Mini Project 0: Advanced Telco Customer Churn Prediction

Building Production-Ready Machine Learning Systems

🎯 Project Objective

Apply advanced concepts from **Week 3 (Exploratory Data Analysis), Week 3 (Ensemble Methods & Decision Trees)**, **Week 4 (Model Evaluation & Class Imbalance), Week 4 (Building Pipelines)** to build a comprehensive churn prediction system for Telco customers. This project emphasizes real-world machine learning challenges including class imbalance, ensemble methods, business-focused evaluation metrics, and **production-ready pipeline development**.

📊 Dataset Information

Source: [Telco Customer Churn Dataset (Kaggle)]()
File: `WA_Fn-UseC_-Telco-Customer-Churn.csv`
Size: 7,043 customers, 21 features
Target Variable: `Churn` (Yes/No)

Key Features:
- **Customer Demographics**: Gender, SeniorCitizen, Partner, Dependents
- **Account Information**: Tenure, Contract, PaperlessBilling, PaymentMethod
- **Services**: PhoneService, MultipleLines, InternetService, OnlineSecurity, etc.
- **Financial**: MonthlyCharges, TotalCharges

📋 Project Requirements

Deadline: 22 Aug 2025

Deliverables:

1. **Jupyter Notebook** with comprehensive analysis and modeling

2. **Modular Python Pipeline** with production-ready code structure
3. **Executive Summary Report** (2-3 pages) with business insights and recommendations
4. **Model Performance Comparison** with proper evaluation metrics
5. **Business Impact Analysis** with actionable recommendations

---

🔍 Part 1: Advanced Exploratory Data Analysis (EDA)

1.1 Initial Data Assessment

- **Data Quality Check**: Examine data types, missing values, and inconsistencies
- **Target Variable Analysis**: Calculate churn rate and discuss class imbalance implications
- **Feature Overview**: Categorize features into demographic, behavioral, and financial groups

1.2 Class Imbalance Analysis

- **Visualize class distribution** with appropriate charts
- **Calculate imbalance ratio** and discuss impact on model evaluation
- **Analyze churn patterns** across different customer segments
- **Business Context**: Explain why class imbalance matters in churn prediction

1.3 Advanced Univariate Analysis

- **Numerical Features**: Distribution analysis, outlier detection using IQR and Z-score methods
- **Categorical Features**: Frequency analysis and relationship with churn
- **Feature Engineering Opportunities**: Identify potential derived features

1.4 Comprehensive Bivariate Analysis

- **Churn vs Demographics**: Age groups, gender, family status impact
- **Churn vs Services**: Service adoption patterns and churn correlation
- **Churn vs Financial**: Monthly charges, total charges, and payment behavior
- **Statistical Significance**: Use appropriate tests (Chi-square, t-tests) to validate relationships

1.5 Multivariate Analysis

- **Correlation Matrix**: Identify multicollinearity issues
- **Feature Interactions**: Explore combinations that influence churn (e.g., Contract + PaymentMethod)

- **Customer Segmentation**: Group customers by behavior patterns

1.6 Business Insights Generation

- **High-Risk Customer Profiles**: Identify characteristics of customers most likely to churn
- **Retention Opportunities**: Services or contract types that reduce churn
- **Revenue Impact**: Calculate potential revenue loss from churning customers

---

⚙️ Part 2: Advanced Model Pipeline & Ensemble Methods

2.1 Data Preprocessing Pipeline

- **Data Cleaning**: Handle inconsistencies (e.g., TotalCharges data type issues)
- **Feature Engineering**: Create meaningful derived features
    - Tenure categories (New, Established, Loyal)
    - Service adoption score
    - Average monthly charges per service
    - Payment reliability indicators
- **Encoding Strategies**: Compare different encoding methods for categorical variables
- **Feature Scaling**: Apply appropriate scaling for numerical features

2.2 Ensemble Model Implementation

Implement and compare the following ensemble methods:

2.2.1 Bagging Method: Random Forest

- **Implementation**: Use scikit-learn RandomForestClassifier
- **Hyperparameters to tune**: n_estimators, max_depth, min_samples_split, max_features
- **Analysis**: Feature importance interpretation and business insights

2.2.2 Boosting Method: XGBoost

- **Implementation**: Use XGBoost library
- **Hyperparameters to tune**: learning_rate, max_depth, n_estimators, subsample
- **Analysis**: Feature importance and model interpretation

2.2.3 Advanced Boosting: CatBoost

- **Implementation**: Use CatBoost library for native categorical handling
- **Advantages**: Automatic categorical encoding, reduced overfitting
- **Analysis**: Compare performance with other methods

2.2.4 Baseline Comparison

- **Logistic Regression**: Simple baseline model
- **Decision Tree**: Single tree for interpretability comparison

2.3 Pipeline Construction

- **Scikit-learn Pipelines**: Create modular, reproducible preprocessing and modeling pipelines
- **Cross-Validation Strategy**: Use stratified k-fold to maintain class distribution
- **Hyperparameter Tuning**: Implement GridSearchCV or RandomizedSearchCV

📈 Part 3: Model Evaluation for Imbalanced Data

3.1 Class Imbalance Considerations

- **Why Accuracy Fails**: Demonstrate with concrete examples why accuracy is misleading
- **Business Impact**: Explain cost of false positives vs. false negatives in churn prediction

3.2 Comprehensive Evaluation Metrics

Evaluate all models using the following metrics with detailed interpretation:

3.2.1 Primary Metrics

- **Precision**: Quality of churn predictions (campaign efficiency)
- **Recall**: Coverage of actual churners (revenue protection)
- **F1-Score**: Balanced performance measure
- **Confusion Matrix:**: Overall performance analysis

3.2.2 Business-Focused Metrics

- **Precision-Recall AUC**: Better for imbalanced data
- **Cost-Sensitive Analysis**: Calculate business impact of different error types
- **Threshold Optimization**: Find optimal threshold for business objectives

3.3 Model Comparison Framework

- **Performance Matrix**: Compare all models across all metrics
- **Statistical Significance**: Use appropriate tests to validate performance differences
- **Business Value Analysis**: Translate metrics into business impact (revenue saved, campaign efficiency)

🔧 Part 4: Production-Ready Pipeline Development

4.1 Modular Code Architecture

Transform your Jupyter notebook analysis into a production-ready pipeline with the following modular structure:

4.1.1 Data Processing Modules

Create separate Python modules for each data processing step:

- `data_ingestion.py`: Data loading and initial validation
- `handle_missing_values.py`: Missing value detection and imputation strategies
- `outlier_detection.py`: Outlier detection using IQR and Z-score methods
- `feature_encoding.py`: Categorical encoding (one-hot, label, ordinal)
- `feature_scaling.py`: Numerical feature scaling (StandardScaler, MinMaxScaler)
- `feature_binning.py`: Feature binning for continuous variables
- `data_splitter.py`: Train/validation/test split with stratification

4.1.2 Model Development Modules

- `model_building.py`: Model factory for different ensemble methods
- `model_training.py`: Training strategies (cross-validation, holdout)
- `model_evaluation.py`: Comprehensive evaluation metrics for imbalanced data
- `model_inference.py`: Prediction pipeline for new data

4.1.3 Pipeline Orchestration

- `data_pipeline.py`: End-to-end data preprocessing pipeline
- `training_pipeline.py`: Complete model training and evaluation pipeline
- `streaming_inference_pipeline.py`: Real-time prediction pipeline

4.2 Pipeline Implementation Requirements

4.2.1 Code Structure Standards

- **Object-Oriented Design**: Use classes and inheritance for reusable components
- **Strategy Pattern**: Implement different strategies for encoding, scaling, etc.
- **Configuration Management**: Use config files for hyperparameters and settings
- **Error Handling**: Robust exception handling and logging
- **Type Hints**: Use Python type hints for better code documentation

4.2.2 Data Pipeline Features

- **Configurable Processing**: Allow different preprocessing strategies
- **Data Validation**: Implement data quality checks and validation
- **Pipeline Persistence**: Save and load preprocessing pipelines
- **Reproducibility**: Ensure consistent results with random seeds

4.2.3 Training Pipeline Features

- **Multiple Model Support**: Train different ensemble methods
- **Hyperparameter Optimization**: Automated hyperparameter tuning
- **Model Comparison**: Systematic comparison across evaluation metrics
- **Model Persistence**: Save trained models and evaluation results

4.2.4 Inference Pipeline Features

- **Batch Prediction**: Process multiple samples efficiently
- **Single Sample Prediction**: Handle individual customer predictions
- **Input Validation**: Validate input data format and ranges
- **Probability Outputs**: Return prediction probabilities for threshold optimization

