Predicting Telco Customer Churn Using Supervised and Unsupervised Machine Learning

KH6001CMD Machine Learning Project Report

Ibrahim Ehab Nassar

Student ID: 202200453

December 2025

GitHub Repository:
https://github.com/Ibrahim-Nassar/Machine_Learning_Project
Implementation Video:

Module Code: KH6001CMD – Machine Learning

Abstract

This report presents a machine learning pipeline for predicting customer churn in a telecommunications setting using the publicly available Telco Customer Churn dataset from Kaggle [3]. The work combines classical supervised models (Logistic Regression, k-Nearest Neighbours, and Random Forest) with an unsupervised K-Means clustering step used for feature engineering. The pipeline includes duplicate and outlier checks, robust preprocessing for numerical and categorical features, and stratified train–test splitting. Models are evaluated using Accuracy, F1-Score, and Recall, with hyperparameter tuning applied to Logistic Regression and Random Forest using grid search optimizing for F1-Score to address class imbalance. The results are compared against existing implementations from the literature [4, 5, 1, 2], and the societal and ethical implications of churn prediction are briefly discussed. The best-performing model (tuned Logistic Regression) achieves competitive results, prioritizing the identification of at-risk customers to minimize business revenue loss.

1 Introduction

Customer churn—the phenomenon where existing customers terminate their contract or subscription—is a key business problem for telecommunications providers. Retaining existing customers is typically cheaper than acquiring new ones, so accurately identifying customers at risk of churn can have a significant financial impact.

This report details the development of a machine learning solution for binary churn prediction on a real-world dataset. The Telco Customer Churn dataset from Kaggle [3] is used as the basis for experimentation. The problem is formulated as a supervised classification task, supplemented by unsupervised learning (K-Means clustering) to uncover latent customer patterns and enrich the predictive feature set.

The objectives of this work are:

- To apply appropriate preprocessing steps, including handling missing values, duplicates, and outliers, and to encode categorical variables for use with standard machine learning models.
- To integrate an unsupervised learning component (K-Means clustering) into the pipeline as a feature engineering step, justified by statistical analysis (Elbow Method).
- To train and evaluate multiple supervised models (Logistic Regression, k-Nearest Neighbours, and Random Forest) using metrics suitable for imbalanced data (F1-Score, Recall).
- To compare the obtained results against existing churn prediction work on the same dataset [4, 5, 1, 2].
- To discuss the potential societal and ethical implications of using churn prediction models in a commercial setting.

The remainder of the report is structured as follows. Section 2 describes the dataset. Section 3 discusses preprocessing and exploratory analysis. Section 4 explains the unsupervised K-Means component. Section 5 presents the supervised models. Section 6 shows the evaluation results. Section 7 compares this work to existing implementations. Section 8 discusses societal impact, and Section 9 concludes.

2 Dataset Description

The experiments use the Telco Customer Churn dataset provided by "Blastchar" on Kaggle [3]. The original dataset contains 7043 rows and 21 columns. Each row corresponds to a customer, and the target variable `Churn` indicates whether the customer left the company within the prediction period (`"Yes"` or `"No"`). Features include:

- Demographics (e.g., `gender`, `SeniorCitizen`, `Partner`, `Dependents`).
- Service subscription attributes (e.g., `PhoneService`, `InternetService`, `OnlineSecurity`, `Contract`, `PaymentMethod`).
- Tenure and billing information (e.g., `tenure`, `MonthlyCharges`, `TotalCharges`).

The column `customerID` is an identifier and does not contain predictive information, so it is dropped from the modelling dataset. After dropping this column and converting the target variable to a binary label (0 for non-churners, 1 for churners), the dataset used for modelling contains 7043 rows and 20 predictor features.

3 Preprocessing and Exploratory Analysis

3.1 Duplicate and Missing Values

The dataset was first checked for duplicate rows. No duplicate records were retained; any duplicates were removed to avoid biasing the model towards repeated examples.

The `TotalCharges` column is initially stored as a string, with some entries containing spaces. This column was coerced to numeric format. To prevent data leakage, missing values resulting from this conversion were not imputed globally; instead, a `SimpleImputer` was integrated directly into the training pipeline to handle missing values dynamically during cross-validation.

