it on training data
only.")
```

```
Preprocessor with imputation created. We'll fit it on training data only.
```

Train- Test Split

```
#Train-test split (stratified)
X = df.drop(columns=["Churn"]).copy()
y = df["Churn"].copy()

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=42, stratify=y
)
print("Train/test split done. Train shape:", X_train.shape, "Test shape:",
X_test.shape)

# Fit preprocessor on training data only
X_train_prep = preprocessor.fit_transform(X_train)
X_test_prep = preprocessor.transform(X_test)

# Save the fitted preprocessor for use with new data
joblib.dump(preprocessor, "preprocessor.joblib")
print("Fitted preprocessor saved to preprocessor.joblib")
print("Transformed shapes:", X_train_prep.shape, X_test_prep.shape)
```

```
Train/test split done. Train shape: (5634, 20) Test shape: (1409, 20)
Fitted preprocessor saved to preprocessor.joblib
Transformed shapes: (5634, 5679) (1409, 5679)
```

Grid Search cv for finding best parameters

```
from sklearn.model_selection import GridSearchCV

log_reg_pipe = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", LogisticRegression(max_iter=1000, solver="lbfgs",
random_state=42))
])

rf_pipe = Pipeline([
    ("preprocessor", preprocessor),
    ("classifier", RandomForestClassifier(random_state=42, n_jobs=-1))
])

log_reg_params = {"classifier_C": [0.1, 1, 10]}
rf_params = {"classifier_n_estimators": [100], "classifier_max_depth": [10,
None]}

log_grid = GridSearchCV(log_reg_pipe, log_reg_params, cv=3, scoring="f1",
n_jobs=-1)
rf_grid = GridSearchCV(rf_pipe, rf_params, cv=3, scoring="f1", n_jobs=-1)

print("🔄 Training Logistic Regression...")
log_grid.fit(X_train, y_train)
print(f"☑ Logistic best params: {log_grid.best_params_}, best F1:
{log_grid.best_score_:.4f}")

print("\n🔄 Training Random Forest...")
rf_grid.fit(X_train, y_train)
print(f"☑ RF best params: {rf_grid.best_params_}, best F1:
{rf_grid.best_score_:.4f}")

best_models = {
    "Logistic Regression": log_grid.best_estimator_,
    "Random Forest": rf_grid.best_estimator_
}

if log_grid.best_score_ >= rf_grid.best_score_:
    best_model_name = "Logistic Regression"
    best_model = log_grid.best_estimator_
else:
    best_model_name = "Random Forest"
    best_model = rf_grid.best_estimator_
```

```
print(f"\n🏆 Best model selected: {best_model_name} (use in Step 7)")
```

```
    Training Logistic Regression...
    Logistic best params: {'classifier_C': 0.1}, best F1: 0.5959

    Training Random Forest...
    RF best params: {'classifier_max_depth': None,
'classifier_n_estimators': 100}, best F1: 0.5492

