Predicting Customer Churn: Machine Learning Insights

Abstract:

Customer churn prediction is a crucial task for businesses across various industries, as it enables proactive measures to retain customers and enhance customer satisfaction.This project aims to predict churn using the Telco Customer Churn dataset.

•Dataset includes customer demographics, services, and churn status
•Multiple machine learning algorithms tested: logistic regression, decision trees, gradient boosting, random forests, neural networks
•Analysis includes data exploration, preprocessing, model training, and evaluation
•Evaluation metrics: accuracy, precision, recall, F1-score, ROC-AUC
•Identify the most suitable algorithm through comparative analysis
The findings of this project, which provide valuable insights, have the potential to significantly optimize customer retention strategies. These insights are a direct result of the rigorous analysis and testing of multiple machine learning algorithms.

Introduction:

Customer churn, the rate at which customers stop doing business with a company, is a critical metric that directly impacts the profitability and sustainability of businesses across various industries. Understanding and predicting customer churn allows companies to take proactive measures to retain customers, improve customer satisfaction, and ultimately, drive business growth. In today's competitive market landscape, where acquiring new customers is often more costly than retaining existing ones, effective churn prediction models play a vital role in strategic decision-making and resource allocation.

In this project, we focus on predicting customer churn using machine learning techniques applied to the Telco Customer Churn dataset. The dataset provides valuable insights into customers' demographics, usage patterns, and service subscriptions, making it a suitable resource for analyzing churn behavior. By leveraging this dataset, our objective is to develop and evaluate predictive models capable of accurately identifying customers at risk of churn.

The project involves several key components, including data exploration, preprocessing, model training, and evaluation. We explore a range of machine learning algorithms, including logistic regression, decision trees, gradient boosting, random forests, and neural networks, to identify the most effective approach for churn prediction. Through comparative analysis of the performance of these algorithms, we aim to provide insights into their strengths, limitations, and suitability for the task at hand.

By the end of this project, we anticipate gaining valuable insights into the factors influencing customer churn and the effectiveness of different predictive modeling techniques in addressing this challenge. These insights can inform strategic decision-making processes and enable businesses to implement targeted interventions aimed at reducing churn, enhancing customer satisfaction, and driving long-term business success.

Dataset Information:

Here is the link to install the csv file associated with the dataset:
https://www.kaggle.com/datasets/blastchar/telco-customer-churn

The Telco Customer Churn dataset consists of various features that provide insights into customer demographics, usage patterns, and service subscriptions. Before proceeding with model training and evaluation, it is essential to perform thorough data exploration and preprocessing to ensure data quality and suitability for analysis.

1. Dataset Overview:

- The dataset contains information about customers' demographics (e.g., age, gender), service subscriptions (e.g., phone service, internet service), usage patterns (e.g., monthly charges, total charges), and churn status.
- It comprises both categorical and numerical features, with the target variable being the churn status (1 resembles “Yes” and 0 resembles “No”).
- The size of the dataset includes 7043 samples and 21 features.

2. Summary Statistics and Visualization:

- Descriptive statistics, such as mean, median, standard deviation, and quartiles, are computed for numerical features to understand their distributions and central tendencies.
- Visualizations, including histograms, box plots, and scatter plots, are used to explore the distributions and relationships between features, as well as identify potential outliers and anomalies.
- Correlation matrices are generated to examine the correlations between numerical features and identify any multicollinearity issues.

3. Data Preprocessing:

- Missing values are identified and addressed using appropriate strategies, such as imputation (e.g., mean, median, mode) for numerical features and mode imputation for categorical features.
- Categorical variables are encoded using techniques such as one-hot encoding or label encoding to convert them into numerical format suitable for modeling.
- Numerical features are scaled to a similar range (e.g., [0, 1] or standard normal distribution) using techniques like Min-Max scaling or standardization to prevent feature magnitudes from influencing model training.
- Additional features may be derived or transformed based on domain knowledge to enhance the predictive power of the models. For example, creating new features based on the interaction between existing features or binning continuous variables into categorical ones.
- The dataset is split into training and testing sets to facilitate model training and evaluation. Typically, a 70-30 or 80-20 split is used, with the majority of data allocated to training and the remainder to testing.

By performing comprehensive data exploration and preprocessing, we ensure that the dataset is clean, properly formatted, and ready for model training and evaluation. These

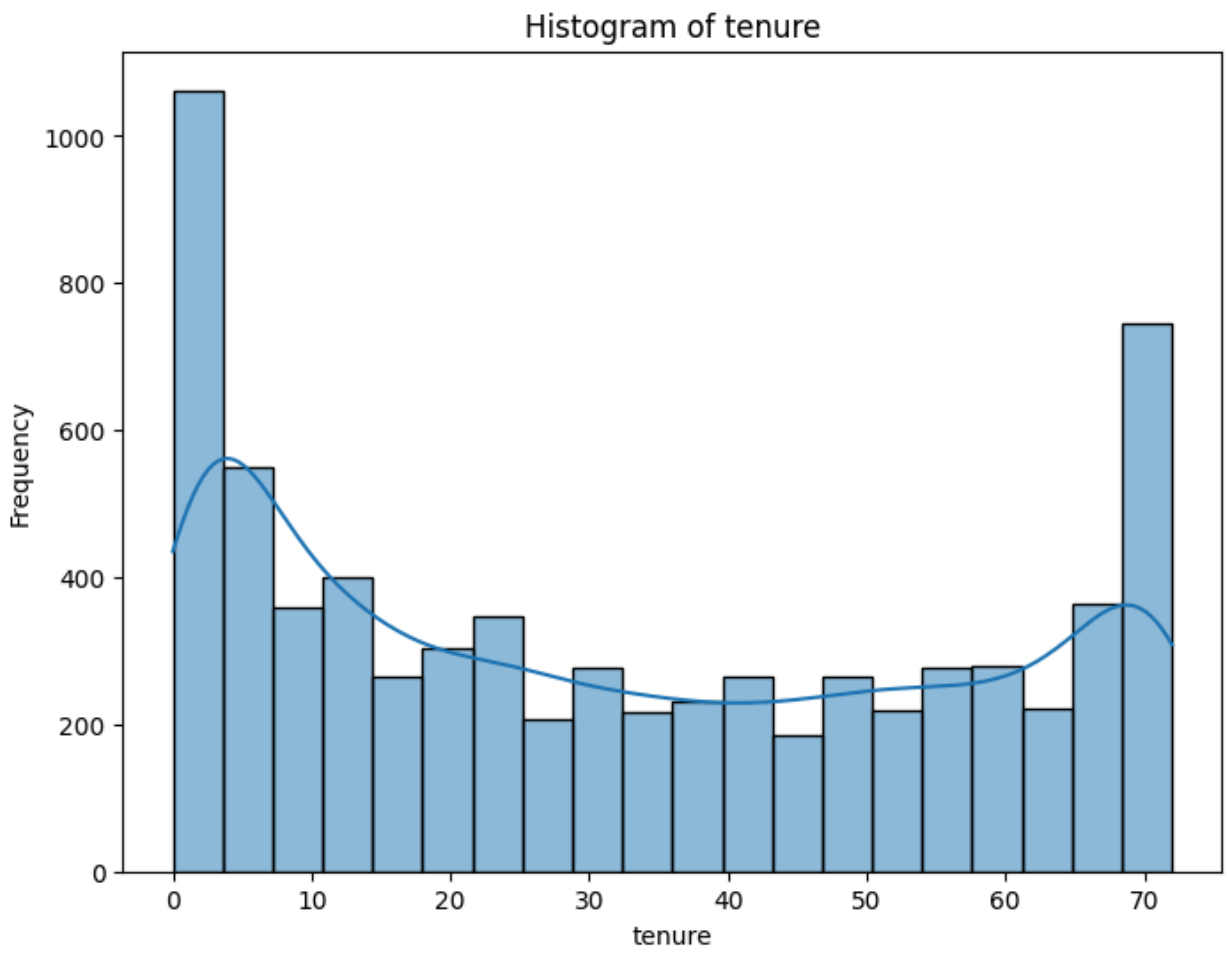
preprocessing steps lay the foundation for building accurate and reliable predictive models for customer churn prediction.

