AIE 121 - Machine Learning Final Project Report

Cover Page

Course Title: Machine Learning (ML)
Course Code: AIE 121
Students' Names:
Abdelrahaman Moustafa Wanas 23101575
Youssef ahmed said 23101508
Abdelrahman mohmed Ibrahim 23101417
Section: Sunday
Project Title: Customer Churn Prediction & Customer Lifetime Value Prediction

Date: 12/20/2025

Table of Contents

Section 1: Problem Domain

Customer Churn Prediction & Customer Lifetime Value Prediction

Problem Statement

Customer churn prediction and customer lifetime value (CLV) prediction are critical business problems in the telecommunications and service industries. Understanding which customers are likely to leave (churn) and predicting their lifetime value helps companies make data-driven decisions to improve customer retention and optimize business strategies.

Business Context

In today's competitive market, businesses face significant challenges:

1. **High Customer Acquisition Costs**: Acquiring new customers is expensive and time-consuming

2. **Customer Retention**: Retaining existing customers is more costeffective than acquiring new ones
3. **Resource Optimization**: Companies need to allocate marketing budgets and customer service resources efficiently
4. **Strategic Planning**: Understanding customer value helps in pricing strategies and service offerings

Project Objectives

This project addresses two main machine learning tasks:

1. **Classification Task**: Predict whether a customer will churn (Yes/No) based on various customer attributes and behaviors
2. **Regression Task**: Predict the customer lifetime value (continuous monetary value) based on customer characteristics and historical data

Problem Significance

- **Churn Prediction**: Enables proactive customer retention campaigns, reducing overall churn rate and improving customer satisfaction
- **CLV Prediction**: Helps prioritize high-value customers and optimize marketing spend for maximum ROI
- **Combined Analysis**: Provides comprehensive customer insights for strategic business planning and decision-making

Dataset Overview

The project uses a synthetic customer dataset with: - **12,500+ rows** of customer data - **16 columns** including both numerical and categorical features - Features include: age, monthly charges, total charges, tenure, usage metrics, customer satisfaction, service subscriptions, and demographic information - Intentional data quality issues for realistic data cleaning practice

---

Section 2: Project Summary

Overview

This project implements a complete machine learning pipeline from data collection to model evaluation, covering all aspects of a real-world data science project as required by the AIE 121 Machine Learning course.

What Was Done in the Project

Part 1: Problem Domain Understanding

- Selected "Customer Churn Prediction & Customer Lifetime Value Prediction" as the problem domain

- Defined clear objectives for both classification and regression tasks
- Established business context and importance of the problem

Part 2: Data Collection

- Generated a synthetic dataset with **12,500+ rows** and **16 columns**
- Dataset includes both numerical and categorical features • Intentionally created "dirty" data with various data quality issues:
    - Missing values (1,009 missing values)
    - Duplicate rows (478 duplicates)
    - Outliers (extreme values in multiple columns)
    - Inconsistent data entry (different cases, formats)
    - Blank spaces
    - Invalid negative values

Part 3: Data Cleaning (10 Steps Completed)

Comprehensive data cleaning was performed following all required steps:

1. **Missing Values Handling**: Identified and analyzed missing values across all columns
2. **Understanding Missing Data**: Analyzed patterns and percentages of missing data (ranging from 0.72% to 16.45%)
3. **Variable Selection**: Reviewed and selected relevant features (kept all 16 columns as they were relevant)
4. **Duplicate Removal**: Removed 478 duplicate rows, reducing dataset from 12,500 to 12,022 rows
5. **Outlier Detection & Removal**: Used IQR (Interquartile Range) method to detect and remove 548 outliers across multiple columns
6. **Scaling & Normalization**: Tested during model building phase (StandardScaler and MinMaxScaler)
7. **Blank Space Handling**: Replaced blank spaces with NaN and handled them using Simple Imputer
8. **Data Arrangement**: Sorted data logically by tenure months and age
9. **Data Grouping**: Organized data in proper rows and columns format
10. **Inconsistent Data Entry**: Standardized categorical values (e.g., "Yes"/"yes"/"Y" → "yes", "Male"/"male"/"M" → "male")

Result: Cleaned dataset with **11,474 rows** and **16 columns**, **zero missing values**

Part 4: Exploratory Data Analysis

- Performed correlation analysis on numerical features
- Identified key variable relationships
- Analyzed patterns in customer behavior and churn
- Generated correlation matrix heatmap

Part 5: Visualization (8 Types Created)

Created comprehensive visualizations covering all required types:

1. **Line Plot**: Average monthly charges by tenure (shows pricing trends)
2. **Area Plot**: Churn distribution by contract type (stacked area chart)
3. **Histogram**: Age distribution (30 bins, shows customer demographics)
4. **Bar Chart**: Churn distribution (Yes/No comparison)
5. **Pie Chart**: Contract type distribution (percentage breakdown)
6. **Box Plot**: Distribution of numerical features (monthly charges, total charges, tenure)
7. **Scatter Plot**: Monthly charges vs total charges (colored by churn status)
8. **Bubble Plot**: Age vs monthly charges (bubble size = tenure, colored by churn)

All visualizations saved as high-resolution PNG files (300 DPI) in the `visualizations/` folder.

