

# ML\_final\_project\_task2

September 30, 2025

```
[ ]: #videolink:https://drive.google.com/file/d/1UNfJxb22Bbqnr-SK7Fw_cgBtXbuorUuT/  
    ↪view?usp=sharing
```

```
[1]: import pandas as pd  
import numpy as np  
from sklearn.model_selection import train_test_split  
from sklearn.pipeline import Pipeline  
from sklearn.impute import SimpleImputer  
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncoder  
from sklearn.compose import ColumnTransformer
```

```
[2]: df = pd.read_csv("Customer_data - customer_data.csv")
```

```
[3]: # Drop customerID (not a feature) and define features (X) and target (y)  
X = df.drop(columns=['customerID', 'Churn'])  
y = df['Churn']
```

```
[4]: # Target Encoding: Convert 'Yes'/'No' in Churn to 1/0  
le = LabelEncoder()  
y_encoded = le.fit_transform(y)
```

```
[5]: # Define feature types  
numerical_features = ['tenure', 'MonthlyCharges', 'TotalCharges']
```

```
[6]: # The remaining are categorical (including SeniorCitizen which is 0/1 but best  
    ↪treated as categorical)  
categorical_features = [col for col in X.columns if col not in  
    ↪numerical_features]
```

```
[7]: # Define preprocessing steps  
numerical_transformer = Pipeline(steps=[  
    ('imputer', SimpleImputer(strategy='median')),  
    ('scaler', StandardScaler()) # Scale numerical features  
)  
  
categorical_transformer = Pipeline(steps=[
```

```
    ('onehot', OneHotEncoder(handle_unknown='ignore', sparse_output=False))
])
```

```
[8]: preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough'
)
```

```
[9]: # --- 2. Data Splitting ---

# Split data into training (80%) and testing (20%) sets
X_train, X_test, y_train, y_test = train_test_split(
    X, y_encoded, test_size=0.2, random_state=42, stratify=y_encoded
)

# Apply preprocessing to training and testing data
X_train_processed = preprocessor.fit_transform(X_train)
X_test_processed = preprocessor.transform(X_test)

print(f"Original feature shape: {X.shape}")
print(f"Processed feature shape (after OHE): {X_train_processed.shape}")
```

Original feature shape: (7043, 19)  
 Processed feature shape (after OHE): (5634, 46)

```
[10]: from sklearn.linear_model import LogisticRegression
    from sklearn.model_selection import GridSearchCV
```

```
[11]: # Define the parameter grid for tuning
    param_grid = {
        'C': [0.01, 0.1, 1, 10], # Inverse of regularization strength
        'class_weight': [None, 'balanced'] # Balanced to handle churn imbalance
    }
```

```
[12]: # Initialize the Logistic Regression model
    log_reg_tuned = LogisticRegression(solver='liblinear', random_state=42,
    ↪max_iter=1000)
```

```
[13]: # Initialize GridSearchCV with F1-score as the metric
    grid_search = GridSearchCV(
        estimator=log_reg_tuned,
        param_grid=param_grid,
        scoring='f1',
        cv=5,
        n_jobs=1
    )
```

```
)
```

```
[14]: # Fit the grid search to the processed training data
grid_search.fit(X_train_processed, y_train)
```

```
[14]: GridSearchCV(cv=5,
                  estimator=LogisticRegression(max_iter=1000, random_state=42,
                                                solver='liblinear'),
                  n_jobs=1,
                  param_grid={'C': [0.01, 0.1, 1, 10],
                              'class_weight': [None, 'balanced']},
                  scoring='f1')
```

```
[15]: # Get the best model
best_log_reg = grid_search.best_estimator_

print("\n--- Best Model Hyperparameters ---")
print(grid_search.best_params_)
```

```
--- Best Model Hyperparameters ---
{'C': 1, 'class_weight': 'balanced'}
```

```
[16]: from sklearn.metrics import classification_report, confusion_matrix,
      ↪ accuracy_score
```

```
[17]: # Make predictions with the best model on the test set
y_pred_tuned = best_log_reg.predict(X_test_processed)

print("\n--- Final Tuned Logistic Regression Model Evaluation ---")
```

```
--- Final Tuned Logistic Regression Model Evaluation ---
```

```
[18]: # Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred_tuned, target_names=['No Churn',
      ↪ 'Churn']))
```

Classification Report:

	precision	recall	f1-score	support
No Churn	0.90	0.72	0.80	1035
Churn	0.50	0.78	0.61	374
accuracy			0.74	1409
macro avg	0.70	0.75	0.71	1409

weighted avg          0.80          0.74          0.75          1409

```
[19]: # Confusion Matrix
print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_pred_tuned))
```

```
Confusion Matrix:
[[747 288]
 [ 81 293]]
```

```
[20]: # Final Accuracy
print(f"\nAccuracy: {accuracy_score(y_test, y_pred_tuned):.4f}")
```

```
Accuracy: 0.7381
```

```
[21]: # Get the feature names for easy mapping
feature_names_out = numerical_features + list(preprocessor.
    ↪named_transformers_['cat']['onehot'].
    ↪get_feature_names_out(categorical_features))
```

```
[22]: # Get the coefficients from the best model
coefficients = best_log_reg.coef_[0]
feature_importance = pd.Series(coefficients, index=feature_names_out)
```

```
[23]: # Display top 10 features contributing to CHURN (positive coefficients)
top_positive = feature_importance.sort_values(ascending=False).head(10)
print("\n--- Top 10 Features Driving CHURN (Positive Coefficients) ---")
print(top_positive)
```

```
--- Top 10 Features Driving CHURN (Positive Coefficients) ---
InternetService_Fiber optic      0.710907
Contract_Month-to-month          0.660077
TotalCharges                      0.473614
StreamingMovies_Yes              0.278028
StreamingTV_Yes                  0.264838
PaymentMethod_Electronic check  0.240885
OnlineSecurity_No                0.200246
TechSupport_No                   0.166391
MultipleLines_Yes                0.135283
DeviceProtection_Yes             0.075981
dtype: float64
```

```
[24]: # Display top 10 features preventing CHURN (negative coefficients)
top_negative = feature_importance.sort_values(ascending=True).head(10)
print("\n--- Top 10 Features Preventing CHURN (Negative Coefficients) ---")
```

```
print(top_negative)
```

```
--- Top 10 Features Preventing CHURN (Negative Coefficients) ---
```

tenure	-1.140572
Contract_Two year	-0.776303
MonthlyCharges	-0.676408
InternetService_DSL	-0.622606
OnlineSecurity_No internet service	-0.274965
OnlineBackup_No internet service	-0.274965
InternetService_No	-0.274965
DeviceProtection_No internet service	-0.274965
StreamingTV_No internet service	-0.274965
StreamingMovies_No internet service	-0.274965

dtype: float64

```
[ ]:
```