Customer Churn Prediction Using Machine Learning

Project members:

Emrullah Ayaz

Selahattin Koray Yıldız

Problem Statement

Customer churn rate is the rate at which a user using an application or an individual who is receiving any service abandons the service or application they are using. This rate is a very important issue for companies. Customer churn, also known as customer attrition, refers to the phenomenon where customers stop doing business with a company. Predicting customer churn is crucial for businesses as it helps them identify at-risk customers and take proactive measures to retain them. Retaining existing customers is often more cost-effective than acquiring new ones, making churn prediction a critical task for improving customer satisfaction and profitability. Machine learning plays a significant role in solving this problem by analyzing historical customer data to identify patterns and predict which customers are likely to churn. By leveraging machine learning algorithms, businesses can develop targeted retention strategies, such as personalized offers or improved customer service, to reduce churn rates. This project aims to build a machine learning model to predict customer churn based on customer behavior, demographics, and transaction history.

Dataset Overview

- **Brief Description of the Dataset:** Telco Customer Churn dataset is a tabular dataset that generally contains detailed information about the customers of a telecommunications company. This dataset provides data for 7043 customers and includes many details such as whether the customers left or not, the reasons for leaving if so, the monthly and total amounts they paid, the locations they live in, and the services they received. With this data, we aimed to use machine learning techniques to identify existing customers at risk of leaving and to develop predictive models to prevent these customers from leaving. This dataset contains 19 categorical, 9 numerical, 4 geographical, and 1 textual feature.
- **Number of Instances (Rows):7043**
- **Number of Features (Columns):33**
- **Types of Features:**

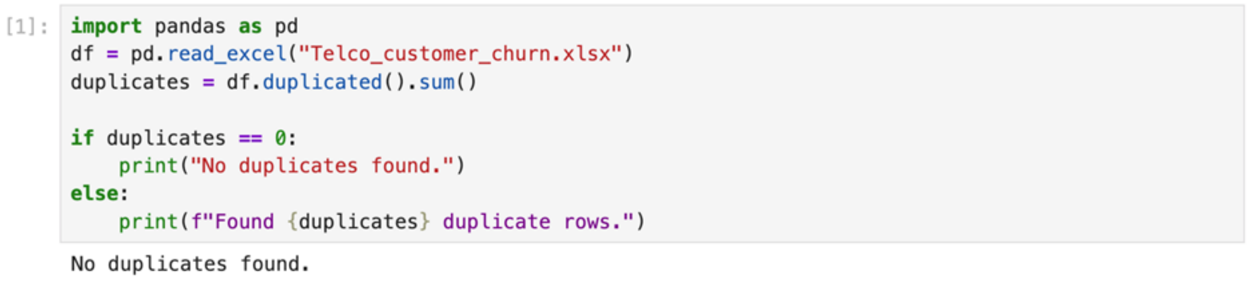
- Categorical: CustomerID, Gender, Senior Citizen, Partner, Dependents, Phone Service, Multiple Lines, Internet Service, Online Security, Online Backup, Device Protection, Tech Support, Streaming TV, Streaming Movies, Contract, Paperless Billing, Payment Method,Churn Label, Churn Reason

- Numerical: Count, Zip Code, Latitude, Longitude, Tenure Months, Monthly Charges, Total Charges, Churn Value, Churn Score, CLTV

- Geographical: Country, State, City, Lat Long
- Textual: Churn Reason

- **Target Variable :**Churn Value (Binary, 1 for churned and 0 for not churned), Churn Label (Categorical, yes/no)

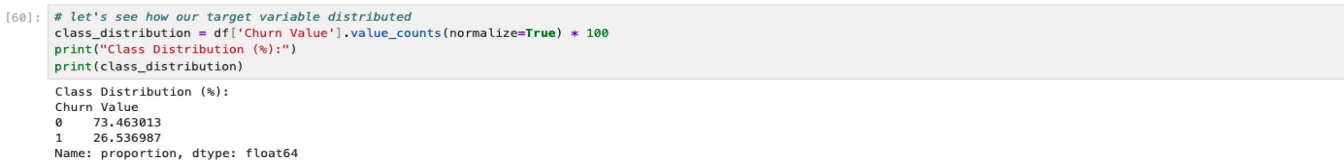
- **Duplicate Entries:** No, there is no duplicate entry in the dataset.



```
[1]: import pandas as pd
     df = pd.read_excel("Telco_customer_churn.xlsx")
     duplicates = df.duplicated().sum()

     if duplicates == 0:
         print("No duplicates found.")
     else:
         print(f"Found {duplicates} duplicate rows.")

     No duplicates found.
```

