Telco Customer Churn Prediction - Model Training & Deployment Documentation

❖ Overview

This project builds a robust and explainable machine learning pipeline for **predicting customer churn** using Telco customer data. The pipeline includes:

- Data preprocessing (binary encoding, one-hot encoding, scaling)
- Class imbalance handling via **SMOTE**
- **Hyperparameter tuning** using GridSearchCV
- Testing on real-world-like examples
- Saving artifacts for reuse in web apps or APIs

❖ Project Flow

1. **Load & preprocess data** (load_and_preprocess_data)
2. **Train and tune models** (train_models)
3. **Preprocess single customers** (preprocess_single_customer)
4. **Evaluate performance**
5. **Save artifacts for production**

❖ Dataset

- **File**: telco_churn.csv
- **Target**: Churn (Yes/No → 1/0)
- **Dropped column**: customerID (non-informative)
- **Preprocessed fields**: Binary & categorical features

---

1. Data Preprocessing (load_and_preprocess_data)

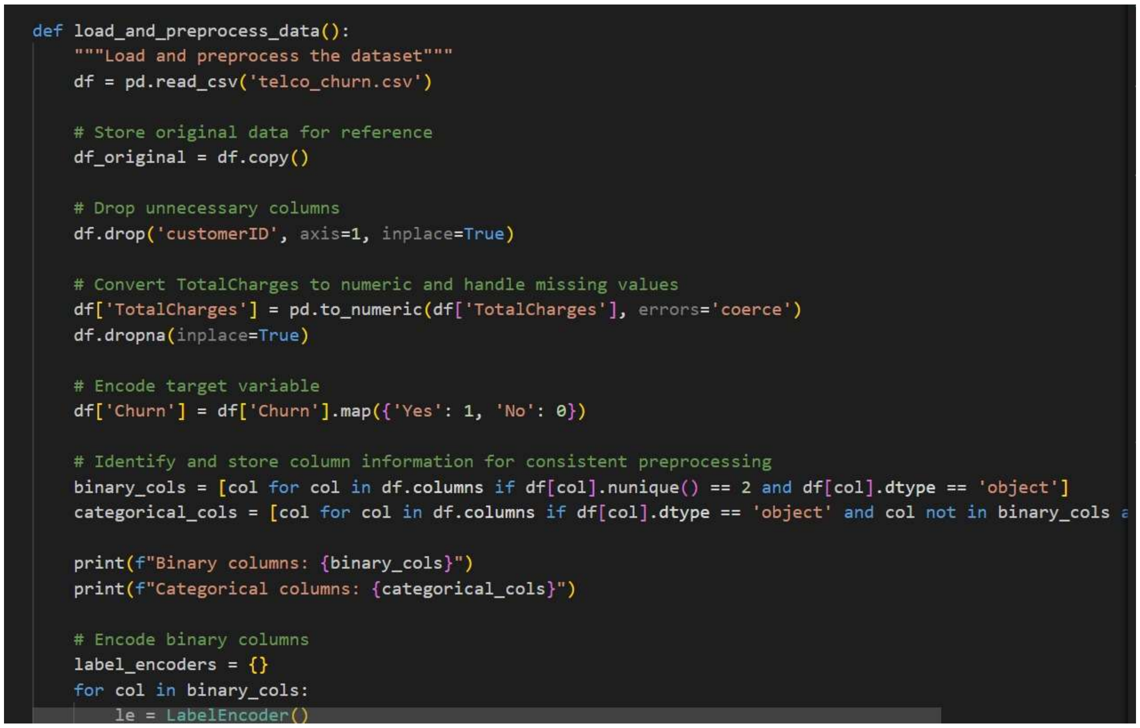
Key Steps:

- Converts TotalCharges to numeric and handles missing values.
- Encodes:
    - **Binary columns** with LabelEncoder

    - **Categorical columns** with pd.get_dummies (one-hot encoding)
- Separates and stores preprocessing info for future input consistency.

Returns:

- Raw and encoded datasets
- Label encoders
- Preprocessing metadata (column types, feature names, etc.)