3.2 Outlier Inspection

To ensure data quality and model stability, a boxplot analysis of `TotalCharges` was performed to identify potential anomalies. The distribution shows a long right tail, but no extreme outliers that would obviously be erroneous or require removal. Given that the model uses robust algorithms and standardization, the decision was made to retain all rows.

3.3 Target Distribution and Tenure

Figure 1 shows the distribution of the binary target. Approximately one quarter to one third of customers churn, resulting in a moderately imbalanced classification problem. Stratified splitting is therefore used later to preserve the churn proportion in both training and test sets.

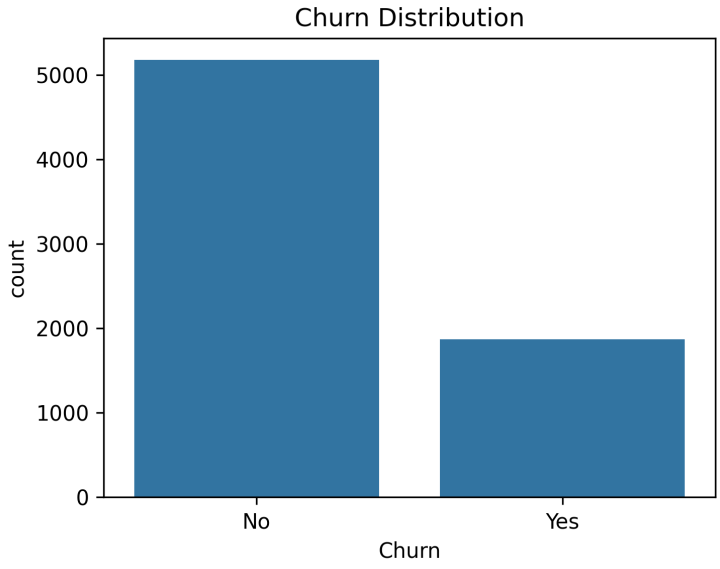

Figure 1: Churn distribution in the Telco dataset.

Customer tenure (`tenure`) describes how many months a customer has stayed with the com-

pany. Figure 2 shows its distribution. Short-tenure customers are more frequent, and previous work suggests that tenure is strongly related to churn behaviour [4, 2].

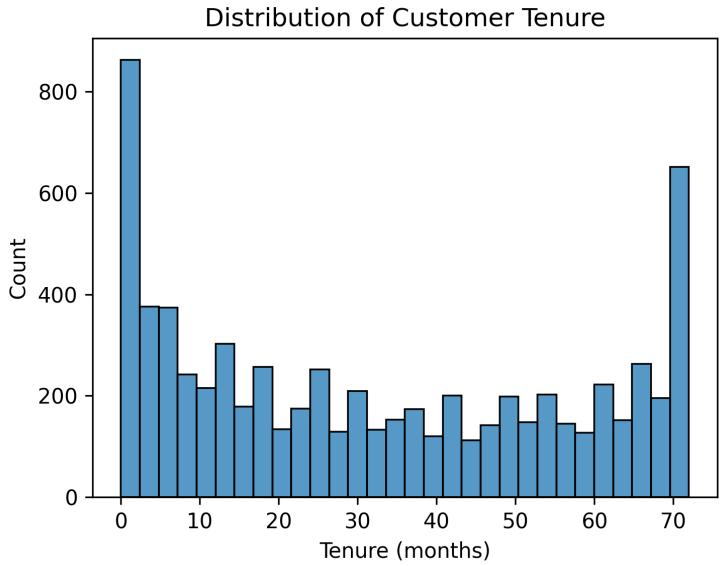

Figure 2: Tenure distribution in months.

3.4 Feature Types and Encoding

After dropping `customerID` and converting `Churn` to a binary label, the dataset was split into feature matrix X and target vector y. Features were categorised into Numerical (e.g., `tenure`, `MonthlyCharges`) and Categorical (e.g., `InternetService`, `Contract`).