Data Processing

We chose our test set with a stratified approach, and we can explain the reason for this as follows:



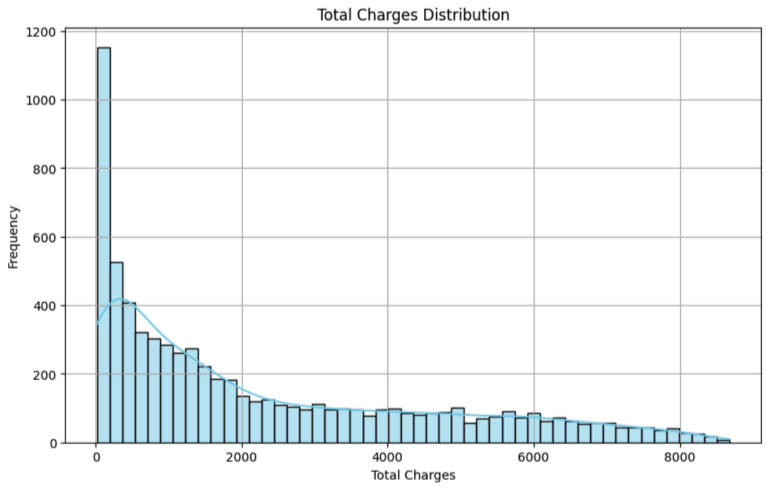
```
[60]: # let's see how our target variable distributed
      class_distribution = df['Churn Value'].value_counts(normalize=True) * 100
      print("Class Distribution (%):")
      print(class_distribution)

      Class Distribution (%):
      Churn Value
      0    73.463013
      1    26.536987
      Name: proportion, dtype: float64
```

As can be seen, in the distribution of our target variable, there is a value of 0 in 73 percent and a value of 1 in 26 percent. If we had chosen our test set randomly, for example, in the k-fold split method, the distribution of 0s and 1s in each fold would not be fair and this would cause the model to be wrong. Therefore, we chose the stratified approach and did the split process, so the number of 0s and 1s in each fold was distributed fairly.

```
[62]: missing_values = df.isnull().sum()
      print("Missing values:")
      print(missing_values[missing_values > 0])
      # most of the customers' churn value is 0, so they don't have churn reason
      # it is sense, because if customer is not leaved, so he/she does not have to have churn reason

      Missing values:
      Total Charges      11
      Churn Reason     5174
      dtype: int64
```

There were missing values in 2 columns. For the Total Charges column, we first looked at the distribution of that column and it was as follows:

```
[63]: plt.figure(figsize=(10, 6))
      sns.histplot(df['Total Charges'], kde=True, bins=50, color='skyblue')
      plt.title('Total Charges Distribution')
      plt.xlabel('Total Charges')
      plt.ylabel('Frequency')
      plt.grid(True)
      plt.show()

      # Skewness value
      print("Skewness:", df['Total Charges'].skew())
```

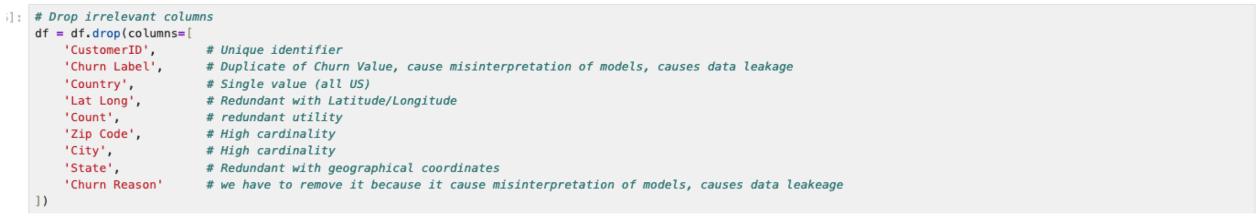


```
Skewness: 0.961642499724251
```

As can be seen, the distribution of the total charges column had excessive skewness. Therefore, as the most accurate approach, we filled in the missing 11 raws with the median value of the total charges column (because it is numerical and skewed).