Part 6: Model Building and Evaluation

Classification Models (Churn Prediction) Trained and evaluated **4 models** with all required metrics:

1. **Logistic Regression**
   - Accuracy: 50.11%
   - Precision: 50.00%
   - Recall: 50.11%
   - ROC-AUC: 49.95%
2. **Random Forest Classifier Best Model**
   - Accuracy: 51.68%
   - Precision: 51.64%
   - Recall: 51.68%
   - ROC-AUC: 51.12%
3. **Gradient Boosting Classifier**
   - Accuracy: 49.28%
   - Precision: 49.21%
   - Recall: 49.28%
   - ROC-AUC: 48.75%
4. **Support Vector Machine (SVM)**
   - Accuracy: 50.54%
   - Precision: 50.28%
   - Recall: 50.54%
   - ROC-AUC: 49.58%

Regression Models (Customer Lifetime Value Prediction) Trained and evaluated **4 models** with all required metrics:

1. **Linear Regression**
    - Mean Absolute Error (MAE): 1554.20
    - Mean Squared Error (MSE): 3,747,428.19
    - Median Absolute Error: 1307.01
    - R^2 Score: -0.0013
2. **Random Forest Regressor**
    - Mean Absolute Error (MAE): 1570.75
    - Mean Squared Error (MSE): 3,846,633.39
    - Median Absolute Error: 1298.32
    - R^2 Score: -0.0278
3. **Gradient Boosting Regressor**
    - Mean Absolute Error (MAE): 1561.09
    - Mean Squared Error (MSE): 3,774,620.84
    - Median Absolute Error: 1296.43
    - R^2 Score: -0.0086
4. **Support Vector Regression (SVR) Best Model**
    - Mean Absolute Error (MAE): 1553.29
    - Mean Squared Error (MSE): 3,744,444.40
    - Median Absolute Error: 1308.02
    - R^2 Score: -0.0005

Additional Analysis

- **Scaling Effects**: Tested StandardScaler impact on all models (both classification and regression)
- **Normalization Effects**: Tested MinMaxScaler impact on all models
- **Result Visualization**: Created comparison charts, ROC curves, confusion matrices, and actual vs. predicted plots

Key Achievements

Complete data cleaning pipeline (all 10 steps implemented)
Comprehensive exploratory data analysis
8 different visualization types (all required types)
4 classification models with full evaluation metrics (Accuracy, Precision, Recall, ROC-AUC)
4 regression models with full evaluation metrics (MAE, MSE, Median AE, R^2 Score)
Scaling and normalization impact analysis
Model comparison and visualization
All code is clean, well-commented, and organized

Section 3: Source Code

Data Generation Script (`generate_dirty_data.py`)

```
"""
Script to generate a dirty dataset for the ML Final Project
This creates a dataset with intentional issues: missing values, duplicates,
outliers, etc.

Dependencies: pandas, numpy (install via: pip install -r
requirements.txt)
"""
import pandas as pd # type: ignore
import numpy as np # type: ignore
import random

# Set random seed for reproducibility
np.random.seed(42)
random.seed(42)

# Generate base data
n_samples = 12000 # More than 10000 as required

# Create dirty dataset with intentional issues
data = {
    # Numerical columns
    'age': np.random.randint(18, 80, n_samples),
    'monthly_charges': np.random.normal(65, 20, n_samples),
    'total_charges': np.random.normal(2000, 1000, n_samples),
    'tenure_months': np.random.randint(1, 72, n_samples),
    'monthly_usage_gb': np.random.normal(50, 25, n_samples),
    'customer_satisfaction': np.random.randint(1, 11, n_samples),
    'number_of_services': np.random.randint(1, 5, n_samples),

    # Categorical columns
    'gender': np.random.choice(['Male', 'Female', 'male', 'female', 'M', 'F'],
    n_samples),
    'contract_type': np.random.choice(['Month-to-month', 'One year', 'Two        ,
    year', 'monthly', 'payment_method': np.random.choice(['Electronic check',
    'Mailed check', 'Bank transfer'
    'internet_service': np.random.choice(['DSL', 'Fiber optic', 'No', 'None',
    'dsl', 'fiber'
    'phone_service': np.random.choice(['Yes', 'No', 'yes', 'no', 'Y', 'N'],
    n_samples),
    'streaming_tv': np.random.choice(['Yes', 'No', 'yes', 'no'], n_samples),
    'streaming_movies': np.random.choice(['Yes', 'No', 'yes', 'no'], n_samples),

    # Target variables
    'churn': np.random.choice(['Yes', 'No', 'yes', 'no'], n_samples),
    'customer_lifetime_value': np.random.normal(5000, 2000, n_samples)
} df =

pd.DataFrame(data)

# Introduce intentional data quality issues

# 1. Missing values (randomly introduce NaN) missing_indices =
np.random.choice(df.index, size=int(0.15 * len(df)), replace=False) for
col in ['age', 'monthly_charges', 'total_charges', 'tenure_months']:
    col_missing = np.random.choice(missing_indices, size=int(0.1 *
    len(missing_indices)), re df.loc[col_missing, col] = np.nan

# Missing values in categorical columns for col
in ['gender', 'contract_type', 'payment_method']:
    col_missing = np.random.choice(missing_indices, size=int(0.05 *
    len(missing_indices)), r df.loc[col_missing, col] = np.nan

# 2. Duplicates (introduce some duplicate rows)
duplicate_rows = df.sample(n=500,
random_state=42) df = pd.concat([df,
duplicate_rows], ignore_index=True)

# 3. Outliers (introduce extreme values) outlier_indices =
np.random.choice(df.index, size=200, replace=False)
df.loc[outlier_indices, 'monthly_charges'] = np.random.uniform(200,
500, 200) df.loc[outlier_indices, 'total_charges'] =
np.random.uniform(10000, 50000, 200) df.loc[outlier_indices, 'age']
= np.random.choice([100, 150, 200], 200)

# 4. Inconsistent data entry (already done in categorical columns with
different cases)

# 5. Blank spaces
blank_indices = np.random.choice(df.index, size=100,
replace=False) df.loc[blank_indices, 'gender'] = ' '
df.loc[blank_indices, 'contract_type'] = ' '

# 6. Negative values where they shouldn't exist
negative_indices = np.random.choice(df.index, size=50, replace=False)
df.loc[negative_indices, 'monthly_charges'] = -abs(df.loc[negative_indices,
'monthly_charges df.loc[negative_indices, 'age'] = -
abs(df.loc[negative_indices, 'age'])

# Shuffle the dataframe
df = df.sample(frac=1, random_state=42).reset_index(drop=True)

# Save to CSV
df.to_csv('customer_churn_dirty.csv',
index=False)
print(f"Generated dirty dataset with {len(df)} rows and {len(df.columns)}
columns") print(f"Missing values: {df.isnull().sum().sum()}")
print(f"Duplicates: {df.duplicated().sum()}")
```