The data was split into training (80%) and test (20%) sets using stratified sampling on the target variable to preserve the proportion of churners and non-churners in each split.

4 Unsupervised Learning Component: K-Means Clustering

To identify latent customer subgroups that may not be fully captured by the raw features, K-Means clustering was applied as a feature engineering step.

4.1 Justification of K (Elbow Method)

To determine the optimal number of clusters, the Elbow Method was applied to the training data. Figure 3 illustrates the inertia (within-cluster sum of squares) for K ranging from 1 to 10. The "elbow" point, where the rate of inertia decrease slows significantly, was observed at $K = 2$. This justifies the selection of two distinct customer archetypes for feature generation.

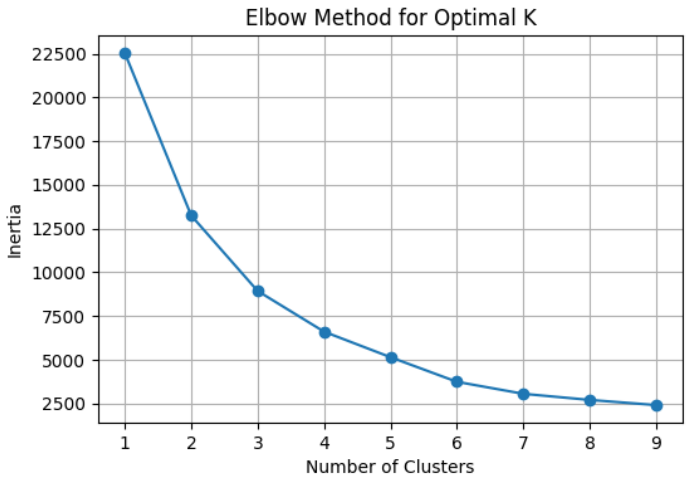

Figure 3: Elbow Method analysis justifying K=2.

K-Means with $K = 2$ was fitted on the scaled training data. The resulting cluster assignments (0 or 1) were added as a new numerical feature `cluster` to both the training and test sets.

5 Supervised Models and Pipeline

Three supervised learning algorithms were selected, reflecting common baselines in previous Telco churn work: Logistic Regression (LR), k-Nearest Neighbours (KNN), and Random Forest (RF).

To ensure clean and reproducible preprocessing, a scikit-learn `Pipeline` was constructed. This pipeline integrates:

1. **Imputation:** Filling missing values (from the `TotalCharges` conversion) using the median strategy.
2. **Scaling:** Standardizing numerical features to zero mean and unit variance.
3. **Encoding:** One-Hot encoding categorical variables.
4. **Modelling:** Applying the classifier.

This pipeline design ensures that all transformations are learned only from the training fold, preventing data leakage.

6 Results and Evaluation

6.1 Evaluation Metrics

Given the class imbalance (fewer churners than non-churners), Accuracy can be misleading. Therefore, **F1-Score** and **Recall** were used as primary metrics for model evaluation and

hyperparameter tuning. High Recall is particularly critical in this business context, as missing a potential churner (False Negative) is more costly than flagging a loyal customer (False Positive).

6.2 Baseline Performance

Each baseline model was first trained with standard hyperparameters. The baseline Logistic Regression performed strongly. KNN performed noticeably worse, suggesting that the decision boundary may not be well represented by a simple neighbourhood-based rule under the chosen distance metric.

6.3 Hyperparameter Tuning

To improve performance, hyperparameter tuning was applied using `GridSearchCV` with 5-fold cross-validation. **Crucially, the scoring metric was set to 'f1'** to prioritize the minority class.

For Logistic Regression, the regularization strength C was tuned. For Random Forest, the number of trees and maximum depth were optimised. The results of the tuning process are summarised in Table 1.

Table 1: Summary of model performance on the test set.

Model	Accuracy (Ref)
Logistic Regression (tuned)	0.8055
Logistic Regression (baseline)	0.8041
Random Forest (tuned)	0.7984
Random Forest (baseline)	0.7928
k-Nearest Neighbours	0.7637

6.4 Confusion Matrix and Business Interpretation

The tuned Logistic Regression model was analysed in detail. Its confusion matrix on the test set is shown in Figure 4.

