Customer Churn Prediction Model Documentation

❖ Overview

This project builds a machine learning model to **predict customer churn** using the Telco Customer Churn dataset. The goal is to identify customers who are likely to leave the company and take proactive retention steps.

The process includes:

- Data preprocessing
- Feature engineering
- Class imbalance handling using SMOTE
- Model training using multiple algorithms
- Model evaluation and selection
- Saving the best model and related artifacts

❖ Dataset Information

Source: telco-customer-churn.csv

This dataset contains customer demographic information, services they have signed up for, account information, and whether they have churned.

Key columns:

- customerID: Unique identifier (dropped during preprocessing)
- Churn: Target variable (Yes/No)
- Other categorical and numerical columns related to services and account

---

1. Data Preprocessing

Handled in the load_and_preprocess_data() function.

Steps:

- **Load Dataset**: Reads the CSV file into a DataFrame.
- **Drop Irrelevant Columns**: Removes customerID since it doesn't add predictive value.
- **Convert TotalCharges to numeric**: Invalid entries are coerced to NaN and then removed.

- **Encode Target (Churn)**: Transformed to binary (Yes → 1, No → 0).
- **Label Encoding**: Applies LabelEncoder to binary categorical columns.
- **One-Hot Encoding**: For non-binary categorical features.
- **Standardization**: Uses StandardScaler to scale numerical features.
- **Handle Imbalance**: Applies **SMOTE** (Synthetic Minority Over-sampling Technique) to balance the dataset.
- **Train-Test Split**: Splits the balanced data (80% training, 20% testing).

```python
def load_and_preprocess_data():
    """Load and preprocess the telco customer churn dataset"""
    try:
        # Load data directly from local CSV file
        df = pd.read_csv('telco-customer-churn.csv')
        print(f"📊 Dataset loaded successfully! Shape: {df.shape}")
    except FileNotFoundError:
        print("❌ Error: 'telco-customer-churn.csv' file not found!")
        print("Please make sure the dataset file is in your project directory.")
        raise FileNotFoundError("Dataset file not found")
    except Exception as e:
        print(f"❌ Error loading dataset: {str(e)}")
        raise

    # Store original data for analysis
    original_df = df.copy()

    # Drop unnecessary columns
    df.drop('customerID', axis=1, inplace=True)

    # Convert TotalCharges to numeric and handle missing values
    df['TotalCharges'] = pd.to_numeric(df['TotalCharges'], errors='coerce')
    print(f"🔧 Missing values before cleaning: {df.isnull().sum().sum()}")
    df.dropna(inplace=True)
    print(f"🔧 Missing values after cleaning: {df.isnull().sum().sum()}")

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

    # Store label encoders for later use
    label_encoders = {}
```

---

2. Model Training

Handled in the train_models() function.

Models Trained:

- **Random Forest Classifier**
- **Gradient Boosting Classifier**
- **Logistic Regression**

Each model is evaluated on:

- **Accuracy**
- **AUC Score (Area Under ROC Curve)**

The **best model** is selected based on the **highest AUC score**, ensuring it performs best at distinguishing between churn and non-churn cases.

```
def train_models(data):
    """Train multiple models and return the best one"""
    X_train, y_train = data['X_train'], data['y_train']
    X_test, y_test = data['X_test'], data['y_test']

    print("\n🤖 Training multiple models...")
    print("="*50)

    models = {
        'Random Forest': RandomForestClassifier(
            n_estimators=200,
            max_depth=15,
            min_samples_split=2,
            min_samples_leaf=1,
            max_features='sqrt',
            random_state=42
        ),
        'Gradient Boosting': GradientBoostingClassifier(
            n_estimators=200,
            learning_rate=0.1,
            max_depth=6,
            random_state=42
        ),
        'Logistic Regression': LogisticRegression(
            random_state=42,
            max_iter=1000
        )
    }

    model_results = {}
```

3. Model Evaluation

For each model:

- **Accuracy**: Proportion of correct predictions.
- **AUC Score**: Measures classification ability across all threshold values.
- **ROC Curve**: Probabilistic confidence in predictions (stored but not plotted in the script).

```
# Select best model based on AUC score
best_model_name = max(model_results.keys(), key=lambda k: model_results[k]['auc_score'])
best_model = model_results[best_model_name]['model']

print(f"\n🏆 Best Model: {best_model_name}")
print(f"🎯 Best AUC Score: {model_results[best_model_name]['auc_score']:.4f}")

return best_model, model_results, best_model_name
```

---

4. Model Artifacts

After training, the following artifacts are saved for future inference or deployment:

File Name	Description
best_model.pkl	The best-performing trained model
scaler.pkl	StandardScaler instance used for feature scaling
feature_names.pkl	List of feature names after one-hot encoding
label_encoders.pkl	Dictionary of label encoders for binary features
test_data.pkl	Contains X_test, y_test, model results, and best model name

---

- **Output Summary**

Once the training completes, you'll see a summary like:

- MODEL TRAINING COMPLETED SUCCESSFULLY!
- Best Model: Gradient Boosting
- Accuracy: 0.8652
- AUC Score: 0.9124