Task no: 2

Code:

```
# 1. Install Required Libraries (only for Colab / Jupyter)

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

# 2. Import Libraries

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

# 3. Load Dataset

print("Loading dataset...")

url = "https://raw.githubusercontent.com/IBM/telco-customer-churn-on-icp4d/master/data/Telco-Customer-Churn.csv"

df = pd.read_csv(url)

print (f"Dataset loaded successfully! Shape: {df.shape}")

# Preview raw dataset

print("\nPreview of Raw Dataset (first 5 rows):")

print(df.head())

# 4. Data Cleaning
```

```
print("\nCleaning data...")

df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')  # Convert non-numeric values to NaN

df.dropna(inplace=True)  # Drop missing values

df['Churn'] = df['Churn'].map({'Yes': 1, 'No': 0})  # Encode churn as 1 (Yes) and 0 (No)

df.drop('customerID', axis=1, inplace=True)  # Remove customerID column (not useful for prediction)

print(f"Data cleaned! New shape: {df.shape}")

# 5. Split Data (Train/Test split)

X = df.drop('Churn', axis=1)  # Features

y = df['Churn']              # Target

X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42, stratify=y

)  # Stratify ensures balanced churn distribution

print(f"\nTraining set: {X_train.shape}, Testing set: {X_test.shape}")# 6. Identify Feature Types

numeric_features = ['tenure', 'MonthlyCharges', 'TotalCharges']  # Continuous values

categorical_features = [col for col in X_train.columns if col not in numeric_features]  # All others

print("\nFeature Types:")

print("Numeric:", numeric_features)

print("Categorical:", categorical_features)

# 7. Preprocessing Pipelines

# Numeric: handle missing values → scale

numeric_transformer = Pipeline(steps=[

    ('imputer', SimpleImputer(strategy='median')),

    ('scaler', StandardScaler())

])
# Categorical: replace missing with "missing" → one-hot encode

categorical_transformer = Pipeline(steps=[

    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
```

```
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
# Combine both numeric + categorical preprocessing
preprocessor = ColumnTransformer(transformers=[
    ('num', numeric_transformer, numeric_features),
    ('cat', categorical_transformer, categorical_features)
])
# 8. Define Models (2 Pipelines: Logistic Regression + Random Forest)
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))
])
# 9. Train & Evaluate Logistic Regression
print("\nTraining Logistic Regression...")
logreg_pipeline.fit(X_train, y_train)
y_pred_logreg = logreg_pipeline.predict(X_test)
print("Logistic Regression Results:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_logreg):.4f}")
print(classification_report(y_test, y_pred_logreg))
# 10. Train & Evaluate Random Forest
print("\nTraining Random Forest...")
rf_pipeline.fit(X_train, y_train)
y_pred_rf = rf_pipeline.predict(X_test)
print("Random Forest Results:")
print(f"Accuracy: {accuracy_score(y_test, y_pred_rf):.4f}")
```

```python
print(classification_report(y_test, y_pred_rf))
# 11. Hyperparameter Tuning for Random Forest
print("\nStarting Hyperparameter Tuning...")
param_grid = {
    'classifier__n_estimators': [100, 200],      # Number of trees
    'classifier__max_depth': [None, 10, 20],     # Depth of tree
    'classifier__min_samples_split': [2, 5]      # Split criteria
}grid_search = GridSearchCV(
    rf_pipeline,
    param_grid,
    cv=3,    # 3-fold cross-validation
    scoring='accuracy',
    n_jobs=-1,    # Use all CPUs
    verbose=1
)
grid_search.fit(X_train, y_train)
# Print best hyperparameters
print(f"\nBest Parameters: {grid_search.best_params_}")
print(f"Best CV Accuracy: {grid_search.best_score_:.4f}")
# 12. Evaluate Best Model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
print("\nTuned Random Forest Results:")
print(f"Final Accuracy: {accuracy_score(y_test, y_pred_best):.4f}")
print(classification_report(y_test, y_pred_best))
# 13. Save Final Pipeline
joblib.dump(best_model, 'telco_churn_pipeline.joblib')
print("\nPipeline saved as 'telco_churn_pipeline.joblib'")
# 14. Visualization - Confusion Matrix
```

```python
print("\nCreating Confusion Matrix...")
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()
# 15. Visualization - Top 10 Feature Importances
print("\nCreating Top 10 Feature Importance Plot...")
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)
)
# Select Top 10 most important features
fi_df = pd.DataFrame({
    'feature': feature_names,
    'importance': feature_importances
}).sort_values('importance', ascending=False).head(10)
# Plot bar chart
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()
print("\nTask Complete! Full ML Pipeline built, tuned, evaluated, saved, and visualized.")
```