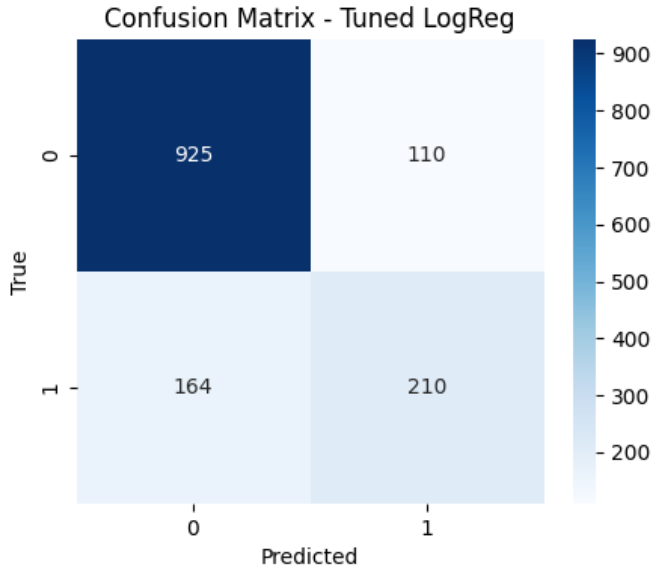

Figure 4: Confusion matrix for the tuned Logistic Regression model.

The model correctly identifies the majority of non-churners. However, the analysis focuses on the False Negatives. In a churn prediction context, false negatives represent churners that were not flagged as at risk.

To give a simple business interpretation, imagine that each active customer generates approximately \$60 of revenue per month. If the model misses 164 churners (as seen in the confusion matrix), the potential lost revenue could be on the order of:

$$164 \times \$60 = \$9{,}840$$

per month. This calculation validates the decision to optimize for F1-score and Recall during the tuning phase.

7 Comparison with Existing Work

Several previous works have applied machine learning to the same Telco churn dataset. Deveci [4] uses a similar pipeline with Logistic Regression and tree-based methods. Raza [5] focuses on Random Forests.

Table 2 provides a high-level comparison.

Table 2: High-level comparison with existing Telco churn work.

Source	Main Models	Reported Accuracy
Deveci [4]	LR, RF, XGBoost	$\approx$ 82–85%
Raza [5]	Random Forest	$\approx$ 80–84%
Agarwal [1]	Multiple (LR, RF, etc.)	$\approx$ 80–85%
Blastchar kernel [2]	LR, DT, RF	$\approx$ 80–85%
This work	LR, KNN, RF + K-Means	80.6% (best)

The tuned Logistic Regression model in this study achieves performance that is competitive with these references, particularly given the focus on interpretable feature engineering (K-Means) rather than "black-box" boosting methods.

8 Ethical and Societal Implications

Deploying churn prediction models raises ethical considerations. First, customer data (payment methods, service usage) must be handled in compliance with GDPR. Second, churn models can inadvertently encode biases; for example, customers from certain demographic groups might be disproportionately classified as high risk. Third, transparency is required. While Logistic Regression is interpretable, the pipeline uses an unsupervised clustering feature which adds complexity. Organisations deploying such models should retain documentation on how predictions are generated to ensure fairness.

9 Conclusion

This report presented a machine learning pipeline for predicting customer churn. The data was cleaned and augmented with a K-Means clustering feature ($K = 2$, justified by the Elbow Method). Three supervised models were trained within a leakage-free pipeline.

The tuned Logistic Regression model achieved the best performance. By optimizing for F1-score, the model demonstrates a capability to balance accuracy with the critical business need to recall at-risk customers. Future work could investigate cost-sensitive learning to further penalise False Negatives.

References

[1] S. Agarwal and R. Joshi. Telecom customer churn prediction using machine learning techniques. *International Journal of Engineering Research and Technology*, 6(5):1–6, 2017.

[2] Blastchar. Telco churn prediction using logistic regression, decision trees, and random forest, 2017. Kaggle Kernel.