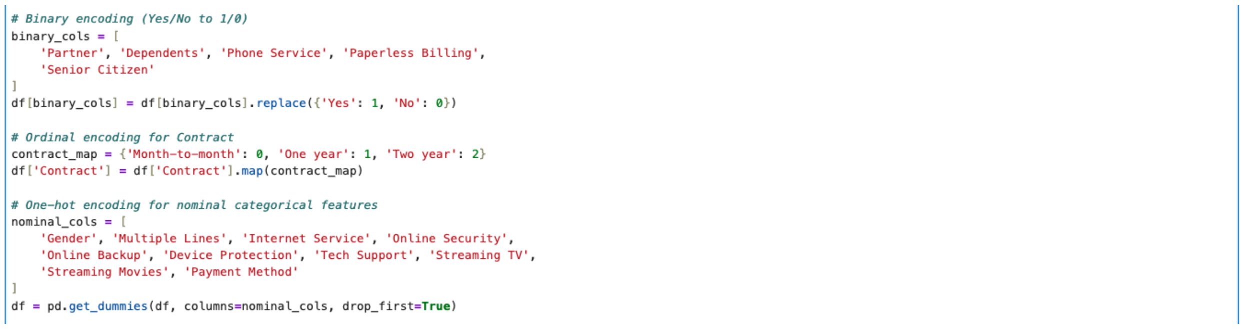
```
[64]: df['Total Charges'] = df['Total Charges'].fillna(df['Total Charges'].median()) # Fill with median (since it’s numerical and skewed)
```

The Churn Reason column was the column where the reasons for customers leaving were written. This column was filled only for customers who left, and it contained missing values for customers who were still continuing. And as you can see, there were approximately 73 percent missing values, because this column was filled only for customers who left, and it was empty for customers who continued. Since this column was text, the missing value values would be the same for 5174 columns, and the values of these 5174 raw in the target variable were also the same. Filling in the missing values in this text column caused data leakage. Because there was a high correlation between these missing values and the target variable. This misled our models and caused an overfitting problem. For these reasons, we dropped the Churn Reason column. In addition to the Churn Reason column, we also dropped columns that were of no use to us or were copies of the target variable. You can see the dropped columns and their reasons in the photo below:



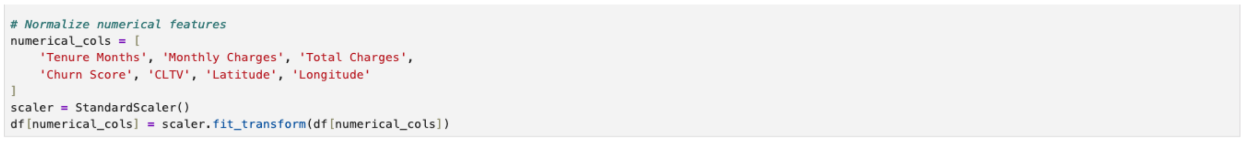
```
# Drop irrelevant columns
df = df.drop(columns=[
    'CustomerID',        # Unique identifier
    'Churn Label',       # Duplicate of Churn Value, cause misinterpretation of models, causes data leakage
    'Country',           # Single value (all US)
    'Lat Long',          # Redundant with Latitude/Longitude
    'Count',             # redundant utility
    'Zip Code',          # High cardinality
    'City',              # High cardinality
    'State',             # Redundant with geographical coordinates
    'Churn Reason'       # we have to remove it because it cause misinterpretation of models, causes data leakeage
])
```

We needed to transform our categorical features. We can see that some of our categorical values are only binary, some are ordinal, and some columns are neither binary nor ordinal, so we transformed them using one hot encoding. You can see the details in the photo below:



```
# Binary encoding (Yes/No to 1/0)
binary_cols = [
    'Partner', 'Dependents', 'Phone Service', 'Paperless Billing',
    'Senior Citizen'
]
df[binary_cols] = df[binary_cols].replace({'Yes': 1, 'No': 0})

# Ordinal encoding for Contract
contract_map = {'Month-to-month': 0, 'One year': 1, 'Two year': 2}
df['Contract'] = df['Contract'].map(contract_map)

# One-hot encoding for nominal categorical features
nominal_cols = [
    'Gender', 'Multiple Lines', 'Internet Service', 'Online Security',
    'Online Backup', 'Device Protection', 'Tech Support', 'Streaming TV',
    'Streaming Movies', 'Payment Method'
]
df = pd.get_dummies(df, columns=nominal_cols, drop_first=True)
```