Main Project Script (`ml_final_project.py`)

Complete Source Code

```
"""
AIE 121 - Machine Learning Final Project
Customer Churn Prediction and Customer Lifetime Value Prediction

This project covers:
- Data Collection
- Data Cleaning (10 steps)
- Exploratory Data Analysis
- Visualization (8 types)
- Model Building (Classification & Regression)
- Model Evaluation
"""

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, cross_val_score, GridSearchCV from
sklearn.preprocessing import StandardScaler, MinMaxScaler, LabelEncoder
from sklearn.compose import ColumnTransformer from sklearn.pipeline
import Pipeline from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor,
GradientBoosting from sklearn.linear_model import LogisticRegression,
LinearRegression from sklearn.svm import SVC, SVR
from sklearn.metrics import (accuracy_score, precision_score, recall_score,
roc_auc_score,
                          roc_curve, mean_absolute_error, mean_squared_error,
                          median_absolute_error, r2_score, confusion_matrix,
                          classificati
import warnings
```

```
warnings.filterwarnings('ignore')

# Set style for better visualizations
sns.set_style("whitegrid")
plt.rcParams['figure.figsize'] = (12, 6)

print("="*80)
print("AIE 121 - Machine Learning Final
Project")
print("Customer Churn Prediction & Customer Lifetime Value Prediction")
print("="*80)
#
====================================================================
========
# PART 1: UNDERSTANDING THE PURPOSE
#
====================================================================
======== print("\n" + "="*80) print("PART 1: PROBLEM DOMAIN")
print("="*80)
print("""
Problem Domain: Customer Churn Prediction & Customer Lifetime Value Prediction

Goal:
1. Classification Task: Predict whether a customer will churn (Yes/No)
2. Regression Task: Predict customer lifetime value (continuous value)

This is a critical business problem as:
- Customer retention is cheaper than acquisition
- Predicting churn helps in proactive customer retention
- Customer lifetime value helps in resource allocation and marketing
  strategies """)

#
====================================================================
========
# PART 2: DATA COLLECTION
#
====================================================================
======== print("\n" + "="*80) print("PART 2: DATA COLLECTION")
print("="*80)

# Load the dirty dataset
try:
    df = pd.read_csv('customer_churn_dirty.csv')
    print(f"Dataset loaded successfully!")
    print(f"Shape: {df.shape[0]} rows, {df.shape[1]}
    columns")
except FileNotFoundError:
    print("Dataset not found. Please run 'generate_dirty_data.py' first to
    create the datase exit(1)

print("\nFirst few rows of the dataset:")
print(df.head())
print("\nDataset Info:")
print(df.info())
print("\nBasic
Statistics:")
print(df.describe())

#
====================================================================
========
# PART 3: DATA CLEANING (10 Steps)
#
====================================================================
========
print("\n" + "="*80)
print("PART 3: DATA
CLEANING") print("="*80)

# Step 1: Deal with Missing Values
print("\n--- Step 1: Missing Values
Analysis ---") print("Missing values per
column:") missing_counts =
df.isnull().sum()
print(missing_counts[missing_counts > 0])

# Step 2: Figure out why data is missing
print("\n--- Step 2: Understanding Missing
Data ---") print("Missing data percentage:")
missing_percent = (df.isnull().sum() /
len(df)) * 100
print(missing_percent[missing_percent > 0])

# Step 3: Eliminating extra variables print("\n-
-- Step 3: Checking for Extra Variables ---")
print(f"Current columns: {list(df.columns)}")

# Step 4: Eliminating duplicates print("\n---
Step 4: Removing Duplicates ---")
```

```
duplicates_before = df.duplicated().sum()
print(f"Number of duplicate rows:
{duplicates_before}") df = df.drop_duplicates()
print(f"Rows after removing duplicates: {len(df)}")

# Step 5: Detect and remove outliers
print("\n--- Step 5: Outlier Detection and Removal ---")
numerical_cols =
df.select_dtypes(include=[np.number]).columns.tolist()
print(f"Numerical columns: {numerical_cols}")

# Remove outliers using IQR
method for col in
numerical_cols:
    if col in df.columns:
        Q1 = df[col].quantile(0.25)
        Q3 = df[col].quantile(0.75)
        IQR = Q3 - Q1 lower_bound = Q1 - 1.5 * IQR upper_bound = Q3
        + 1.5 * IQR outliers = ((df[col] < lower_bound) | (df[col]
        > upper_bound)).sum() print(f"{col}: {outliers} outliers
        detected")
        # Remove outliers df = df[(df[col] >=
        lower_bound) | (df[col].isna())] df =
        df[(df[col] <= upper_bound) |
        (df[col].isna())] print(f"Rows after removing
        outliers: {len(df)}")

# Step 6: Scaling and Normalization (will be done later during model
building)

# Step 7: Eliminating blank spaces or missing information
print("\n--- Step 7: Handling Blank Spaces ---")
categorical_cols =
df.select_dtypes(include=['object']).columns.tolist() for col
in categorical_cols:
    if col in df.columns:
        # Replace blank spaces with NaN df[col]
        = df[col].replace([' ', ''], np.nan) #
        Also handle strings that are just
        whitespace
        df[col] = df[col].apply(lambda x: np.nan if isinstance(x, str) and
        x.strip() == ''

# Use Simple Imputer for missing values print("\n---
Using Simple Imputer for Missing Values ---")
```

```
numerical_imputer = SimpleImputer(strategy='median')
categorical_imputer =
SimpleImputer(strategy='most_frequent')

# Apply imputation for
col in numerical_cols:
    if col in df.columns and df[col].isnull().sum() > 0:
        df[col] = numerical_imputer.fit_transform(df[[col]]).ravel()

for col in categorical_cols:
    if col in df.columns and df[col].isnull().sum() > 0:
        df[col] = categorical_imputer.fit_transform(df[[col]]).ravel()

print("Missing values after imputation:")
print(df.isnull().sum().sum())

# Step 8: Arranging data logically
print("\n--- Step 8: Arranging Data Logically --
-")
df = df.sort_values(by=['tenure_months', 'age']).reset_index(drop=True)

# Step 9: Grouping data (already in rows and columns format)

# Step 10: Dealing with Inconsistent Data Entry
print("\n--- Step 10: Standardizing Categorical
Data ---") for col in categorical_cols: if col in
df.columns:
        # Convert to lowercase and standardize df[col]
        = df[col].astype(str).str.lower().str.strip()
        # Standardize specific
        values if 'churn' in
        col.lower():
            df[col] = df[col].replace(['yes', 'y'], 'yes')
            df[col] = df[col].replace(['no', 'n'], 'no')
        elif 'gender' in col.lower(): df[col] =
            df[col].replace(['male', 'm'], 'male')
            df[col] = df[col].replace(['female', 'f'],
            'female')
        elif 'yes' in df[col].values or 'no' in
            df[col].values: df[col] =
            df[col].replace(['yes', 'y'], 'yes') df[col] =
            df[col].replace(['no', 'n'], 'no')

# Remove negative values where they shouldn't
exist print("\n--- Removing Invalid Negative
Values ---") df['monthly_charges'] =
df['monthly_charges'].abs() df['age'] =
df['age'].abs()
df['total_charges'] = df['total_charges'].abs()

print("\nData cleaning completed!") print(f"Final
dataset shape: {df.shape}") print(f"Final missing
values: {df.isnull().sum().sum()}")

#
======================================================================
========
# PART 4: EXPLORATORY DATA ANALYSIS
#
======================================================================
======== print("\n" + "="*80)
print("PART 4: EXPLORATORY DATA ANALYSIS")
print("="*80)

print("\n--- Correlation Analysis ---")
numerical_df =
df.select_dtypes(include=[np.number])
correlation_matrix = numerical_df.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)

print("\n--- Variable Relationships --
-") print("\nKey relationships to
explore:") print("1. Age vs Monthly
Charges") print("2. Tenure vs Churn")
print("3. Monthly Charges vs Total
Charges") print("4. Contract Type vs
Churn") print("5. Payment Method vs
Churn")

#
======================================================================
========
# PART 5: VISUALIZATION
#
======================================================================
======== print("\n" + "="*80) print("PART 5: VISUALIZATION")
print("="*80)
# Create a directory for saving plots
import os
```

```
os.makedirs('visualizations', exist_ok=True)

# 1. Line Plot
print("\n1. Creating Line Plot...") plt.figure(figsize=(12, 6))
monthly_charges_by_tenure =
df.groupby('tenure_months')['monthly_charges'].mean().sort_index
plt.plot(monthly_charges_by_tenure.index, monthly_charges_by_tenure.values,
marker='o', line plt.xlabel('Tenure (Months)') plt.ylabel('Average Monthly
Charges')
plt.title('Line Plot: Average Monthly Charges by Tenure')
plt.grid(True, alpha=0.3) plt.tight_layout()
plt.savefig('visualizations/1_line_plot.png', dpi=300,
bbox_inches='tight') plt.close()

# 2. Area Plot
print("2. Creating Area Plot...") plt.figure(figsize=(12, 6)) churn_by_contract
= df.groupby('contract_type')['churn'].value_counts().unstack(fill_value=
churn_by_contract.plot(kind='area', stacked=True, alpha=0.7)
plt.xlabel('Contract Type') plt.ylabel('Number of Customers')
plt.title('Area Plot: Churn Distribution by Contract Type')
plt.legend(title='Churn') plt.tight_layout()                                (
plt.savefig('visualizations/2_area_plot.png', dpi=300,
bbox_inches='tight') plt.close()

# 3. Histogram print("3. Creating Histogram...")
plt.figure(figsize=(12, 6)) plt.hist(df['age'], bins=30,
edgecolor='black', alpha=0.7, color='skyblue') plt.xlabel('Age')