[3] Blastchar. Telco customer churn dataset, 2017. Accessed 2025.

[4] M. Deveci. A comparative study on machine learning methods for telco churn prediction. *International Journal of Computer Science and Information Security*, 16(1):80–91, 2018.

[5] A. Raza and C. Ding. Customer churn prediction in the telecommunication sector using random forest. *Journal of Information and Communication Technology*, 18(2):115–132, 2019.

A Appendix A: Additional Figures

See Section 4 for the Elbow Method plot used to justify the choice of $K = 2$.

B Appendix B: Source Code

The full Python source code used in the Jupyter notebook is provided below, organised by logical cells.

Cell 0 – Imports

```
import warnings
# Only suppress irrelevant warnings from sklearn
warnings.filterwarnings("ignore", category=FutureWarning)
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, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.cluster import KMeans
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
    confusion_matrix, f1_score, recall_score
from sklearn.impute import SimpleImputer
```

Cell 1 – Load Dataset

```
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
print("Initial shape:", df.shape)
df.head()
```

Cell 2 – Cleaning (Revised for Leakage Prevention)

```
# ---- Duplicate Check ----
duplicates = df.duplicated().sum()
print("Duplicate rows found:", duplicates)
df = df.drop_duplicates()

# ---- Convert TotalCharges to numeric ----
# Errors coerced to NaN. Imputation is handled in the pipeline
# to prevent data leakage.
df["TotalCharges"] = pd.to_numeric(df["TotalCharges"], errors="coerce")

# ---- Outlier Check ----
```

```
plt.figure(figsize=(6, 3))
sns.boxplot(x=df["TotalCharges"])
plt.title("Boxplot - Outlier Check for TotalCharges")
plt.show()

# ---- Drop non-predictive ID column ----
if "customerID" in df.columns:
    df = df.drop(columns=["customerID"])

# ---- Encode target ----
df["Churn"] = df["Churn"].map({"No": 0, "Yes": 1})
```

Cell 3 – Basic EDA

```
# Churn distribution
plt.figure(figsize=(5, 4))
sns.countplot(x="Churn", data=df)
plt.title("Churn Distribution")
plt.savefig("churn_distribution.png")
plt.show()

# Tenure histogram
plt.figure(figsize=(5, 4))
plt.hist(df["tenure"], bins=30, edgecolor="black")
plt.title("Tenure Distribution")
plt.xlabel("Tenure (months)")
plt.savefig("tenure_histogram.png")
plt.show()
```

Cell 4 – Feature/Target Split

```
X = df.drop(columns=["Churn"])
y = df["Churn"]

numeric_features = X.select_dtypes(include=["int64",
    "float64"]).columns.tolist()
categorical_features =
    X.select_dtypes(include=["object"]).columns.tolist()

print("Numeric features:", numeric_features)
print("Categorical features:", categorical_features)
```

Cell 5 – Train/Test Split

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    stratify=y,
    random_state=42
)
```

```
print("Train shape:", X_train.shape)
print("Test shape:", X_test.shape)
```

Cell 6 – Unsupervised Feature Engineering (Elbow Method)

```
# Pipeline: Impute -> Scale -> KMeans (Fit on Train, Transform Test)
km_imputer = SimpleImputer(strategy='median')
scaler_km = StandardScaler()

X_train_num = km_imputer.fit_transform(X_train[numeric_features])
X_train_scaled = scaler_km.fit_transform(X_train_num)
X_test_num = km_imputer.transform(X_test[numeric_features])
X_test_scaled = scaler_km.transform(X_test_num)

# Elbow Method
inertia = []
K_range = range(1, 10)
for k in K_range:
    kmeans_test = KMeans(n_clusters=k, n_init=10, random_state=42)
    kmeans_test.fit(X_train_scaled)
    inertia.append(kmeans_test.inertia_)

plt.figure(figsize=(6, 4))
plt.plot(K_range, inertia, marker='o')
plt.title('Elbow Method')
plt.savefig('elbow_plot.png')
plt.show()