We have normalized our numerical features with StandardScaler, so our model will work with numerical numbers within a certain range and produce more efficient results. You can examine the normalization code we made for numerical columns in the photo below:

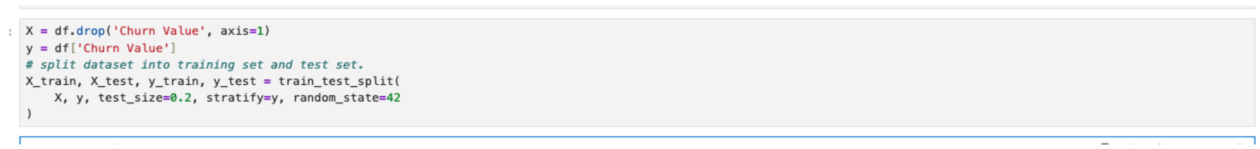


```
# Normalize numerical features
numerical_cols = [
    'Tenure Months', 'Monthly Charges', 'Total Charges',
    'Churn Score', 'CLTV', 'Latitude', 'Longitude'
]
scaler = StandardScaler()
df[numerical_cols] = scaler.fit_transform(df[numerical_cols])
```

Model Selection and Training

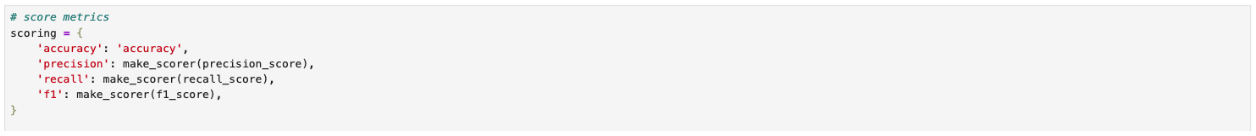
We tested 6 different models here. These models:
Logistic regression
Random Forest
XGBoost
Gradient Boosting
SVM

KNN
we chose the models and trained on these models. While doing this training, we divided the data we had into two. We did this as follows, we separated 80 percent for training and 20 percent for testing. However, while doing all these, we definitely had to preserve the class distribution ratio and we achieved this with stratify=y. You can see how we did this below.



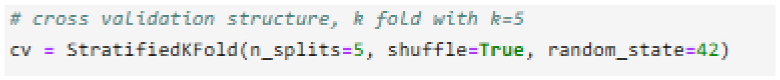
```
X = df.drop('Churn Value', axis=1)
y = df['Churn Value']
# split dataset into training set and test set.
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, stratify=y, random_state=42
)
```

We examined different metrics to evaluate the model:
Accuracy
Precision
Recall
F1 Score
We evaluated the model according to the metrics.



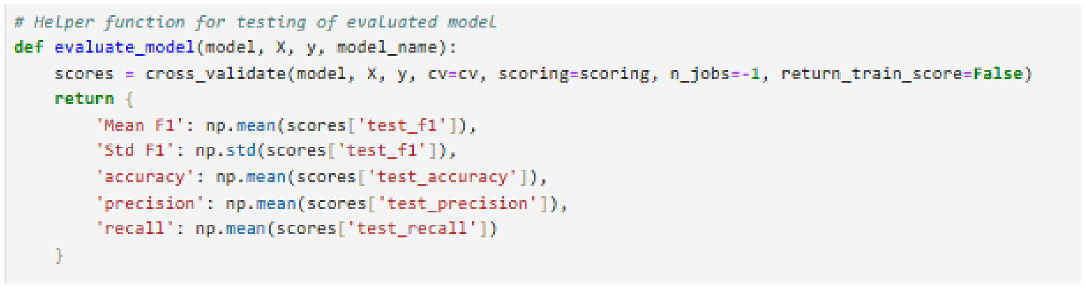
```
# score metrics
scoring = {
    'accuracy': 'accuracy',
    'precision': make_scorer(precision_score),
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),
}
```

Then, using k-fold stratified cross-validation, we divided the data into 5 parts and performed testing/training on each one.