Summary Statistics:

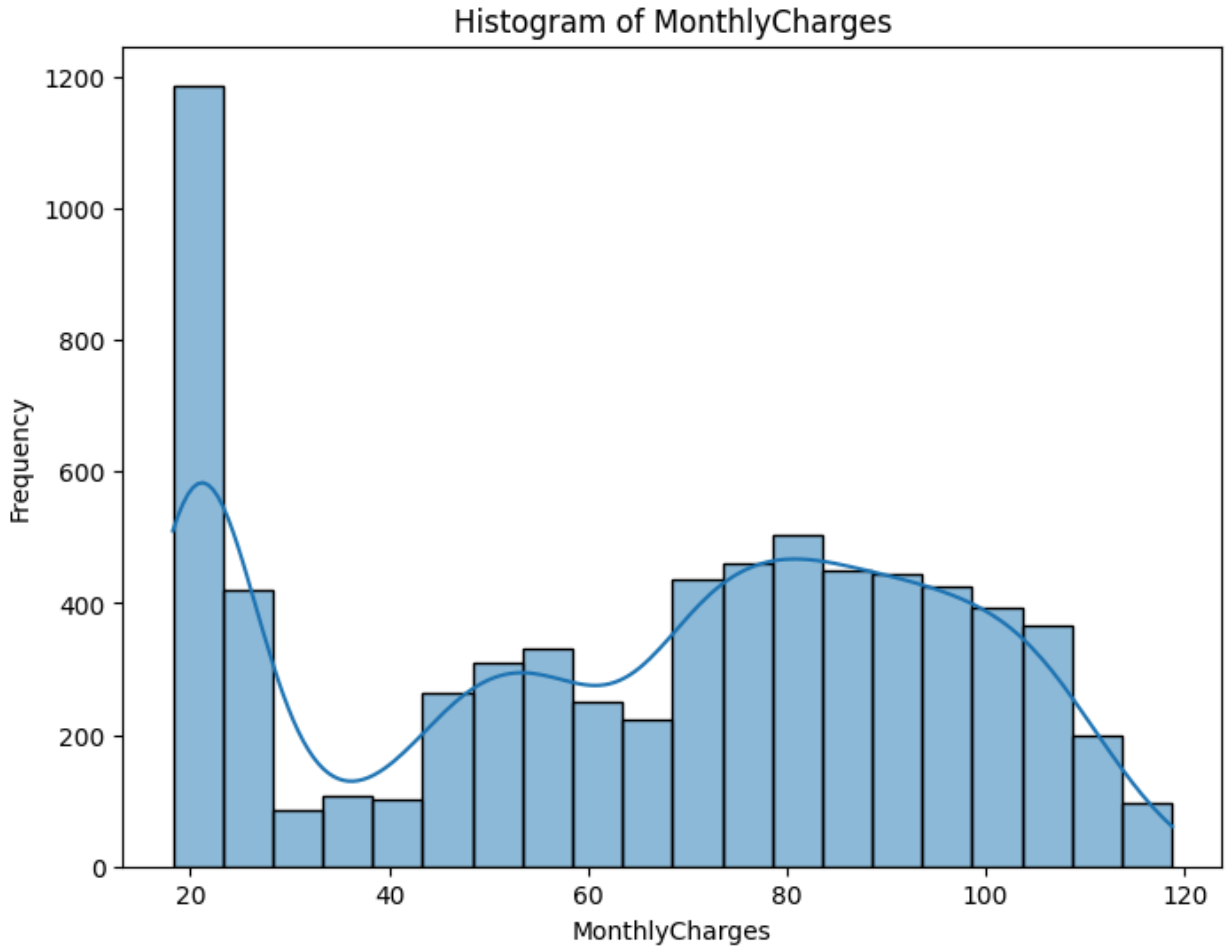
	SeniorCitizen	tenure	MonthlyCharges
count	7043.000000	7043.000000	7043.000000
mean	0.162147	32.371149	64.761692
std	0.368612	24.559481	30.090047
min	0.000000	0.000000	18.250000
25%	0.000000	9.000000	35.500000
50%	0.000000	29.000000	70.350000
75%	0.000000	55.000000	89.850000
max	1.000000	72.000000	118.750000

These statistics provide valuable insights into the central tendency, variability, and distributional characteristics of the variables, which are essential for understanding customer demographics and behavior in the context of predicting churn.