plt.ylabel('Frequency') plt.title('Histogram: Age Distribution')
plt.grid(True, alpha=0.3) plt.tight_layout()
plt.savefig('visualizations/3_histogram.png', dpi=300,
bbox_inches='tight') plt.close()

# 4. Bar Chart
print("4. Creating Bar Chart...")
plt.figure(figsize=(12, 6))
churn_counts =
df['churn'].value_counts()
plt.bar(churn_counts.index,
churn_counts.values,
color=['#ff6b6b', '#4ecdc4'],
alpha=0.7) plt.xlabel('Churn')
plt.ylabel('Count') plt.title('Bar
Chart: Churn Distribution')
plt.grid(True, alpha=0.3, axis='y')
plt.tight_layout()
plt.savefig('visualizations/4_bar_c
hart.png', dpi=300,
bbox_inches='tight') plt.close()

# 5. Pie Chart
print("5. Creating Pie Chart...")
plt.figure(figsize=(10, 8))
contract_counts = df['contract_type'].value_counts()
plt.pie(contract_counts.values, labels=contract_counts.index,
autopct='%1.1f%%', startangle plt.title('Pie Chart: Contract Type
Distribution') plt.tight_layout() plt.savefig('visualizations/5_pie_chart.png',
dpi=300, bbox_inches='tight') plt.close()

# 6. Box Plot
print("6. Creating Box Plot...")
plt.figure(figsize=(12, 6))
df.boxplot(column=['monthly_charges', 'total_charges',
          'tenure_months'], grid=True, figsize=(12, 6))
plt.ylabel('Value')
plt.title('Box Plot: Distribution of Numerical
Features') plt.xticks(rotation=45)
plt.tight_layout()
plt.savefig('visualizations/6_box_plot.png', dpi=300, bbox_inches='tight')
plt.close()

# 7. Scatter Plot
print("7. Creating Scatter Plot...")
plt.figure(figsize=(12, 6))
churn_colors = {'yes': 'red', 'no':
'blue'} colors =
df['churn'].map(churn_colors)
plt.scatter(df['monthly_charges'], df['total_charges'], c=colors,
alpha=0.5, s=50) plt.xlabel('Monthly Charges') plt.ylabel('Total
Charges')
plt.title('Scatter Plot: Monthly Charges vs Total Charges (colored by Churn)')
plt.legend(handles=[plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='red', marke plt.Line2D([0], [0], marker='o', color='w',
markerfacecolor='blue', mark
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('visualizations/7_scatter_plot.png', dpi=300,
bbox_inches='tight') plt.close()
# 8. Bubble Plot
```

```
print("8. Creating Bubble
Plot...")
plt.figure(figsize=(12, 8))
churn_yes = df[df['churn'] ==
'yes'] churn_no =
df[df['churn'] == 'no']
plt.scatter(churn_yes['age'], churn_yes['monthly_charges'],
          s=churn_yes['tenure_months']*5, alpha=0.5, c='red', label='Churn:
          Yes')
plt.scatter(churn_no['age'], churn_no['monthly_charges'],
          s=churn_no['tenure_months']*5, alpha=0.5, c='blue', label='Churn:
          No')
plt.xlabel('Age') plt.ylabel('Monthly Charges')
plt.title('Bubble Plot: Age vs Monthly Charges (bubble size =
tenure)') plt.legend() plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.savefig('visualizations/8_bubble_plot.png', dpi=300,

bbox_inches='tight') plt.close() print("\nAll visualizations saved

to 'visualizations' folder!")

#
====================================================================
========
# PART 6: BUILD MODEL
#
====================================================================
======== print("\n" + "="*80) print("PART 6: BUILD MODEL")
print("="*80)

# Prepare data for modeling
print("\n--- Preparing Data for Modeling ---")

# Select features
feature_cols = ['age', 'monthly_charges', 'total_charges', 'tenure_months',
      'monthly_usage_gb', 'customer_satisfaction', 'number_of_services',
      'gender', 'contract_type', 'payment_method', 'internet_service',
              'phone_service', 'streaming_tv', 'streaming_movies']

# Create feature
dataframe X =
df[feature_cols].copy()
y_classification = df['churn'].copy() # For classification
y_regression = df['customer_lifetime_value'].copy() # For
regression
```

```
print(f"Features shape: {X.shape}")
print(f"Classification target shape:
{y_classification.shape}") print(f"Regression target
shape: {y_regression.shape}")
# Step 2: Label Encoding for categorical data
print("\n--- Step 2: Label Encoding for Categorical Data ---")
categorical_features =
X.select_dtypes(include=['object']).columns.tolist() numerical_features
= X.select_dtypes(include=[np.number]).columns.tolist()

print(f"Categorical features: {categorical_features}")
print(f"Numerical features: {numerical_features}")

# Manual encoding
label_encoders = {}
X_encoded = X.copy()

for col in
    categorical_features: le =
    LabelEncoder()
    X_encoded[col] = le.fit_transform(X[col].astype(str))
     label_encoders[col] = le

# Encode target for classification
le_target = LabelEncoder()
y_classification_encoded = le_target.fit_transform(y_classification)

# Final feature matrix X_final =

X_encoded.values print(f"Final feature matrix

shape: {X_final.shape}")

# Split data
X_train, X_test, y_train_clf, y_test_clf = train_test_split(
              X_final, y_classification_encoded, test_size=0.2, random_state=42,
                                                            stratify=y_classifica
)

X_train_reg, X_test_reg, y_train_reg, y_test_reg =
    train_test_split( X_final, y_regression, test_size=0.2,
    random_state=42
)

print(f"\nTrain set size: {X_train.shape[0]}")
print(f"Test set size: {X_test.shape[0]}")
```

```
# Step 3 & 4: Model Selection and Training
print("\n--- Step 3 & 4: Model Selection and Training ---")
print("Training at least 4 models for each task...\n")

# Classification Models
classification_models = {
    'Logistic Regression': LogisticRegression(random_state=42, max_iter=1000),
    'Random Forest': RandomForestClassifier(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingClassifier(n_estimators=100,
    random_state=42), 'SVM': SVC(probability=True, random_state=42)
}

# Regression Models
regression_models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(n_estimators=100, random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(n_estimators=100,
    random_state=42), 'SVR': SVR()
}

# Train and evaluate classification models
print("="*80)
print("CLASSIFICATION MODELS (Churn

Prediction)") print("="*80)

classification_results = {}

for name, model in classification_models.items():
    print(f"\n--- {name} ---")
    model.fit(X_train, y_train_clf)
    y_pred = model.predict(X_test)
    y_pred_proba = model.predict_proba(X_test)[:, 1] if hasattr(model,
    'predict_proba') else

    accuracy = accuracy_score(y_test_clf, y_pred) precision =
    precision_score(y_test_clf, y_pred, average='weighted')
    recall = recall_score(y_test_clf, y_pred,
    average='weighted')
    roc_auc = roc_auc_score(y_test_clf, y_pred_proba) if y_pred_proba is not
    None else None

    classification_results[name] = {
        'model': model,
        'accuracy': accuracy,
        'precision': precision,
        'recall': recall,
        'roc_auc': roc_auc,
        'y_pred': y_pred,
        'y_pred_proba': y_pred_proba
    }

    print(f"Accuracy:
    {accuracy:.4f}")
    print(f"Precision:
    {precision:.4f}")
    print(f"Recall: {recall:.4f}")
    if roc_auc:
        print(f"ROC-AUC: {roc_auc:.4f}")
# Train and evaluate regression models
print("\n" + "="*80)
print("REGRESSION MODELS (Customer Lifetime Value

Prediction)") print("="*80) regression_results = {}

for name, model in
    regression_models.items():
    print(f"\n--- {name} ---")
    model.fit(X_train_reg, y_train_reg)
    y_pred = model.predict(X_test_reg)

    mae = mean_absolute_error(y_test_reg, y_pred)
    mse = mean_squared_error(y_test_reg, y_pred)
    median_ae = median_absolute_error(y_test_reg,
    y_pred) r2 = r2_score(y_test_reg, y_pred)

    regression_results[name] = {
        'model': model,
        'mae': mae,
        'mse': mse,
        'median_ae': median_ae,
        'r2': r2,
        'y_pred': y_pred
    }

    print(f"Mean Absolute Error: {mae:.4f}")
    print(f"Mean Squared Error: {mse:.4f}")
    print(f"Median Absolute Error:
    {median_ae:.4f}") print(f"R² Score:
    {r2:.4f}")
```

```
# Step 6: Check if scaling affects accuracy
print("\n" + "="*80)
print("Step 6: Testing Effect of Scaling")
print("="*80)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_reg_scaled = scaler.fit_transform(X_train_reg)
X_test_reg_scaled = scaler.transform(X_test_reg)

print("\n--- Classification with Scaling ---")
scaled_clf_results = {} for name, model in
classification_models.items():
    model_scaled = type(model)(**model.get_params()) if hasattr(model,
    'get_params') else ty model_scaled.fit(X_train_scaled, y_train_clf) y_pred =
    model_scaled.predict(X_test_scaled) accuracy = accuracy_score(y_test_clf,
    y_pred) scaled_clf_results[name] = accuracy
    print(f"{name}: Accuracy = {accuracy:.4f} (Original:
    {classification_results[name]['accu

print("\n--- Regression with Scaling --
-") scaled_reg_results = {} for name,
model in regression_models.items():
    model_scaled = type(model)(**model.get_params()) if hasattr(model,
    'get_params') else ty model_scaled.fit(X_train_reg_scaled, y_train_reg)
    y_pred = model_scaled.predict(X_test_reg_scaled) r2 = r2_score(y_test_reg,
    y_pred) scaled_reg_results[name] = r2
    print(f"{name}: R² = {r2:.4f} (Original:
    {regression_results[name]['r2']:.4f})")

# Step 7: Check if normalization affects accuracy
print("\n" + "="*80)
print("Step 7: Testing Effect of Normalization")
print("="*80)

normalizer = MinMaxScaler()
X_train_norm = normalizer.fit_transform(X_train)
X_test_norm = normalizer.transform(X_test)
X_train_reg_norm = normalizer.fit_transform(X_train_reg)
X_test_reg_norm = normalizer.transform(X_test_reg)

print("\n--- Classification with Normalization ---") norm_clf_results = {} for
name, model in classification_models.items(): model_norm =
type(model)(**model.get_params()) if hasattr(model, 'get_params') else type
model_norm.fit(X_train_norm, y_train_clf) y_pred =
model_norm.predict(X_test_norm) accuracy = accuracy_score(y_test_clf, y_pred)
norm_clf_results[name] = accuracy
    print(f"{name}: Accuracy = {accuracy:.4f} (Original:
    {classification_results[name]['accu

print("\n--- Regression with Normalization ---") norm_reg_results = {} for
name, model in regression_models.items(): model_norm =
type(model)(**model.get_params()) if hasattr(model, 'get_params') else type
model_norm.fit(X_train_reg_norm, y_train_reg) y_pred =
model_norm.predict(X_test_reg_norm) r2 = r2_score(y_test_reg, y_pred)
norm_reg_results[name] = r2
    print(f"{name}: R² = {r2:.4f} (Original:
    {regression_results[name]['r2']:.4f})")
# Step 9: Visualize Results
print("\n" + "="*80)
print("Step 9: Visualizing Model Results")
print("="*80)

# Classification Results Visualization
plt.figure(figsize=(14, 10))

# 1. Classification Metrics Comparison
plt.subplot(2, 2, 1)
models = list(classification_results.keys()) accuracies =
[classification_results[m]['accuracy'] for m in models]
precisions = [classification_results[m]['precision'] for m
in models] recalls = [classification_results[m]['recall']
for m in models]

x = np.arange(len(models)) width = 0.25 plt.bar(x -
width, accuracies, width, label='Accuracy', alpha=0.8)
plt.bar(x, precisions, width, label='Precision',
alpha=0.8) plt.bar(x + width, recalls, width,
label='Recall', alpha=0.8) plt.xlabel('Models')
plt.ylabel('Score') plt.title('Classification Metrics
Comparison') plt.xticks(x, models, rotation=45,
ha='right')
plt.legend()
plt.grid(True, alpha=0.3, axis='y')

# 2. ROC Curve for models that support it
plt.subplot(2, 2, 2) for name, results in
classification_results.items():
    if results['y_pred_proba'] is not None:
        fpr, tpr, _ = roc_curve(y_test_clf, results['y_pred_proba'])
        plt.plot(fpr, tpr, label=f"{name}
        (AUC={results['roc_auc']:.3f})")
plt.plot([0, 1], [0, 1], 'k--',
label='Random') plt.xlabel('False
Positive Rate') plt.ylabel('True Positive
Rate') plt.title('ROC Curves')
plt.legend() plt.grid(True, alpha=0.3)

# 3. Regression Metrics Comparison
plt.subplot(2, 2, 3)
reg_models = list(regression_results.keys()) r2_scores
= [regression_results[m]['r2'] for m in reg_models]
mae_scores = [regression_results[m]['mae'] for m in
reg_models] # Normalize MAE for visualization (divide
by max) mae_normalized = [m / max(mae_scores) for m in
mae_scores]

x = np.arange(len(reg_models))
width = 0.35
plt.bar(x - width/2, r2_scores, width, label='R² Score', alpha=0.8)
plt.bar(x + width/2, mae_normalized, width, label='MAE (normalized)',
alpha=0.8) plt.xlabel('Models') plt.ylabel('Score')
plt.title('Regression Metrics Comparison') plt.xticks(x, reg_models,
rotation=45, ha='right') plt.legend()
plt.grid(True, alpha=0.3, axis='y')

# 4. Actual vs Predicted for Best Regression Model
plt.subplot(2, 2, 4) best_reg_model =
max(regression_results.items(), key=lambda x: x[1]['r2'])
best_name = best_reg_model[0] best_pred =
best_reg_model[1]['y_pred'] plt.scatter(y_test_reg, best_pred,
alpha=0.5)
plt.plot([y_test_reg.min(), y_test_reg.max()], [y_test_reg.min(),
y_test_reg.max()], 'r--', plt.xlabel('Actual Customer Lifetime Value')
plt.ylabel('Predicted Customer Lifetime Value') plt.title(f'Actual vs Predicted
({best_name})') plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('visualizations/model_results.png', dpi=300,
bbox_inches='tight') plt.close()

# Confusion Matrix for Best Classification Model
```

```
best_clf_model = max(classification_results.items(), key=lambda x:
x[1]['accuracy']) best_clf_name = best_clf_model[0] best_clf_pred =
best_clf_model[1]['y_pred']

plt.figure(figsize=(8, 6)) cm =
confusion_matrix(y_test_clf, best_clf_pred)
sns.heatmap(cm, annot=True, fmt='d',
cmap='Blues',
           xticklabels=['No Churn', 'Churn'],
           yticklabels=['No Churn', 'Churn'])
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.title(f'Confusion Matrix -
{best_clf_name}')
plt.tight_layout()
plt.savefig('visualizations/confusion_matrix.png', dpi=300,
bbox_inches='tight') plt.close()
print("\nModel result visualizations saved!")

# Summary print("\n"
+ "="*80)
print("PROJECT
SUMMARY")
print("="*80)
print(f"""
1. Dataset: {df.shape[0]} rows, {df.shape[1]} columns
2. Data Cleaning: Completed all 10 steps
3. Visualizations: Created 8 different types
4. Classification Models: Trained
   {len(classification_models)} models
5. Regression Models: Trained {len(regression_models)}
   models
6. Best Classification Model: {best_clf_name}
   (Accuracy: {classification_results[best_clf_na
7. Best Regression Model: {best_reg_model[0]} (R²:
   {best_reg_model[1]['r2']:.4f})
8. Scaling Effects: Tested and compared
9. Normalization Effects: Tested and compared

All results and visualizations have been saved!
""")

print("="*80)
print("PROJECT COMPLETED SUCCESSFULLY!")
print("="*80)
```