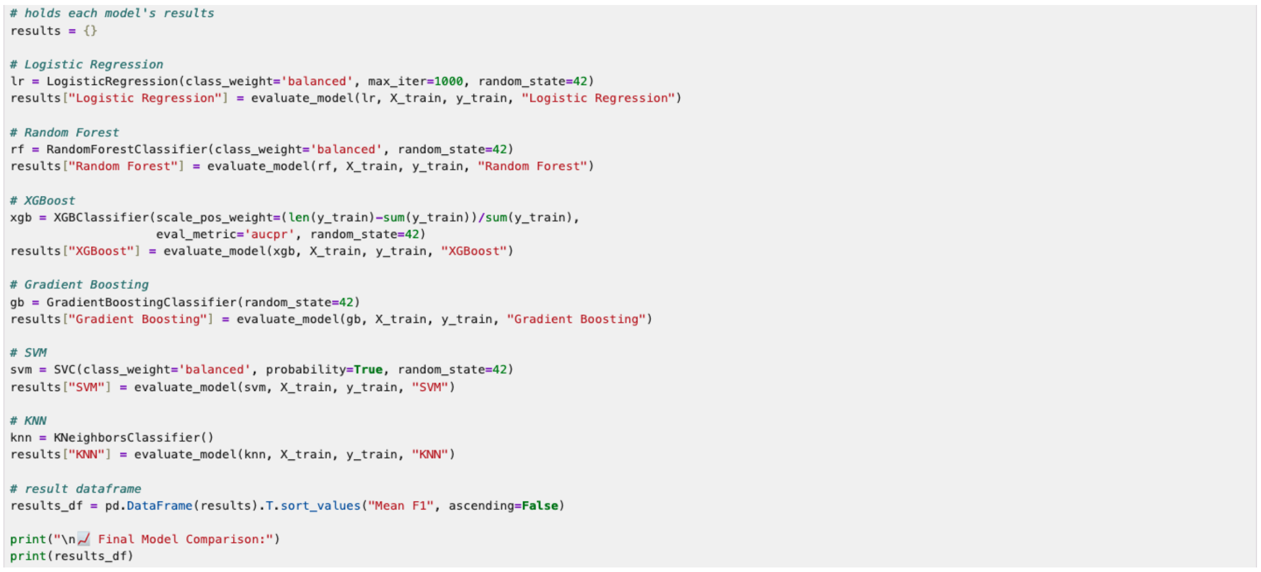
```
# cross validation structure, k fold with k=5
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

After doing all this, we moved on to the model evaluation part.



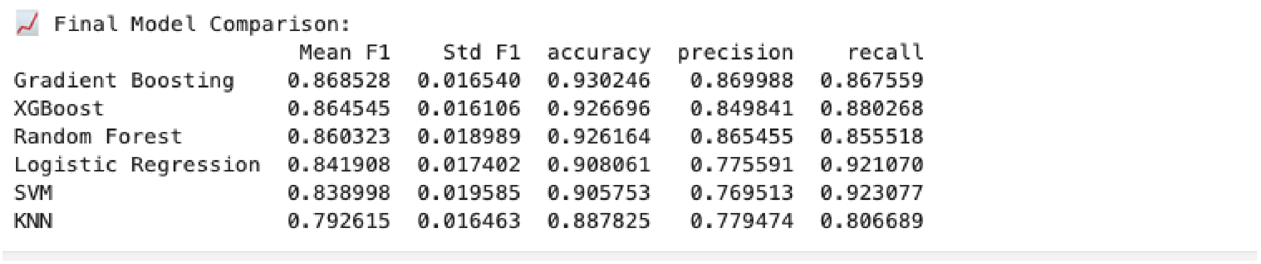
```
# Helper function for testing of evaluated model
def evaluate_model(model, X, y, model_name):
    scores = cross_validate(model, X, y, cv=cv, scoring=scoring, n_jobs=-1, return_train_score=False)
    return {
        'Mean F1': np.mean(scores['test_f1']),
        'Std F1': np.std(scores['test_f1']),
        'accuracy': np.mean(scores['test_accuracy']),
        'precision': np.mean(scores['test_precision']),
        'recall': np.mean(scores['test_recall'])
    }
```

We evaluated the given model with cross_validate. We were able to see the average scores for each metric.



