Task no: 2

Code Step-by-Step Explanation:

Step 1: Install and Import Libraries:

```python
!pip install -q scikit-learn pandas numpy joblib matplotlib seaborn
```

This line ensures all required libraries are installed, especially when using Google Colab or Jupyter Notebook.

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
```

Here we import all necessary libraries:

- **pandas/numpy** for data handling,
- **sklearn** for preprocessing, modeling, and evaluation,
- **joblib** for saving/loading the pipeline,
- **matplotlib/seaborn** for visualization.

Step 2: Load Dataset:

```
url = "https://raw.githubusercontent.com/IBM/telco-customer-churn-on-icp4d/master/data/Telco-Custo
df = pd.read_csv(url)
print(df.head())
```

We load the Telco Customer Churn dataset directly from GitHub into a Pandas DataFrame.
df.head() shows the first five rows so we can understand the structure.

Step 3: Data Cleaning:

```
df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
df.dropna(inplace=True)
df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})
df.drop('customerID', axis=1, inplace=True)
```

TotalCharges: originally stored as strings → converted into numeric values. Invalid entries become NaN.

dropna(): removes rows with missing values.

Churn: target column converted to numbers → Yes = 1, No = 0.

customerID: dropped because it doesn't help in prediction.

Step 4: Train/Test Split:

```
X = df.drop('Churn', axis=1)
y = df['Churn']
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

We separate features (X) and target (y).
Dataset is split into 80% training and 20% testing.stratify=y ensures both sets keep the same proportion of churn vs non-churn customers.

Step 5: Feature Identification:

```
numeric_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
categorical_features = [col for col in X_train.columns if col not in numeric_features]
```

Numeric features: continuous values.

Categorical features: all other columns.

This separation is needed for different preprocessing steps.

Step 6: Preprocessing Pipelines:

1.Numeric Pipeline:

```python
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

Missing values are replaced with the **median**.

Features are scaled (mean = 0, std = 1).

2. Categorical Pipeline:

```python
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

Missing values are replaced with the word "missing".

One-hot encoding converts categories into binary columns.

3. Combine Both:

```python
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
```

This applies the numeric pipeline to numeric columns and the categorical pipeline to categorical columns.

Step 7: Build Model Pipelines:

```python
logreg_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression(random_state=42, max_iter=1000))
])

rf_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier(random_state=42))
])
```

Two pipelines are created:

1. Logistic Regression
2. Random Forest

Each pipeline includes preprocessing + classifier, so the whole process is automated.

Step 8: Train Logistic Regression:

```python
logreg_pipeline.fit(X_train, y_train)
y_pred_logreg = logreg_pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred_logreg))
print(classification_report(y_test, y_pred_logreg))
```

1.**Trains Model** - Teaches Logistic Regression using training data

2.**Makes Predictions** - Predicts churn for test customers

3.**Checks Accuracy** - Calculates percentage of correct predictions

4.**Shows Report** - Displays precision, recall, F1-score for each class

Step 9: Train Random Forest:

```python
rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)
print("Accuracy:", accuracy_score(y_test, y_pred_rf))
print(classification_report(y_test, y_pred_rf))
```

1. **Model Training** - Trains Random Forest on training data
2. **Prediction** - Makes predictions on test data
3. **Evaluation** - Calculates accuracy and performance metrics
4. **Results** - Shows ~79% accuracy with detailed classification report

Step 10: Hyperparameter Tuning:

```python
param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5]
}

grid_search = GridSearchCV(
    rf_pipeline,
    param_grid,
    cv=3,
    scoring='accuracy',
    n_jobs=-1,
    verbose=1
)
grid_search.fit(X_train, y_train)
```

We perform **GridSearchCV** to test different hyperparameters for Random Forest:

- Number of trees (**n_estimators**),
- Maximum depth of trees (**max_depth**),
- Minimum samples to split a node (**min_samples_split**).

GridSearchCV runs cross-validation and selects the best parameters.

Step 11: Evaluate Tuned Model:

```python
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print("Final Accuracy:", accuracy_score(y_test, y_pred_best))
print(classification_report(y_test, y_pred_best))
```

We use the best tuned model and evaluate it on the test set.This gives the final performance metrics.

Step 12: Save the Final Pipeline:

```python
joblib.dump(best_model, 'telco_churn_pipeline.joblib')
```

The final model, along with preprocessing steps, is saved into a file.
This makes the pipeline reusable in production without retraining.

Step 13: Confusion Matrix Visualization:

```python
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(6, 5))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix - Tuned Random Forest')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
```

This code creates a Confusion Matrix visualization:

1. Calculates confusion matrix values (TP, TN, FP, FN)
2. Creates a heatmap visualization using Seaborn
3. Shows actual values on y-axis, predictions on x-axis
4. Displays counts in each cell with blue color gradient
5. Labels axes and adds title for clarity

Purpose: Visually shows how many predictions were correct/incorrect across different classes.

Step 14: Feature Importance:

```
feature_importances = best_model.named_steps['classifier'].feature_importances_

feature_names = numeric_features + list(
    best_model.named_steps['preprocessor']
    .named_transformers_['cat']
    .named_steps['onehot']
    .get_feature_names_out(categorical_features)
)

fi_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importances
}).sort_values('importance', ascending=False).head(10)

plt.figure(figsize=(8, 5))
sns.barplot(x='importance', y='feature', data=fi_df, palette='viridis')
plt.title('Top 10 Feature Importances')
plt.tight_layout()
plt.show()
```

We extract feature importances from the Random Forest model.

We match importance values with feature names (numeric + one-hot encoded categorical).

A bar chart shows the top 10 most important features that influence churn predictions**.**

Final Summary:

1. The pipeline loads and cleans the ***Telco Churn*** dataset.

2. It handles missing values, scales numeric features, and encodes categorical features.

3.**Two models** were tested***: Logistic Regression and Random Forest***.

4. **Random Forest** was tuned with ***GridSearchCV*** to improve accuracy.

5.The best model was saved **with jobl**ib for production use.

6. Performance was evaluated using accuracy, precision, recall***, F1-score***, and ***confusion matrix***.

7. Finally, feature importance **visualization** highlighted which factors most affect customer churn.