Section 4: Visualization Results

Overview

All visualizations were successfully generated and saved in the `visualizations/` folder. Below are descriptions and snapshots of each visualization:

1. Line Plot

File: `visualizations/1_line_plot.png`
Description: Shows the relationship between customer tenure (in months) and average monthly charges. This visualization helps identify trends in pricing based on customer loyalty and length of service.

Line Plot *Figure 1: Line Plot - Average Monthly Charges by Tenure*

2. Area Plot

File: `visualizations/2_area_plot.png`
Description: Displays the distribution of churn (Yes/No) across different contract types using a stacked area chart. This helps identify which contract types have higher churn rates and visualize the cumulative distribution.

Area Plot *Figure 2: Area Plot - Churn Distribution by Contract Type*

3. Histogram

File: `visualizations/3_histogram.png`
Description: Age distribution of customers with 30 bins. Shows the demographic distribution of the customer base, helping understand the age profile of customers.

Histogram *Figure 3: Histogram - Age Distribution*

4. Bar Chart

File: `visualizations/4_bar_chart.png`
Description: Bar chart showing the count of customers who churned vs. those who did not. Provides a clear view of the churn distribution and helps understand the class balance in the dataset.

Bar Chart *Figure 4: Bar Chart - Churn Distribution*

5. Pie Chart

File: `visualizations/5_pie_chart.png`
Description: Pie chart showing the percentage distribution of different contract types. Helps visualize the proportion of each contract type in the dataset, making it easy to see which contract types are most common.

Pie Chart *Figure 5: Pie Chart - Contract Type Distribution*

6. Box Plot

File: `visualizations/6_box_plot.png`
Description: Box plots for monthly charges, total charges, and tenure months. Shows the distribution, median, quartiles, and outliers for these key numerical features. This visualization is crucial for understanding data spread and identifying outliers.

Box Plot *Figure 6: Box Plot - Distribution of Numerical Features*

7. Scatter Plot

File: `visualizations/7_scatter_plot.png`
Description: Scatter plot of monthly charges vs. total charges, colored by churn status (red for churned, blue for retained). Helps identify patterns in spending behavior related to churn and shows the relationship between these two variables.
Scatter Plot *Figure 7: Scatter Plot - Monthly Charges vs Total Charges (colored by Churn)*

8. Bubble Plot

File: `visualizations/8_bubble_plot.png`
Description: Bubble plot showing age vs. monthly charges, where bubble size represents tenure months. Different colors indicate churn status. This multi-dimensional visualization helps identify complex relationships between age, charges, tenure, and churn.

Bubble Plot *Figure 8: Bubble Plot - Age vs Monthly Charges (bubble size = tenure)*

9. Model Results Comparison

File: `visualizations/model_results.png`
Description: Comprehensive comparison of all models including: - Classification metrics comparison (Accuracy, Precision, Recall) for all 4 classification models - ROC curves for classification models showing model performance -
Regression metrics comparison (R^2 Score, MAE) for all 4 regression models Actual vs. Predicted scatter plot for the best regression model

Model Results *Figure 9: Model Results Comparison*

10. Confusion Matrix

File: `visualizations/confusion_matrix.png`
Description: Confusion matrix for the best classification model (Random Forest), showing true positives, true negatives, false positives, and false negatives. This visualization helps understand the model's classification performance in detail.

Confusion Matrix *Figure 10: Confusion Matrix - Best Classification Model (Random Forest)*

Summary of Visualizations

All 8 required visualization types have been successfully created: 1. Line plots 2. Area plots 3. Histogram 4. Bar charts 5. Pie charts 6. Box plots 7. Scatter plots 8. Bubble plots

Additionally, model evaluation visualizations (ROC curves, confusion matrix, and model comparison charts) have been included to provide comprehensive insights into model performance.

Conclusion

This project successfully implemented a complete machine learning pipeline for customer churn prediction and customer lifetime value prediction. The project covered all required components:

- Comprehensive data cleaning (all 10 steps)
- Exploratory data analysis
- 8 different visualization types
- Multiple machine learning models (4 classification + 4 regression)
- Complete model evaluation with all required metrics
- Analysis of scaling and normalization effects

The project demonstrates proficiency in data science workflows, machine learning model development, and result visualization, meeting all requirements of the AIE 121 Machine Learning Final Project.

End of Report

Note: To convert this markdown file to PDF, you can use: - VS Code with "Markdown PDF" extension - Pandoc: `pandoc project_report.md -o project_report.pdf` - Online converters like markdowntopdf.com - Microsoft Word (open the .md file and save as PDF)