```python
# holds each model's results
results = {}

# Logistic Regression
lr = LogisticRegression(class_weight='balanced', max_iter=1000, random_state=42)
results["Logistic Regression"] = evaluate_model(lr, X_train, y_train, "Logistic Regression")

# Random Forest
rf = RandomForestClassifier(class_weight='balanced', random_state=42)
results["Random Forest"] = evaluate_model(rf, X_train, y_train, "Random Forest")

# XGBoost
xgb = XGBClassifier(scale_pos_weight=(len(y_train)-sum(y_train))/sum(y_train),
                    eval_metric='aucpr', random_state=42)
results["XGBoost"] = evaluate_model(xgb, X_train, y_train, "XGBoost")

# Gradient Boosting
gb = GradientBoostingClassifier(random_state=42)
results["Gradient Boosting"] = evaluate_model(gb, X_train, y_train, "Gradient Boosting")

# SVM
svm = SVC(class_weight='balanced', probability=True, random_state=42)
results["SVM"] = evaluate_model(svm, X_train, y_train, "SVM")

# KNN
knn = KNeighborsClassifier()
results["KNN"] = evaluate_model(knn, X_train, y_train, "KNN")

# result dataframe
results_df = pd.DataFrame(results).T.sort_values("Mean F1", ascending=False)

print("\n📈 Final Model Comparison:")
print(results_df)
```

In the last part of our code, we defined the 6 different models we mentioned at the beginning and performed model training with 5k-fold cross validation with the evaluate_model function. We recorded the average results as (np.mean). Our main goal here is to train all models under the same conditions and make an evaluation accordingly.

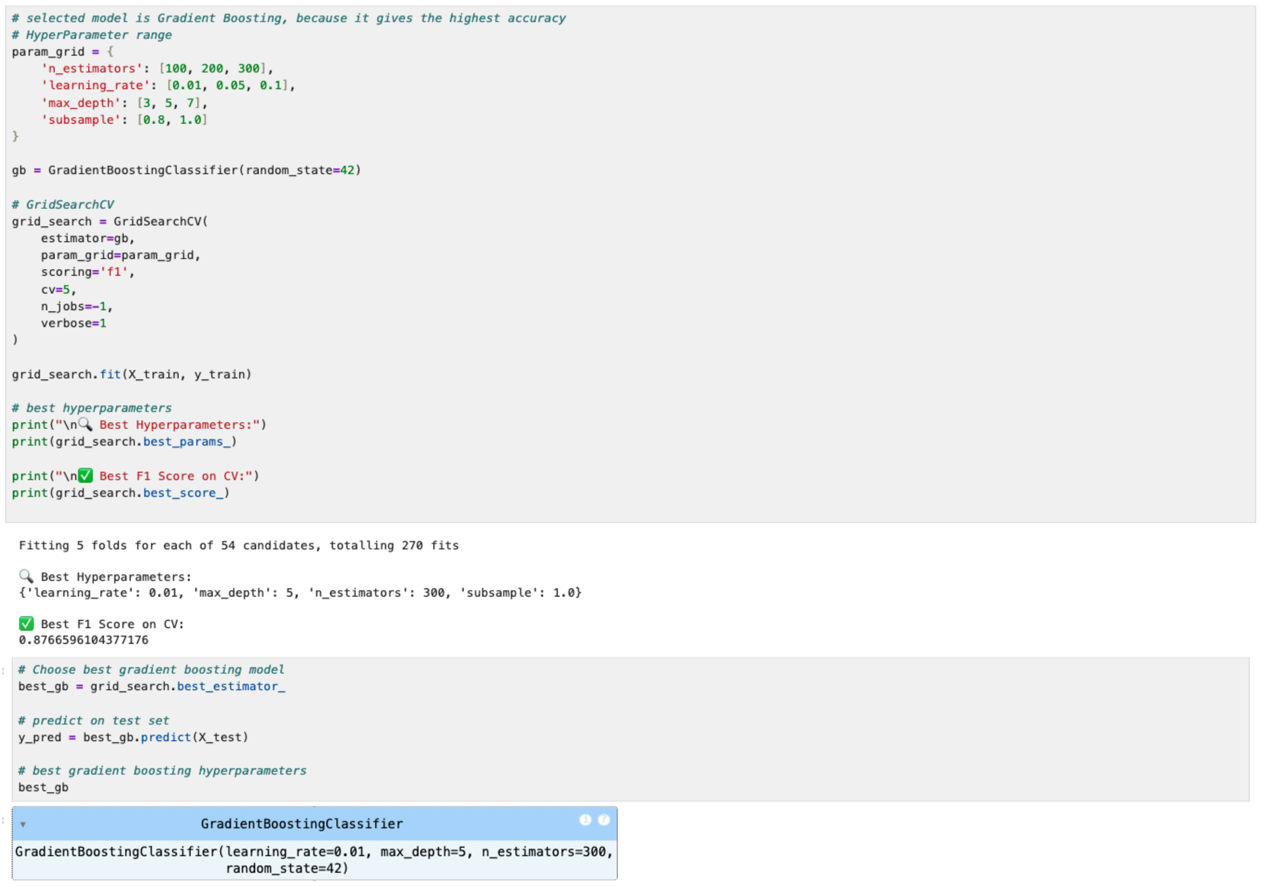