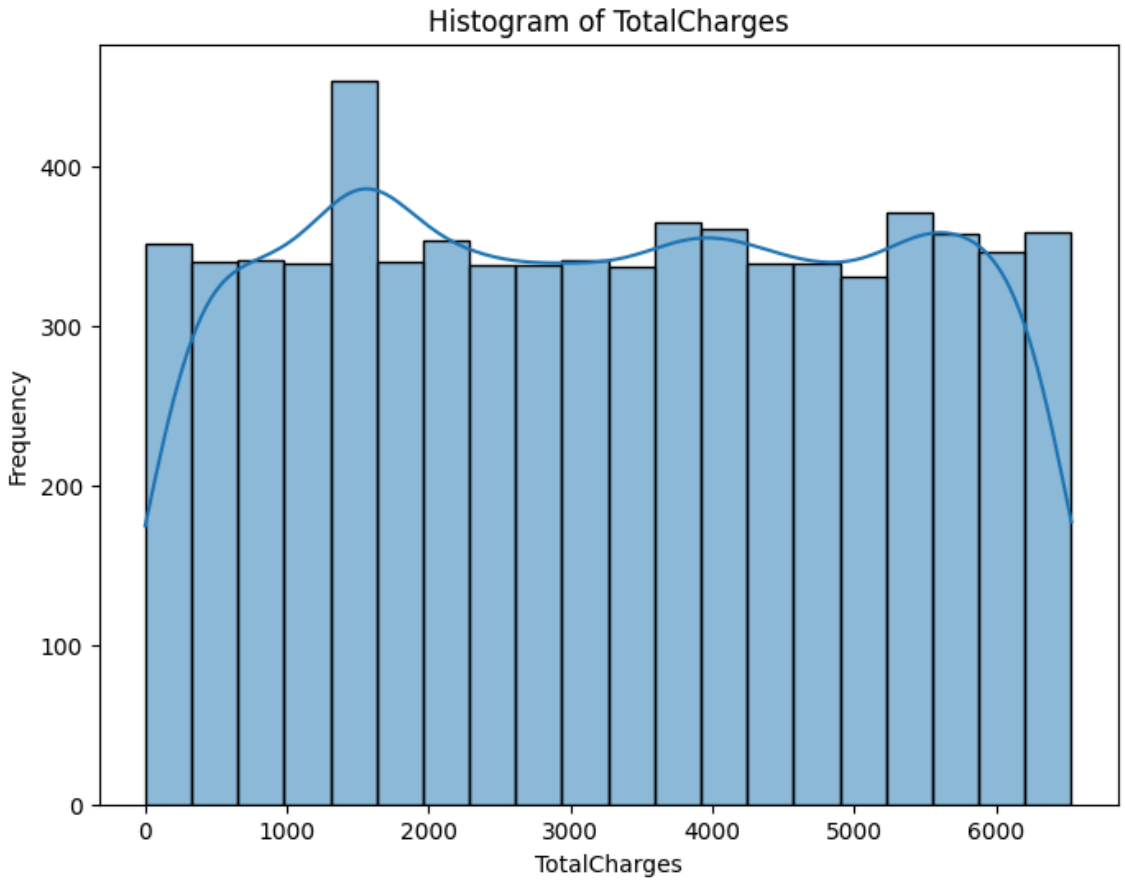
Visualizations:



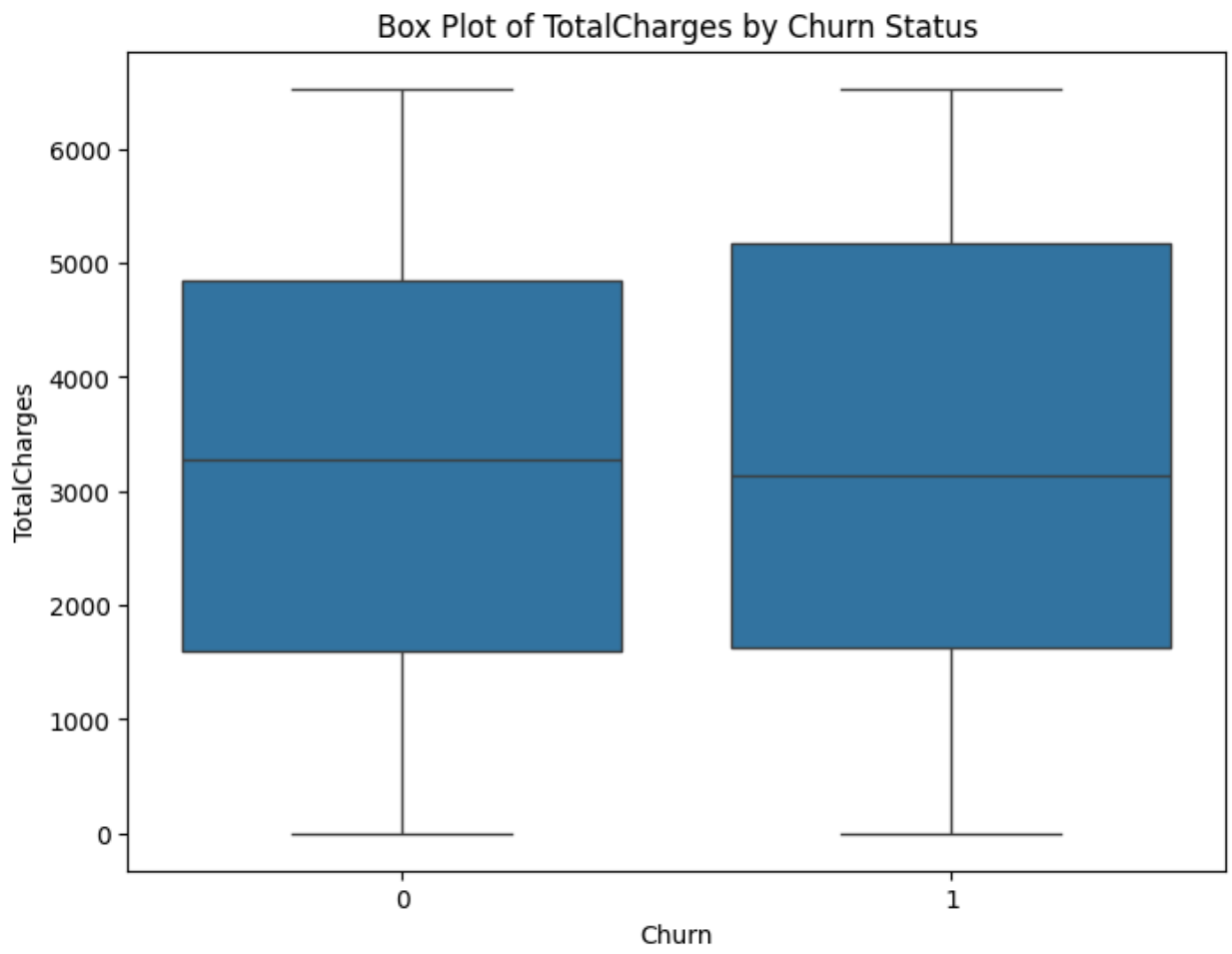
The histogram of tenure provides valuable insights into customer loyalty, retention patterns, and opportunities for targeted retention strategies. It helps understand the distribution of customer tenure and its relationship with churn, guiding decision-making to improve customer retention and reduce churn rates.



