Task 4: Churn Prediction Model Description: Choose suitable machine learning algorithms (e.g., logistic regression, decision trees) for churn prediction. Split data into training and testing sets, train and evaluate multiple models using metrics like accuracy, precision, recall, and F1-score. Perform feature selection and hyperparameter tuning for optimal performance.

Skills : Machine learning algorithms

Model training and evaluation,

Feature selection, hyperparameter tuning

Understanding of classification metrics.

In [142…

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
```

In [143…

```
df=pd.read_csv('C://Users//ALWAYSRAMESH//Downloads//Telco_Customer_Churn_Dataset  (
```

In [144…

```
df.head()
```

Out[144…

	customerID	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	Multipl
0	7590-VHVEG	Female	0	Yes	No	1	No	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	No
4	9237-HQITU	Female	0	No	No	2	Yes	

5 rows × 21 columns

In [145…
```
# Replace missing values with mode (categorical) or mean (numerical)
imputer = SimpleImputer(strategy='most_frequent')
df.fillna(df.mode().iloc[0], inplace=True)
```

In [146…
```
label_enc = LabelEncoder()

# Apply Label Encoding to categorical columns
for col in df.select_dtypes(include=['object']).columns:
    df[col] = label_enc.fit_transform(df[col])
```

In [147…
```
# Define Features and Target
X = df.drop(columns=['Churn'])  # Assuming 'Churn' is the target column
y = df['Churn']
```

In [148…
```
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta
```

In [149…
```
# Standardization
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

In [150…
```
# Define models
models = {
    "Logistic Regression": LogisticRegression(),
    "Decision Tree": DecisionTreeClassifier(),
    "Random Forest": RandomForestClassifier()
}
```

In [151…
```
# Train & Evaluate Models
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
```

In [152…
```
# Evaluate Model
print(f"\n{name} Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print(classification_report(y_test, y_pred))
```

```
Random Forest Performance:
Accuracy: 0.7927608232789212
Precision: 0.6423611111111112
Recall: 0.4946524064171123
F1 Score: 0.5589123867069486
              precision    recall  f1-score   support

           0       0.83      0.90      0.86      1035
           1       0.64      0.49      0.56       374

    accuracy                           0.79      1409
   macro avg       0.74      0.70      0.71      1409
weighted avg       0.78      0.79      0.78      1409
```

In [153…

```python
# Hyperparameter Grid
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [5, 10, None],
    'min_samples_split': [2, 5, 10]
}
```

In [154…

```python
# Grid Search CV
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=5, scoring='acc
grid_search.fit(X_train, y_train)
```

Out[154…

```
▸ GridSearchCV
▸ best_estimator_: RandomForestClassifier
    ▸ RandomForestClassifier
```

In [155…

```python
# Best Parameters
print("\nBest Parameters for Random Forest:", grid_search.best_params_)
```

```
Best Parameters for Random Forest: {'max_depth': None, 'min_samples_split': 10, 'n_e
stimators': 200}
```

In [156…

```python
# Best Model Performance
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
```

In [157…

```python
print("\nOptimized Model Performance:")
print("Accuracy:", accuracy_score(y_test, y_pred_best))
print("Precision:", precision_score(y_test, y_pred_best))
print("Recall:", recall_score(y_test, y_pred_best))
print("F1 Score:", f1_score(y_test, y_pred_best))
```

```
Optimized Model Performance:
Accuracy: 0.7970191625266146
Precision: 0.6571428571428571
Recall: 0.4919786096256685
F1 Score: 0.5626911314984709
```

In [ ]: