**LEVEL 2 INTERMEDIATE**

**Task 1: Logistic Regression for Binary Classification**

Description: Implement a logistic regression model to predict binary outcomes (e.g., whether a customer will churn).

Load and preprocess the dataset.

Train a logistic regression model using scikit-learn.

Interpret model coefficients and the odds ratio.

Evaluate the model using metrics such as accuracy, precision, recall, and the ROC curve.

Tools: Python, pandas, scikit-learn, matplotlib.

**Code**

**Step 1**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import (

accuracy\_score, precision\_score, recall\_score, roc\_auc\_score,

confusion\_matrix, roc\_curve

)

# Load dataset from Desktop (example path)

df = pd.read\_csv(r”C:\Users\USERNAME\Desktop\churn-bigml-20.csv”)

print(df.shape)

df.head()

**Explain**

* We import the main Python libraries for data handling (pandas, numpy), plotting (matplotlib, seaborn), and machine learning (scikit-learn).
* Load the CSV file from Desktop into a dataframe (df).
* df.shape → tells number of rows and columns.
* df.head() → shows the first 5 rows to preview the dataset.

**Step 2**

# Check for missing values

print(df.isnull().sum())

# Encode categorical columns into numeric

for col in df.select\_dtypes(include=['object']).columns:

df[col] = LabelEncoder().fit\_transform(df[col].astype(str))

# Define features (X) and target (y)

X = df.drop("Churn", axis=1) # predictors

y = df["Churn"] # target (0 = No churn, 1 = Churn)

# Split dataset into train and test (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42, stratify=y

)

# Scale features for better model performance

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

**Explain**

* **Missing values** are checked.
* Many datasets have categorical features (like “Gender: Male/Female”). LabelEncoder converts them to numbers (Male=0, Female=1).
* Split dataset into:
* **X** → independent features (age, contract, usage, etc.)
* **y** → dependent variable (Churn) → the thing we want to predict.
* Use **train\_test\_split**:
* 80% → training data
* 20% → testing data
* stratify=y ensures both classes (churn/no churn) are balanced.
* **StandardScaler**: Logistic regression works better when features are scaled (so that large values like “tenure=500” don’t dominate small values like “age=20”).

**Step 3**

# Initialize and train model

log\_reg = LogisticRegression(max\_iter=1000)

log\_reg.fit(X\_train\_scaled, y\_train)

# Predictions

y\_pred = log\_reg.predict(X\_test\_scaled)

y\_prob = log\_reg.predict\_proba(X\_test\_scaled)[:, 1] # probability for churn=1

**Explain**

* We create a **Logistic Regression model**.
* max\_iter=1000 ensures the algorithm has enough iterations to converge.
* fit() trains the model using training data.
* predict() gives **0 (No Churn)** or **1 (Churn)** predictions for test data.
* predict\_proba() gives probabilities (e.g., 0.85 → 85% chance of churn).

**Step 4**

# Coefficients and odds ratios

coefficients = pd.DataFrame({

"Feature": X.columns,

"Coefficient": log\_reg.coef\_[0],

"Odds Ratio": np.exp(log\_reg.coef\_[0])

}).sort\_values(by="Odds Ratio", ascending=False)

Coefficients

**Explain**

* Logistic regression gives **coefficients** for each feature → tells how much it influences churn.
* np.exp(coefficient) → converts coefficient into **Odds Ratio**.
* Odds Ratio > 1 → increases chance of churn.
* Odds Ratio < 1 → decreases chance of churn.
* Example: If “Contract type” has an odds ratio of 3 → customers with that contract type are **3 times more likely** to churn.

**Step 5**

# Metrics

acc = accuracy\_score(y\_test, y\_pred)

prec = precision\_score(y\_test, y\_pred)

rec = recall\_score(y\_test, y\_pred)

roc\_auc = roc\_auc\_score(y\_test, y\_prob)

print(f"Accuracy: {acc:.2f}")

print(f"Precision: {prec:.2f}")

print(f"Recall: {rec:.2f}")

print(f"ROC-AUC: {roc\_auc:.2f}")

# Confusion Matrix

cm = confusion\_matrix(y\_test, y\_pred)

sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",

xticklabels=["No Churn", "Churn"],

yticklabels=["No Churn", "Churn"])

plt.xlabel("Predicted")

plt.ylabel("Actual")

plt.title("Confusion Matrix")

plt.show()

# ROC Curve

fpr, tpr, thresholds = roc\_curve(y\_test, y\_prob)

plt.plot(fpr, tpr, label=f"ROC Curve (AUC={roc\_auc:.2f})")

plt.plot([0,1], [0,1], "k--")

plt.xlabel("False Positive Rate")

plt.ylabel("True Positive Rate")

plt.title("ROC Curve")

plt.legend()

plt.show()

**Explain**

* **Accuracy** → % of correct predictions.
* **Precision** → out of all predicted churns, how many were actually churns.
* **Recall (Sensitivity)** → out of all actual churns, how many did we catch.
* **ROC-AUC** → overall model ability to distinguish churn vs no churn.
* **Confusion Matrix** → table showing counts of:
* True Positives (correct churns)
* True Negatives (correct non-churns)
* False Positives (predicted churn but didn’t churn)
* False Negatives (missed churns)
* **ROC Curve** → graphical representation of model performance across thresholds.