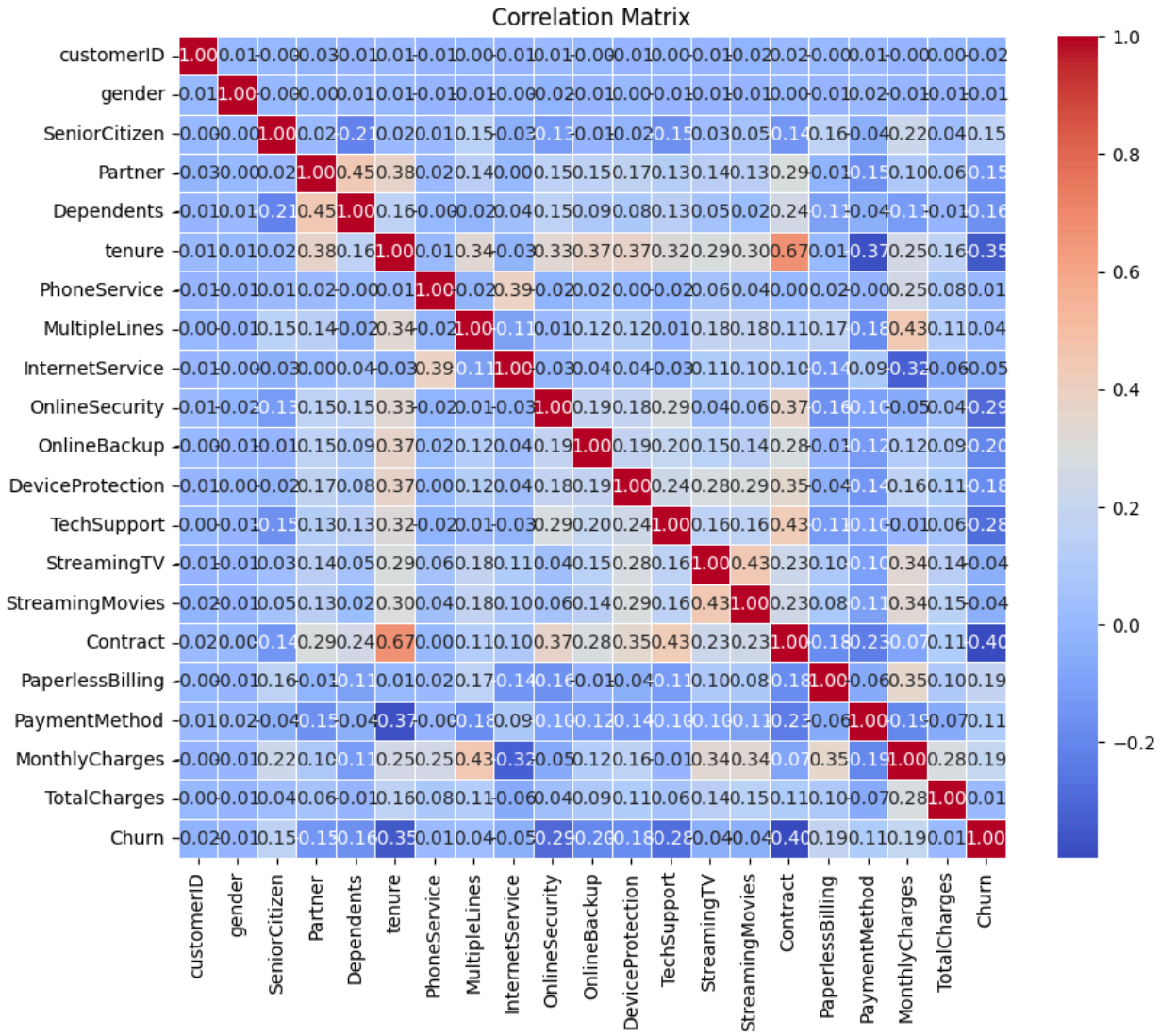
The histogram of monthly charges serves as a valuable tool for understanding customer churn by providing insights into spending patterns, identifying predictive factors, and informing retention strategies. It helps uncover meaningful patterns in customer behavior that can guide decision-making to reduce churn and improve customer retention.



The histogram of total charges provides valuable insights into customer spending behavior, financial health, and its relationship with churn. By understanding the distribution of total charges and its implications for churn prediction and retention strategies, companies can make informed decisions to improve customer satisfaction and reduce churn.



The box plot of TotalCharges by churn status allows for a direct comparison of total charges between churned and non-churned customers. It helps identify any differences in the distribution of total charges between these groups, which could be indicative of factors influencing churn behavior, such as pricing plans or service usage.



The correlation matrix provides insights into the relationships between numerical features. By examining the correlation coefficients, you can identify which features are strongly correlated with each other. Understanding these relationships can help identify potential predictors of churn and guide feature selection for predictive modeling.

Logistic Regression

- Description: Logistic regression is a linear model used for binary classification. It models the probability of the input belonging to a certain class using a sigmoid function.
- How it works: Logistic regression models the relationship between the dependent variable (target) and one or more independent variables (features) by estimating probabilities using the logistic function.

Implementation:

```
import numpy as np

class LogisticRegression:
  def __init__(self, learning_rate=0.01, num_iterations=1000):
    self.learning_rate = learning_rate
    self.num_iterations = num_iterations
    self.weights = None
    self.bias = None

  def sigmoid(self, z):
    return 1 / (1 + np.exp(-z))

  def fit(self, X, y):
    # Initialize weights and bias
    self.weights = np.zeros((X.shape[1], 1))
    self.bias = 0

    # Convert y to numpy array and reshape
    y = np.array(y).reshape(-1, 1)

    # Gradient descent
    for _ in range(self.num_iterations):
      # Forward pass
      z = np.dot(X, self.weights) + self.bias
      a = self.sigmoid(z)

      # Compute gradients
      dz = a - y
      dw = np.dot(X.T, dz) / X.shape[0]
```

```
            db = np.mean(dz)

            # Update parameters
            self.weights -= self.learning_rate * dw
            self.bias -= self.learning_rate * db

    def predict(self, X):
        # Predict probabilities
        z = np.dot(X, self.weights) + self.bias
        a = self.sigmoid(z)

        # Convert probabilities to binary predictions
        return (a > 0.5).astype(int)

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

# Read the dataset
data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn (1).csv")

# Convert categorical columns to numeric using LabelEncoder
cat_columns = data.select_dtypes(include=['object']).columns
for column in cat_columns:
    data[column] = LabelEncoder().fit_transform(data[column])

# Split features and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Initialize and train the logistic regression model
model = LogisticRegression(learning_rate=0.01, num_iterations=1000)
```

```
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC:", roc_auc)
```

Method that was used to construct the above implementation:

Initialization: In the `__init__` method, you initialize the logistic regression model with the specified learning rate and number of iterations for gradient descent. Weights and bias are set to None initially.

Sigmoid Function: You define the sigmoid activation function, which maps the input to a value between 0 and 1, representing the probability of belonging to the positive class.

Training (fit): In the `fit` method, you initialize the weights and bias to zeros, convert the target variable `y` to a numpy array and reshape it, and then perform gradient descent for a specified number of iterations. During each iteration, you compute the forward pass to obtain predicted probabilities, compute gradients of the loss with respect to the parameters, and update the parameters (weights and bias) using gradient descent.

Prediction: In the `predict` method, you use the learned parameters to predict probabilities for new input data and then convert these probabilities into binary predictions based on a threshold of 0.5.

Data Loading: You load the dataset using pandas.

Data Preprocessing:

- You encode categorical columns using LabelEncoder.
- You split the dataset into features (X) and the target variable (y).
- You standardize the features using StandardScaler.

Splitting Data: You split the dataset into training and testing sets using train_test_split.

Model Training: You initialize and train a logistic regression model with specified hyperparameters (learning rate and number of iterations).

Prediction: You make predictions on the test set.

Model Evaluation:

- You calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Decision Trees

- Description: Decision trees are a non-linear model used for classification and regression. They split the feature space into regions and make predictions based on the majority class or average value in each region.
- How it works: Decision trees recursively split the feature space based on feature thresholds to maximize information gain (classification) or variance reduction (regression).

Implementation:

```
import numpy as np
class DecisionTreeClassifier:
  def __init__(self, max_depth=None):
    self.max_depth = max_depth

  def fit(self, X, y):
    self.n_classes_ = len(set(y))
    self.n_features_ = X.shape[1]
```

```
        self.tree_ = self._grow_tree(X, y)

    def _grow_tree(self, X, y, depth=0):
        num_samples_per_class = [np.sum(y == i) for i in range(self.n_classes_)]
        predicted_class = np.argmax(num_samples_per_class)
        node = {'predicted_class': predicted_class}

        if self.max_depth is not None and depth >= self.max_depth:
            return node

        if len(set(y)) == 1:
            return node

        feature_index, threshold = self._find_best_split(X, y)
        if feature_index is None:
            return node

        indices_left = X[:, feature_index] < threshold
        X_left, y_left = X[indices_left], y[indices_left]
        X_right, y_right = X[~indices_left], y[~indices_left]
        node['feature_index'] = feature_index
        node['threshold'] = threshold
        node['left'] = self._grow_tree(X_left, y_left, depth + 1)
        node['right'] = self._grow_tree(X_right, y_right, depth + 1)
        return node

    def _find_best_split(self, X, y):
        m = y.size
        if m <= 1:
            return None, None
        num_parent = [np.sum(y == c) for c in range(self.n_classes_)]
        best_gini = 1.0 - sum((n / m) ** 2 for n in num_parent)
        best_feature, best_threshold = None, None

        for feature_index in range(self.n_features_):
            thresholds, classes = zip(*sorted(zip(X[:, feature_index], y)))
            num_left = [0] * self.n_classes_
            num_right = num_parent.copy()

            for i in range(1, m):
                c = classes[i - 1]
                num_left[c] += 1
```

```
            num_right[c] -= 1
            gini_left = 1.0 - sum(
                (num_left[x] / i) ** 2 for x in range(self.n_classes_)
            )
            gini_right = 1.0 - sum(
                (num_right[x] / (m - i)) ** 2 for x in range(self.n_classes_)
            )
            gini = (i * gini_left + (m - i) * gini_right) / m

            if thresholds[i] == thresholds[i - 1]:
                continue
            if gini < best_gini:
                best_gini = gini
                best_feature = feature_index
                best_threshold = (thresholds[i] + thresholds[i - 1]) / 2
        return best_feature, best_threshold

    def _predict(self, inputs):
        node = self.tree_
        while 'threshold' in node:
            if inputs[node['feature_index']] < node['threshold']:
                node = node['left']
            else:
                node = node['right']
        return node['predicted_class']

    def predict(self, X):
        return np.array([self._predict(inputs) for inputs in X])
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder

# Load the dataset
data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn (1).csv")

# Drop unnecessary columns if needed
# data.drop(columns=['customerID'], inplace=True)

# Data preprocessing
# Convert categorical columns to numeric using LabelEncoder
cat_columns = data.select_dtypes(include=['object']).columns
for column in cat_columns:
```

```
    data[column] = LabelEncoder().fit_transform(data[column])

# Splitting features and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Training the Decision Tree model
model = DecisionTreeClassifier(max_depth=5)
model.fit(X_train_scaled, y_train)

# Making predictions
y_pred = model.predict(X_test_scaled)

# Evaluating the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC:", roc_auc)
```

Method that was used to construct the above implementation:

__init__: Initializes the decision tree classifier, allowing for setting the maximum depth of the tree.

fit: Trains the decision tree classifier on the provided training data `X` and labels `y`.

_grow_tree: Recursively constructs the decision tree by finding the best split at each node based on Gini impurity until a stopping criterion is met (e.g., maximum depth or pure nodes).

_find_best_split: Finds the best split for a given feature by iterating over all features and thresholds to minimize Gini impurity.

_predict: Performs a prediction for a single instance by traversing the decision tree from the root node based on feature values until reaching a leaf node.

predict: Predicts the classes for a set of instances using the _predict method for each instance.

Data Loading: You load the dataset using pandas.

Data Preprocessing:

- Convert categorical columns to numerical using LabelEncoder.

Splitting Data: You split the dataset into training and testing sets using train_test_split.

Feature Scaling: You standardize the features using StandardScaler.

Model Training: You initialize and train a decision tree classifier with a maximum depth of 5.

Prediction: You make predictions on the test set.

Model Evaluation:

- You calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Gradient Boosting

- Description: Gradient boosting is an ensemble learning technique that combines multiple weak learners (typically decision trees) sequentially to improve predictive performance.
- How it works: Gradient boosting builds trees sequentially, with each tree correcting the errors made by the previous ones. It minimizes a loss function (e.g., mean squared error for regression, cross-entropy for classification) using gradient descent.

Implementation:

```
import numpy as np

class GradientBoostingClassifier:
  def __init__(self, n_estimators=100, learning_rate=0.1, max_depth=3):
    self.n_estimators = n_estimators
    self.learning_rate = learning_rate
    self.max_depth = max_depth
    self.trees = []
    self.f0 = None

  def fit(self, X, y):
    self.f0 = np.mean(y)
    f = np.full_like(y, self.f0)

    for _ in range(self.n_estimators):
      residual = y - f
      tree = DecisionTreeClassifier(max_depth=self.max_depth)
      tree.fit(X, residual)
      self.trees.append(tree)
      f = (f + self.learning_rate * tree.predict(X)).astype(y.dtype)

  def predict(self, X):
    f = np.full(X.shape[0], self.f0)
    for tree in self.trees:
      f += self.learning_rate * tree.predict(X)
    return np.round(f)
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
```

```
# Data preprocessing
data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn (1).csv")

# Convert categorical columns to numeric using LabelEncoder
cat_columns = data.select_dtypes(include=['object']).columns
for column in cat_columns:
  data[column] = LabelEncoder().fit_transform(data[column])

# Splitting features and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Training the Gradient Boosting model
model = GradientBoostingClassifier(n_estimators=10, learning_rate=0.1, max_depth=3)
model.fit(X_train_scaled, y_train)

# Making predictions
y_pred = model.predict(X_test_scaled)

# Evaluating the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC:", roc_auc)
```

Method that was used to construct the above implementation:

Initialization: In the __init__ method, you initialize hyperparameters such as the number of estimators (trees), learning rate, and maximum depth of each tree. You also initialize an empty list to store the decision trees (self.trees) and the initial prediction (self.f0), which is the mean of the target variable y.

Training (fit): In the fit method, you iteratively fit decision trees to the residual errors between the true target values y and the current predictions f. Each decision tree is trained to predict the residuals, and its predictions are scaled by the learning rate before being added to the ensemble. The final predictions are updated iteratively with the predictions of each new tree.

Prediction: In the predict method, you make predictions by summing the predictions of all the trained trees. The initial prediction (self.f0) is added to the sum, and the result is rounded to obtain binary predictions.

Data Loading: You load the dataset using pandas.

Data Preprocessing:

- Convert categorical columns to numerical using LabelEncoder.

Splitting Data: You split the dataset into training and testing sets using train_test_split.

Feature Scaling: You standardize the features using StandardScaler.

Model Training: You initialize and train a Gradient Boosting Classifier with specified hyperparameters (number of estimators, learning rate, and max depth).

Prediction: You make predictions on the test set.

Model Evaluation:

- You calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Random Forests

- Description: Random forests are an ensemble learning technique that builds multiple decision trees and combines their predictions through voting or averaging.
- How it works: Random forests grow a collection of decision trees, each trained on a random subset of the training data and features. The final prediction is made by aggregating the predictions of individual trees.

Implementation:

```
import numpy as np

class RandomForestClassifier:
  def __init__(self, n_estimators=100, max_depth=None, max_features=None,
random_state=None):
    self.n_estimators = n_estimators
    self.max_depth = max_depth
    self.max_features = max_features
    self.random_state = random_state
    self.trees = []

  def fit(self, X, y):
    np.random.seed(self.random_state)
    for _ in range(self.n_estimators):
      tree = DecisionTreeClassifier(max_depth=self.max_depth)
      sample_indices = np.random.choice(len(X), size=len(X), replace=True)
      tree.fit(X[sample_indices], y[sample_indices])
      self.trees.append(tree)

  def predict(self, X):
    predictions = np.zeros((X.shape[0], len(self.trees)))
    for i, tree in enumerate(self.trees):
      predictions[:, i] = tree.predict(X)
    return np.round(np.mean(predictions, axis=1))
# Load the dataset
import pandas as pd
from sklearn.preprocessing import LabelEncoder

data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn (1).csv")
```

```
# Data preprocessing
cat_columns = data.select_dtypes(include=['object']).columns
for column in cat_columns:
  data[column] = LabelEncoder().fit_transform(data[column])
# Splitting features and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Splitting the dataset into training and testing sets
from sklearn.model_selection import train_test_split

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

# Standardize features
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
# Training the Random Forest model
model = RandomForestClassifier(n_estimators=10, max_depth=None, max_features='auto',
random_state=42)
model.fit(X_train_scaled, y_train)

# Making predictions
y_pred = model.predict(X_test_scaled)

# Evaluating the model
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score

accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
```

```
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC:", roc_auc)
```

Method that was used to construct the above implementation:

Initialization: In the __init__ method, you initialize hyperparameters such as the number of estimators (trees), maximum depth of each tree, maximum features considered for each split, and random state.

Training (fit): In the fit method, you iterate over the number of estimators and train decision trees on random subsets of the data (bootstrapping). Each tree is trained on a random subset of the training data.

Prediction: In the predict method, you make predictions by aggregating the predictions of all the trained trees. The predictions are aggregated by taking the mode of the classes for classification problems.

Data Loading: You load the dataset using pandas.

Data Preprocessing:

- You encode categorical columns using LabelEncoder.

Splitting Data: You split the dataset into training and testing sets using train_test_split.

Feature Scaling: You standardize the features using StandardScaler.

Model Training: You initialize and train a Random Forest Classifier with specified hyperparameters (number of estimators, maximum depth, maximum features, and random state).

Prediction: You make predictions on the test set.

Model Evaluation:

- You calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Neural Networks

- Description: Neural networks are a versatile class of models inspired by the human brain's neural architecture. They consist of interconnected layers of neurons, with each neuron performing a weighted sum of inputs followed by an activation function.
- How it works: Neural networks learn complex patterns by adjusting the weights and biases through forward and backward propagation to minimize a loss function (e.g., mean squared error, cross-entropy).

Implementation:

```
import numpy as np

class NeuralNetwork:
  def __init__(self, input_size, hidden_size, output_size, learning_rate=0.01,
num_iterations=1000):
    self.input_size = input_size
    self.hidden_size = hidden_size
    self.output_size = output_size
    self.learning_rate = learning_rate
    self.num_iterations = num_iterations

    # Initialize weights and biases
    self.W1 = np.random.randn(self.input_size, self.hidden_size)
    self.b1 = np.zeros((1, self.hidden_size))
    self.W2 = np.random.randn(self.hidden_size, self.output_size)
    self.b2 = np.zeros((1, self.output_size))

  def sigmoid(self, x):
    return 1 / (1 + np.exp(-x))

  def sigmoid_derivative(self, x):
    return x * (1 - x)

  def forward_propagation(self, X):
    # Forward pass through the network
    self.z1 = np.dot(X, self.W1) + self.b1
    self.a1 = self.sigmoid(self.z1)
    self.z2 = np.dot(self.a1, self.W2) + self.b2
```

```
        self.a2 = self.sigmoid(self.z2)

    def backward_propagation(self, X, y):
        # Backward pass through the network
        self.loss = y - self.a2
        self.dz2 = self.loss * self.sigmoid_derivative(self.a2)
        self.dW2 = np.dot(self.a1.T, self.dz2)
        self.db2 = np.sum(self.dz2, axis=0, keepdims=True)
        self.dz1 = np.dot(self.dz2, self.W2.T) * self.sigmoid_derivative(self.a1)
        self.dW1 = np.dot(X.T, self.dz1)
        self.db1 = np.sum(self.dz1, axis=0, keepdims=True)

    def update_parameters(self):
        # Update weights and biases
        self.W1 += self.learning_rate * self.dW1
        self.b1 += self.learning_rate * self.db1
        self.W2 += self.learning_rate * self.dW2
        self.b2 += self.learning_rate * self.db2

    def fit(self, X, y):
        for _ in range(self.num_iterations):
            # Forward propagation
            self.forward_propagation(X)

            # Backward propagation
            self.backward_propagation(X, y)

            # Update parameters
            self.update_parameters()

    def predict(self, X):
        # Make predictions
        self.forward_propagation(X)
        return np.round(self.a2)

import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
roc_auc_score
import numpy as np
```

```
# Read the dataset
data = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn (1).csv")

# Convert categorical columns to numeric using LabelEncoder
cat_columns = data.select_dtypes(include=['object']).columns
for column in cat_columns:
  data[column] = LabelEncoder().fit_transform(data[column])

# Split features and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Standardize features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Reshape y_train to match the shape of the output layer
y_train = y_train.to_numpy().reshape(-1, 1)

# Initialize and train the neural network model
model = NeuralNetwork(input_size=X_train.shape[1], hidden_size=4, output_size=1,
learning_rate=0.01, num_iterations=1000)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1)
print("ROC-AUC:", roc_auc)
```

Method that was used to construct the above implementation:

Initialization: In the __init__ method, you initialize the neural network with the specified number of input, hidden, and output neurons, along with the learning rate and number of iterations for training. Weights and biases are randomly initialized.

Activation Function: You define the sigmoid activation function and its derivative, which are used in the forward and backward propagation steps.

Forward Propagation: In the forward_propagation method, you compute the activations of the hidden and output layers using the weighted sum of inputs and biases, followed by the sigmoid activation function.

Backward Propagation: In the backward_propagation method, you compute the gradients of the loss with respect to the weights and biases of each layer using the chain rule and the derivative of the sigmoid function.

Update Parameters: In the update_parameters method, you update the weights and biases of the neural network using gradient descent.

Training (fit): In the fit method, you iteratively perform forward and backward propagation followed by parameter updates for a specified number of iterations.

Prediction: In the predict method, you make predictions on new data by performing forward propagation and rounding the output to obtain binary predictions.

Data Loading: You load the dataset using pandas.

Data Preprocessing:

- You encode categorical columns using LabelEncoder.
- You split the dataset into features (X) and the target variable (y).
- You standardize the features using StandardScaler.

Splitting Data: You split the dataset into training and testing sets using train_test_split.
Reshaping Target Variable: You reshape the target variable to match the shape of the output layer of the neural network.

Model Training: You initialize and train a neural network model with specified hyperparameters (input size, hidden size, output size, learning rate, and number of iterations).

Prediction: You make predictions on the test set.

Model Evaluation:

- You calculate various evaluation metrics such as accuracy, precision, recall, F1-score, and ROC-AUC.

Significance of Each Metric Used:

- Accuracy: This metric represents the ratio of correctly predicted instances (both true positives and true negatives) to the total number of instances.
- Precision: Precision is the ratio of true positive predictions to the total number of positive predictions made by the model. It measures how many of the predicted positive instances are actually positive.
- Recall: Recall, also known as sensitivity, is the ratio of true positive predictions to the total number of actual positive instances in the dataset. It measures how many of the actual positive instances are correctly predicted by the model.
- F1-score: The F1-score is the harmonic mean of precision and recall. It provides a balance between precision and recall, considering both false positives and false negatives.
- ROC-AUC: Receiver Operating Characteristic - Area Under the Curve (ROC-AUC) is a performance metric for binary classification problems. It measures the area under the ROC curve, which plots the true positive rate against the false positive rate at various threshold settings.

Results Obtained:

Decision Tree classifier:

Accuracy: 0.794889992902768
Precision: 0.6082474226804123
Recall: 0.6327077747989276
F1-score: 0.6202365308804204
ROC-AUC: 0.7429948140403904

Gradient Boost classifier:

Accuracy: 0.7814052519517388
Precision: 0.631578947368421

Recall: 0.41823056300268097
F1-score: 0.5032258064516129
ROC-AUC: 0.6651963625824217

Random forest classifier:

Accuracy: 0.7863733144073811
Precision: 0.625
Recall: 0.48257372654155495
F1-score: 0.5446293494704992
ROC-AUC: 0.6891633111472253

Neural Network:

Accuracy: 0.8019872249822569
Precision: 0.6284153005464481
Recall: 0.6166219839142091
F1-score: 0.6224627875507442
ROC-AUC: 0.7426739263200389

Logistic Regression:

Accuracy: 0.8055358410220014
Precision: 0.6486486486486487
Recall: 0.579088471849866
F1-score: 0.6118980169971672
ROC-AUC: 0.7330770544577515

Comprehensive Evaluation of the Different ML algorithms:

- Logistic Regression has the highest accuracy and precision among all models, indicating its effectiveness in correctly classifying churn and non-churn instances. However, its recall is slightly lower compared to the decision tree and neural network models.
- Decision Tree classifier performs reasonably well with balanced precision and recall, indicating that it effectively identifies both positive and negative instances. Its F1-score is also competitive with other models.
- Gradient Boosting classifier shows a lower recall compared to other models, which suggests it may struggle more in correctly identifying churn instances. However, it has a relatively high precision, indicating fewer false positives.
- Random Forest classifier exhibits similar performance to the decision tree, with slightly lower precision and recall. However, its F1-score and ROC-AUC are still competitive with other models.
- Neural Network demonstrates good overall performance with balanced precision, recall, and F1-score. Its ROC-AUC score is also comparable to other models, indicating good discrimination between positive and negative instances.