4.3 Pipeline Testing and Validation

- **Unit Tests**: Test individual components and functions
- **Integration Tests**: Test end-to-end pipeline functionality
- **Data Validation Tests**: Ensure data quality and consistency
- **Model Performance Tests**: Validate model performance benchmarks

---

🎯 Part 5: Business Impact Analysis

5.1 Customer Segmentation for Retention

- **High-Risk Segment**: Customers with high churn probability
- **Medium-Risk Segment**: Customers requiring proactive engagement
- **Low-Risk Segment**: Loyal customers for upselling opportunities

5.2 Retention Strategy Recommendations

- **Targeted Interventions**: Specific actions for each risk segment
- **Resource Allocation**: Budget optimization for retention campaigns
- **Expected ROI**: Calculate return on investment for retention efforts

---

🏆 Bonus Challenges (Optional)

5.1 Advanced Class Imbalance Handling

- **SMOTE**: Synthetic Minority Oversampling Technique
- **Cost-Sensitive Learning**: Adjust class weights in algorithms
- **Ensemble of Balanced Models**: Combine models trained on balanced subsets

5.2 Advanced Ensemble Techniques

- **Stacking**: Implement meta-learner combining base models
- **Voting Classifiers**: Hard and soft voting combinations
- **Model Blending**: Weighted combination of predictions

5.3 Advanced Feature Engineering

- **Temporal Features**: Customer lifecycle stage analysis
- **Interaction Features**: Meaningful feature combinations
- **Domain-Specific Features**: Telecom industry insights

5.4 Pipeline Optimization

- **Performance Profiling**: Identify and optimize bottlenecks
- **Memory Optimization**: Efficient data handling for large datasets
- **Parallel Processing**: Implement parallel training and prediction
- **Configuration Management**: Advanced configuration and parameter management

---

📝 Evaluation Criteria

Technical Excellence (40%)

- Proper implementation of ensemble methods
- Correct evaluation metrics for imbalanced data
- Quality of data preprocessing and feature engineering
- Pipeline architecture and modularity

Business Insight (30%)

- Quality of EDA insights and business interpretation
- Actionable recommendations for retention strategies

- Understanding of business impact and ROI
- Clear communication of technical concepts

Methodology (20%)

- Appropriate handling of class imbalance
- Proper cross-validation and hyperparameter tuning
- Statistical rigor in analysis and comparison
- Reproducibility of results

Code Quality (10%)

- Clean, well-documented, and modular code
- Proper error handling and logging
- Following Python best practices and PEP 8
- Comprehensive testing and validation

🎓 Learning Outcomes

Upon completion, students will demonstrate:

1. **Advanced EDA Skills**: Ability to extract meaningful business insights from data
2. **Ensemble Method Mastery**: Understanding of bagging, boosting, and their applications
3. **Imbalanced Data Expertise**: Proper evaluation and handling of class imbalance
4. **Business Acumen**: Translation of technical results into business value
5. **Production Pipeline Development**: Building modular, maintainable ML pipelines
6. **Software Engineering Skills**: Writing clean, testable, and scalable code

---

📚 Required Libraries and Tools

Core Libraries

- `pandas`, `numpy`: Data manipulation and analysis
- `matplotlib`, `seaborn`, `plotly`: Data visualization
- `scikit-learn`: Machine learning algorithms and evaluation
- `xgboost`: Gradient boosting implementation
- `catboost`: Advanced boosting with categorical support

Pipeline Development

- `pickle`, `joblib`: Model and pipeline serialization
- `logging`: Comprehensive logging and debugging
- `argparse`: Command-line interface for pipelines
- `pytest`: Unit testing framework
- `typing`: Type hints for better code documentation

Optional Advanced Libraries

- `imbalanced-learn`: Class imbalance handling techniques
- `optuna`: Advanced hyperparameter optimization
- `pydantic`: Data validation and settings management

💡 Success Tips

1. **Start with EDA**: Begin with thorough exploratory analysis in Jupyter notebook
2. **Modularize Gradually**: Transform notebook code into modular pipeline components
3. **Test Early and Often**: Write tests as you develop each module
4. **Focus on Business Value**: Always connect technical findings to business implications
5. **Document Everything**: Clear documentation helps with understanding and debugging
6. **Think Production**: Build code that could actually run in a business environment