```
📈 Final Model Comparison:
                      Mean F1    Std F1  accuracy  precision    recall
Gradient Boosting    0.868528  0.016540  0.930246   0.869988  0.867559
XGBoost              0.864545  0.016106  0.926696   0.849841  0.880268
Random Forest        0.860323  0.018989  0.926164   0.865455  0.855518
Logistic Regression  0.841908  0.017402  0.908061   0.775591  0.921070
SVM                  0.838998  0.019585  0.905753   0.769513  0.923077
KNN                  0.792615  0.016463  0.887825   0.779474  0.806689
```

In the last stage, we evaluated our trained models according to the metrics we provided. In the photo above, you can examine the mean, std, accuracy, precision and recall values of the models in a data frame. In this way, we can choose which model is more successful. Based on the data here, we determined that our most successful model is Gradient Boosting and decided to fine-tune this model.

Fine-tune Your Model

According to the results we obtained in the previous stage, we selected Gradient Boosting as the best performing model. Now, in order to further increase the success rate of this model, a hyperparameter adjustment was made using GridSearchCV. F1 score was selected as the evaluation metric. This metric establishes a balance between precision and recall. You can see the parameter range we determined for Grid Search below. These parameter combinations were evaluated with 5k fold stratified cross validation. We trained the model for each combination and tested it according to the F1 score.



```python
# selected model is Gradient Boosting, because it gives the highest accuracy
# HyperParameter range
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.01, 0.05, 0.1],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0]
}

gb = GradientBoostingClassifier(random_state=42)

# GridSearchCV
grid_search = GridSearchCV(
    estimator=gb,
    param_grid=param_grid,
    scoring='f1',
    cv=5,
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

# best hyperparameters
print("\n🔍 Best Hyperparameters:")
print(grid_search.best_params_)

print("\n✅ Best F1 Score on CV:")
print(grid_search.best_score_)
```

```
Fitting 5 folds for each of 54 candidates, totalling 270 fits

🔍 Best Hyperparameters:
{'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 300, 'subsample': 1.0}

✅ Best F1 Score on CV:
0.8766596104377176
```

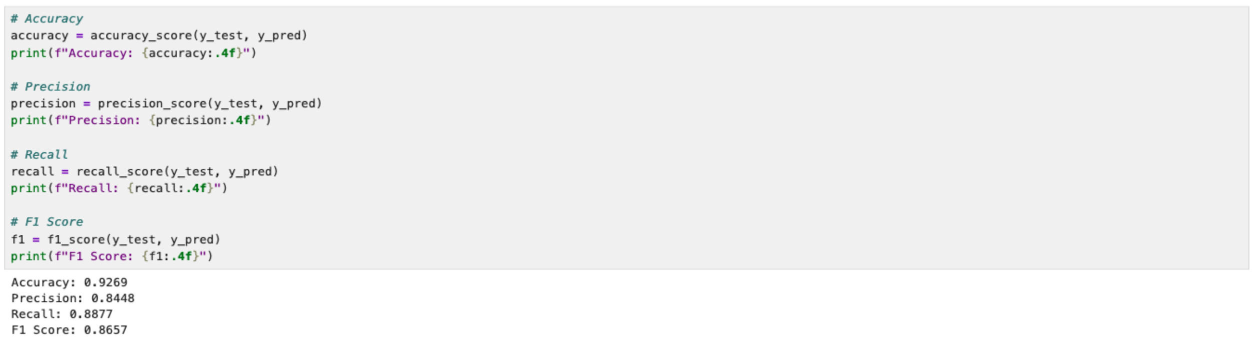
```python
# Choose best gradient boosting model
best_gb = grid_search.best_estimator_

# predict on test set
y_pred = best_gb.predict(X_test)

# best gradient boosting hyperparameters
best_gb
```

```
GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.01, max_depth=5, n_estimators=300,
                           random_state=42)
```

As can be seen here, the learning rate = 0.01, max depth = 5, estimators = 300, subsample = 1.0 was found as a result of the process. These results gave us the highest F1 score. For these reasons, our model was re-trained with these parameters and made ready for the evaluation phase on the test data.

Testing

In this stage, we used the finely tuned Gradient Boosting model from the previous stage to evaluate the performance of the model on the test set. We tested this model on the test data. We used some metrics in this stage. These are: Accuracy, Precision, Recall and F1 Score.