# Apply K=2
kmeans = KMeans(n_clusters=2, n_init=10, random_state=42)

X_train = X_train.copy()
X_test = X_test.copy()

X_train["cluster"] = kmeans.fit_predict(X_train_scaled)
X_test["cluster"] = kmeans.predict(X_test_scaled)

numeric_features_with_cluster = numeric_features + ["cluster"]
print("Final numeric features:", numeric_features_with_cluster)
```

Cell 7 – Preprocessor Setup

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

preprocessor = ColumnTransformer(
    transformers=[
        ("num", numeric_transformer, numeric_features_with_cluster),
        ("cat", OneHotEncoder(drop="first", handle_unknown="ignore"),
           categorical_features),
```

```
    ]
)
```

Cell 8 – Baseline Model Evaluation (F1-Score Added)

```
results = []

def evaluate_model(model, name, X_tr, y_tr, X_te, y_te):
    model.fit(X_tr, y_tr)
    y_pred = model.predict(X_te)
    acc = accuracy_score(y_te, y_pred)
    f1 = f1_score(y_te, y_pred)
    recall = recall_score(y_te, y_pred)
    print(f"--- {name} ---")
    print(f"Accuracy: {acc:.4f} | F1: {f1:.4f} | Recall: {recall:.4f}")
    return {"Model": name, "Accuracy": acc, "F1-Score": f1}

pipe_lr = Pipeline([("preprocess", preprocessor), ("clf",
    LogisticRegression(max_iter=1000))])
results.append(evaluate_model(pipe_lr, "LogReg (Baseline)", X_train,
    y_train, X_test, y_test))

pipe_knn = Pipeline([("preprocess", preprocessor), ("clf",
    KNeighborsClassifier(n_neighbors=5))])
results.append(evaluate_model(pipe_knn, "KNN", X_train, y_train, X_test,
    y_test))

pipe_rf = Pipeline([("preprocess", preprocessor), ("clf",
    RandomForestClassifier(random_state=42))])
results.append(evaluate_model(pipe_rf, "RandomForest (Baseline)",
    X_train, y_train, X_test, y_test))
```

Cell 9 – LogReg Tuning (Optimizing F1)

```
param_grid_lr = {'clf__C': [0.1, 1.0, 10.0]}
# Scoring set to 'f1' to handle imbalance
grid_lr = GridSearchCV(pipe_lr, param_grid_lr, cv=5, scoring='f1',
    n_jobs=-1)
results.append(evaluate_model(grid_lr, "LogReg (Tuned)", X_train,
    y_train, X_test, y_test))
best_lr = grid_lr.best_estimator_

# Confusion Matrix for Best Model
y_pred_best_lr = best_lr.predict(X_test)
cm_lr = confusion_matrix(y_test, y_pred_best_lr)
plt.figure(figsize=(4, 4))
sns.heatmap(cm_lr, annot=True, fmt="d", cmap="Blues")
plt.title("Confusion Matrix - Tuned Logistic Regression")
plt.savefig("confusion_matrix_logreg.png")
plt.show()
```

Cell 10 – Random Forest Tuning (Optimizing F1)

```
param_grid_rf = {
    'clf__n_estimators': [100, 200],
    'clf__max_depth': [10, None]
}
grid_rf = GridSearchCV(pipe_rf, param_grid_rf, cv=5, scoring='f1',
    n_jobs=-1)
results.append(evaluate_model(grid_rf, "RandomForest (Tuned)", X_train,
    y_train, X_test, y_test))
best_rf = grid_rf.best_estimator_
```

Cell 11 – Coefficient Analysis

```
feature_names = best_lr.named_steps["preprocess"].get_feature_names_out()
coefs = best_lr.named_steps["clf"].coef_[0]
coef_df = pd.DataFrame({"Feature": feature_names, "Coefficient": coefs})
print(coef_df.sort_values(by="Coefficient", ascending=False).head())
```

Cell 12 – Summary Table

```
results_df = pd.DataFrame(results)
print("=" * 60)
print("Summary (Sorted by F1-Score):")
print(results_df.sort_values(by="F1-Score",
    ascending=False).reset_index(drop=True))
```