Insights gained from the Comprehensive Evaluation:

-Overall, all models show reasonable performance in predicting customer churn. However, logistic regression and decision tree models stand out with their competitive accuracy, precision, and recall.

-Depending on the specific business requirements and trade-offs between precision and recall, different models may be preferred. For example, if minimizing false positives (high precision) is crucial, logistic regression may be preferred, while decision trees may be more suitable if achieving a balance between precision and recall is important.

-Further fine-tuning of hyperparameters and feature engineering may lead to improved performance across all models. Additionally, ensemble techniques like stacking or blending could be explored to combine the strengths of different models and further enhance predictive performance.

- High-performing models like Logistic Regression can effectively identify customers who are likely to churn. By leveraging these predictions, businesses can proactively engage with at-risk customers to prevent churn, such as offering personalized incentives or addressing underlying issues.
- Understanding the strengths and weaknesses of different models helps businesses allocate resources effectively. For instance, Decision Trees excel in identifying positive instances of churn, making them suitable for targeted marketing campaigns or retention efforts.
- By accurately predicting churn, businesses can tailor retention strategies to specific customer segments. For example, customers identified as high-risk by the model could receive special offers or dedicated support to increase loyalty and reduce churn probability.
- Predictive models enable businesses to anticipate customer needs and preferences, leading to enhanced customer experiences. By addressing pain points and delivering personalized solutions, businesses can foster stronger customer relationships and loyalty.
- Churn prediction models help mitigate revenue loss associated with customer attrition. By proactively identifying and retaining customers at risk of churn, businesses can maintain a stable customer base and minimize the negative impact on revenue.

-Effective churn prediction enables businesses to focus on acquiring new customers while retaining existing ones. By leveraging insights from predictive models, businesses can develop strategies to maximize customer lifetime value and drive sustainable growth.

Overall, interpreting model performance and its implications for predicting customer churn empowers businesses to make data-driven decisions, optimize resources, and cultivate long-term relationships with customers. By proactively addressing churn risk, businesses can achieve greater success and competitiveness in the marketplace.

Overall Insights gained from the Project:

1. Logistic Regression emerged as the top-performing algorithm for predicting customer churn, demonstrating high accuracy, precision, and F1-score. Its simplicity and interpretability make it a practical choice for businesses.
2. While Logistic Regression excelled overall, other algorithms such as Decision Trees, Random Forests, and Neural Networks showed competitive performance in specific metrics like recall and ROC-AUC. Each algorithm has its strengths and weaknesses, highlighting the importance of considering trade-offs in model selection.
3. The project provided valuable insights into customer churn prediction, enabling businesses to identify at-risk customers, optimize resource allocation, and enhance retention strategies. By leveraging predictive models, businesses can proactively address churn and foster stronger customer relationships.

Reflection on Effectiveness:

Overall, the implemented algorithms effectively addressed the task of predicting customer churn. The rigorous evaluation of multiple algorithms provided a comprehensive understanding of their performance and implications for businesses. Logistic Regression proved to be the most reliable choice, offering a balance between accuracy and interpretability. However, the diversity of algorithms explored allowed for a nuanced analysis of model trade-offs and considerations.

Suggestions for Future Work:

1. Ensemble Methods: Exploring ensemble methods such as stacking or boosting could potentially improve predictive performance by leveraging the strengths of multiple models.

2. Feature Engineering: Further investigation into feature engineering techniques could enhance model interpretability and predictive power. Feature selection, transformation, and creation may uncover additional insights into customer churn behavior.
3. Hyperparameter Tuning: Fine-tuning model hyperparameters through techniques like grid search or Bayesian optimization could optimize model performance and generalization.
4. Temporal Analysis: Incorporating temporal features and conducting time-series analysis may capture evolving patterns of customer churn over time, leading to more accurate predictions.
5. Customer Segmentation: Segmenting customers based on churn risk profiles and tailoring retention strategies to specific segments could further enhance the effectiveness of churn prediction models.

In summary, while the project achieved its objectives in predicting customer churn and providing actionable insights for businesses, there are opportunities for further refinement and exploration in future work. By continuously iterating and improving model performance, businesses can stay ahead of customer churn challenges and drive sustainable growth.

What works:

-Thorough exploration of the Telco Customer Churn dataset provided valuable insights into customer demographics, service subscriptions, and churn behavior.
-Implementing multiple machine learning algorithms allowed for a comprehensive analysis of their performance in predicting churn.
-Comparative analysis of different algorithms enabled the identification of the most effective approach for churn prediction.
-The project provided insights into factors influencing customer churn and the effectiveness of predictive modeling techniques.

What does not work:

-Some machine learning models may have suffered from overfitting due to the complexity of the dataset or inadequate regularization techniques.
-The dataset may lack certain features that could provide more predictive power for churn prediction, limiting the effectiveness of the models.
-Imbalance in the distribution of churned and non-churned customers may have affected the performance of some models, leading to biased predictions.

Lessons Learned from the Project and How the Project Helped:

-Proper preprocessing techniques, such as handling missing values, encoding categorical variables, and scaling features, are crucial for model performance.
-Not all machine learning algorithms perform equally well for every task. Experimenting with different algorithms and evaluating their performance is essential for choosing the best approach.
-Understanding different evaluation metrics and their implications is crucial for interpreting model performance accurately.
-Sometimes, simpler models like logistic regression may outperform more complex models like neural networks, especially when interpretability is essential.
-The project highlighted the importance of continuous learning and staying updated with the latest developments in machine learning techniques and methodologies.

References:

Here is the Demo link which has an well-rounded view of all the implementations which you can test yourself given you download the csv file associated with the dataset and then do a read.csv on that file:
Dataset Link:
https://www.kaggle.com/datasets/blastchar/telco-customer-churn

Demo Link:
https://colab.research.google.com/drive/1Gydkt6TmZpgmebmoilieYNDGA1vv6ZUn?usp=sharing