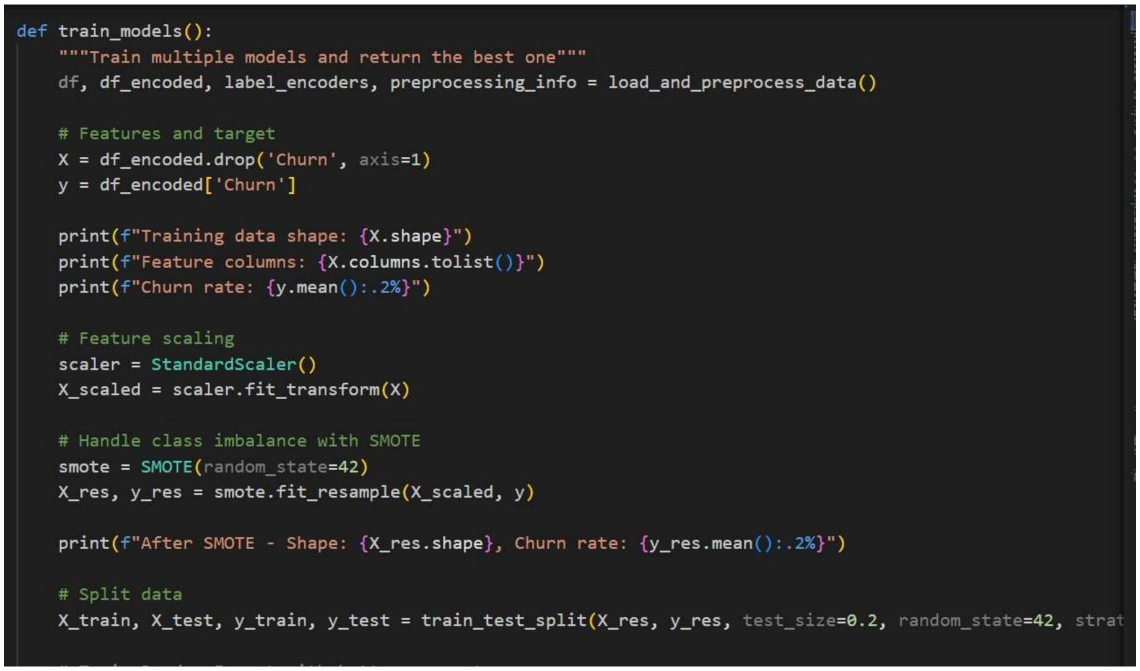
```python
def load_and_preprocess_data():
    """Load and preprocess the dataset"""
    df = pd.read_csv('telco_churn.csv')

    # Store original data for reference
    df_original = 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')
    df.dropna(inplace=True)

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

    # Identify and store column information for consistent preprocessing
    binary_cols = [col for col in df.columns if df[col].nunique() == 2 and df[col].dtype == 'object']
    categorical_cols = [col for col in df.columns if df[col].dtype == 'object' and col not in binary_cols a

    print(f"Binary columns: {binary_cols}")
    print(f"Categorical columns: {categorical_cols}")

    # Encode binary columns
    label_encoders = {}
    for col in binary_cols:
        le = LabelEncoder()
```

---

2. Model Training & Evaluation (train_models)

Enhancements:

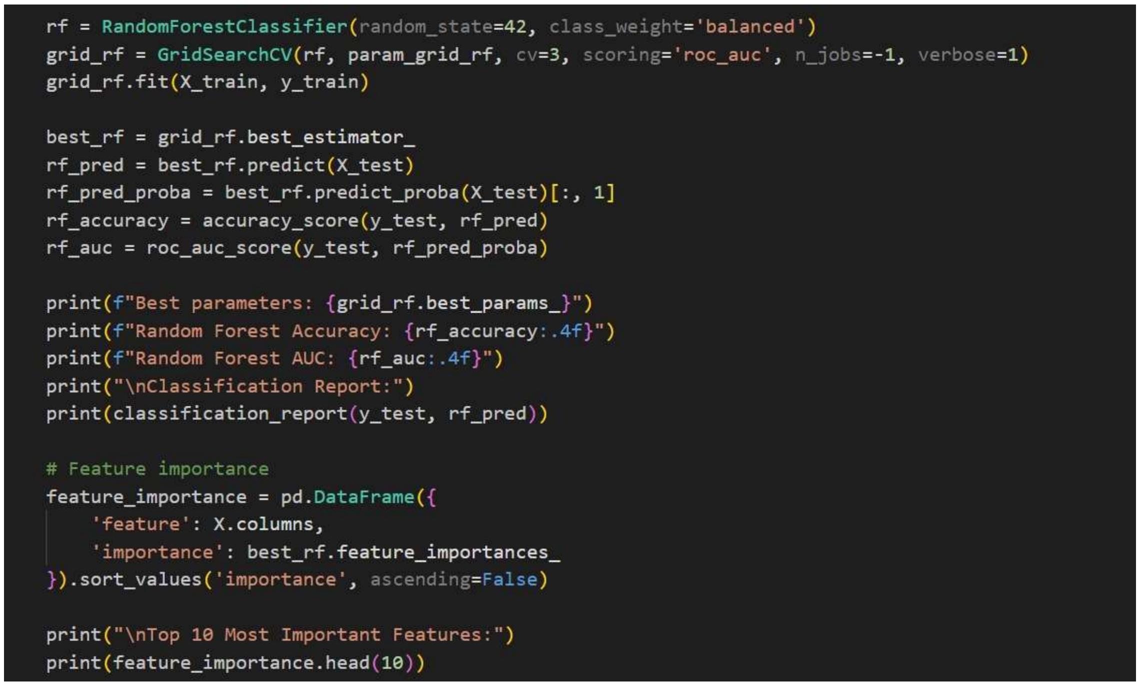
- Applies **StandardScaler** to all numerical data.
- Uses **SMOTE** to balance classes.
- **Stratified Train-Test Split** ensures balanced representation.
- Uses **GridSearchCV** to optimize Random Forest parameters like:
    - n_estimators, max_depth, min_samples_split, min_samples_leaf, and max_features



```
def train_models():
    """Train multiple models and return the best one"""
    df, df_encoded, label_encoders, preprocessing_info = load_and_preprocess_data()

    # Features and target
    X = df_encoded.drop('Churn', axis=1)
    y = df_encoded['Churn']

    print(f"Training data shape: {X.shape}")
    print(f"Feature columns: {X.columns.tolist()}")
    print(f"Churn rate: {y.mean():.2%}")

    # Feature scaling
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    # Handle class imbalance with SMOTE
    smote = SMOTE(random_state=42)
    X_res, y_res = smote.fit_resample(X_scaled, y)

    print(f"After SMOTE - Shape: {X_res.shape}, Churn rate: {y_res.mean():.2%}")

    # Split data
    X_train, X_test, y_train, y_test = train_test_split(X_res, y_res, test_size=0.2, random_state=42, strat
```

Model Evaluation:

- **Metrics printed**:
  - Accuracy
  - AUC Score
  - Classification Report
  - Top 10 most important features



```
rf = RandomForestClassifier(random_state=42, class_weight='balanced')
grid_rf = GridSearchCV(rf, param_grid_rf, cv=3, scoring='roc_auc', n_jobs=-1, verbose=1)
grid_rf.fit(X_train, y_train)

best_rf = grid_rf.best_estimator_
rf_pred = best_rf.predict(X_test)
rf_pred_proba = best_rf.predict_proba(X_test)[:, 1]
rf_accuracy = accuracy_score(y_test, rf_pred)
rf_auc = roc_auc_score(y_test, rf_pred_proba)

print(f"Best parameters: {grid_rf.best_params_}")
print(f"Random Forest Accuracy: {rf_accuracy:.4f}")
print(f"Random Forest AUC: {rf_auc:.4f}")
print("\nClassification Report:")
print(classification_report(y_test, rf_pred))

# Feature importance
feature_importance = pd.DataFrame({
    'feature': X.columns,
    'importance': best_rf.feature_importances_
}).sort_values('importance', ascending=False)

print("\nTop 10 Most Important Features:")
print(feature_importance.head(10))
```

Test Predictions:

- Two realistic customer profiles (high & low churn risk)
- Processes and outputs churn probability with customer insights.

Saved Artifacts

Artifact	**File Name**	**Purpose**
Trained Model	best_model.pkl	Final model with best parameters
Scaler	scaler.pkl	Used to normalize customer input
Label Encoders	label_encoders.pkl	For consistent encoding of binary fields
Preprocessing Metadata	preprocessing_info.pkl	Stores column info for new inputs
Feature Columns List	feature_names.pkl	Ensures input columns are in correct order
Test Data for Evaluation	test_data.pkl	For performance checks or dashboards

---

4. Preprocessing New Customers (preprocess_single_customer)

Handles preprocessing of a **single customer dictionary** for prediction.

Features:

- Handles missing fields
- Uses stored label encoders
- Applies one-hot encoding
- Reorders columns to match training
- Scales features using trained scaler

Output:

- Scaled NumPy array ready for model prediction