    Best model selected: Logistic Regression (use in Step 7)
```

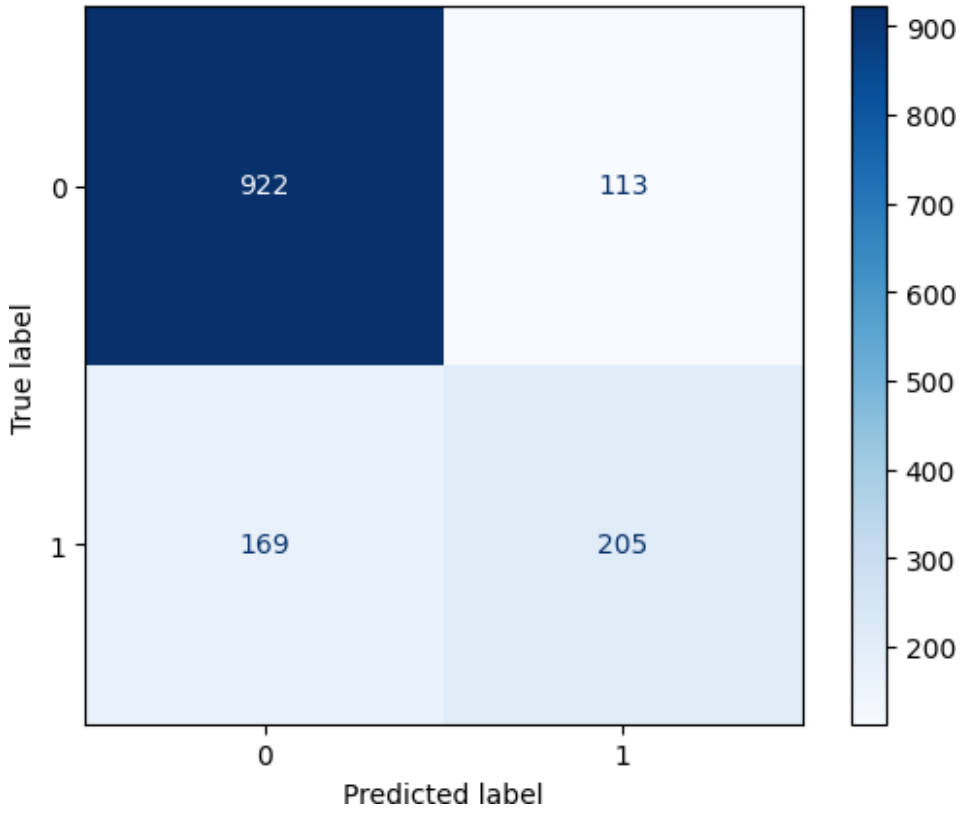
Model Evaluation

```
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix, ConfusionMatrixDisplay

# 7) Evaluate the final chosen model (Logistic Regression) on the test set
print("🔍 Evaluating Logistic Regression on test set...")

# Predict on test set
y_pred = best_model.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)

print("\n📊 Model Performance on Test Set:")
print(f"☑ Accuracy:  {accuracy:.4f}")
print(f"☑ Precision: {precision:.4f}")
print(f"☑ Recall:    {recall:.4f}")
print(f"☑ F1 Score:  {f1:.4f}")

# Display confusion matrix
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=best_model.classes_)
disp.plot(cmap="Blues")
```

```
    Evaluating Logistic Regression on test set...

    Model Performance on Test Set:
    Accuracy:  0.7999
    Precision: 0.6447
```

```
    Recall:    0.5481
    F1 Score:  0.5925
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at
0x79cd5d844380>
```



Predictions on new data

```python
import pandas as pd

# --- 1. Create sample input data with the SAME columns used in training ---
# (We include all columns from training except the target column 'Churn')
sample_data = pd.DataFrame([{
    "customerID": "TEST123",
    "gender": "Female",
    "SeniorCitizen": 0,
    "Partner": "Yes",
    "Dependents": "No",
    "tenure": 12,
    "PhoneService": "Yes",
    "MultipleLines": "No",
    "InternetService": "Fiber optic",
    "OnlineSecurity": "No",
    "OnlineBackup": "Yes",
    "DeviceProtection": "No",
```

```
    "TechSupport": "Yes",
    "StreamingTV": "Yes",
    "StreamingMovies": "No",
    "Contract": "Month-to-month",
    "PaperlessBilling": "Yes",
    "PaymentMethod": "Electronic check",
    "MonthlyCharges": 70.35,
    "TotalCharges": 845.5
}])

# --- 2. Reordering columns to exactly match training data ---
# This ensures that ColumTransformer sees columns in the same order
sample_data = sample_data[X_train.columns]

# --- 3. Made prediction using the trained best_model pipeline ---
sample_prediction = best_model.predict(sample_data)[0]   # 0 or 1
sample_probability = best_model.predict_proba(sample_data)[0][1]  #
probability of churn

# --- 4. Display result nicely ---
if sample_prediction == 1:
    print(f"☑ Prediction: Customer is LIKELY to CHURN (Churn Probability:
{sample_probability:.2%})")
else:
    print(f"☑ Prediction: Customer is NOT likely to churn (Churn
Probability: {sample_probability:.2%})")
```

```
   Prediction: Customer is LIKELY to CHURN (Churn Probability: 52.88%)
```

```
#video explanation link
https://drive.google.com/file/d/1il7hqRUCp7kxTXzgjajZd7nqxLU3j79L/view?usp=dr
ivesdk
```