```
# Accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")

# Precision
precision = precision_score(y_test, y_pred)
print(f"Precision: {precision:.4f}")

# Recall
recall = recall_score(y_test, y_pred)
print(f"Recall: {recall:.4f}")

# F1 Score
f1 = f1_score(y_test, y_pred)
print(f"F1 Score: {f1:.4f}")
```

```
Accuracy: 0.9269
Precision: 0.8448
Recall: 0.8877
F1 Score: 0.8657
```

After testing the model with test data, the model shows high success in terms of accuracy according to the data we have obtained.

Summary

- **Discussion about the performance of our model**

After performing k-fold tests on 6 different models, we saw that the Gradient Boosting model gave the best result. After making our choice with this model, we fine-tuned and tested our model. We obtained 92.69 percent accuracy, 84 percent precision, 88 percent recall, and 86 percent F1 score metrics of our model. The 92.69 percent accuracy of our model was an expected result, and we can see that it has a high performance compared to the other 5 models. We were satisfied with this high accuracy, because we did everything by the book in the preprocessing process before training the model and filled in the necessary missing values.

- **Unexpected Results**

At first, there were too many missing values in the Churn Reason column. We had a hard time filling these missing values because these missing values were directly related to the target variable. The reason for this was that the churn value of columns with missing values, that is, without a churn reason, was 0. If the customer had not left, the churn reason was directly null. When we tried to fill the Churn Reasons with a value, all rows had the same value and this was misleading our models, causing data leakage. It was creating an overfitting problem and the accuracy of our models was 100%. This was something we did not expect. Later, we decided to remove the Churn Reason column and the results started to be as we wanted.

- **Ways of improving the performance of model**

1st way - If we had lowered the learning rate while finding the hyperparameters, our model could have had better performance. However, since we did not have the computing power to reduce the learning rate any further, we determined our steps at the lowest possible level.
2nd way - By doing feature engineering, we could have a more performing model by deriving new features from existing features. For this, the most logical feature engineering should be done by consulting domain experts and new features should be derived.

3rd way - Increasing the number of dataset samples can help the model to be trained better and to perform better.

References

Celik, O., & Osmanoglu, U. O. (2019). Comparing techniques used in customer churn analysis. *Journal of Multidisciplinary Developments, 4*(1), 30–38.https://www.researchgate.net/profile/Oezer-Celik-2/publication/337103029_Comparing_to_Techniques_Used_in_Customer_Churn_Analysis/links/5dc52a29a6fdcc2d2ffc1a24/Comparing-to-Techniques-Used-in-Customer-Churn-Analysis.pdf

Elmetwally, T. (n.d.). *Customer Churn Analysis & Plotly Visualization* [Notebook]. Kaggle.https://www.kaggle.com/code/tawfikelmetwally/customer-churn-analysis-plotly-visualization

Kumar, A. S., & Chandrakala, D. (2016). A survey on customer churn prediction using machine learning techniques. *International Journal of Computer Applications, 154*(10), 13–18.https://www.researchgate.net/profile/Saran-A/publication/310757545_A_Survey_on_Customer_Churn_Prediction_using_Machine_Learning_Techniques/links/5bb5fb8a299bf13e605e2ae9/A-Survey-on-Customer-Churn-Prediction-using-Machine-Learning-Techniques.pdf

Manzoor, A., Qureshi, M. A., Kidney, E., & Longo, L. (n.d.). A review on machine learning methods for customer churn prediction and recommendations for business practitioners. *IEEE*.https://ieeexplore.ieee.org/abstract/document/10531735

Vafeiadis, T., Diamantaras, K. I., Sarigiannidis, G., & Chatzisavvas, K. Ch. (2015). A comparison of machine learning techniques for customer churn prediction. *Simulation Modelling Practice and Theory, 55*, 1–9.https://www.sciencedirect.com/science/article/abs/pii/S1569190X15000386

Yean, Z. C. (n.d.). Telco customer churn (IBM) dataset. Kaggle. Retrieved April 4, 2025, from https://www.kaggle.com/datasets/yeanzc/telco-customer-churn-ibm-dataset