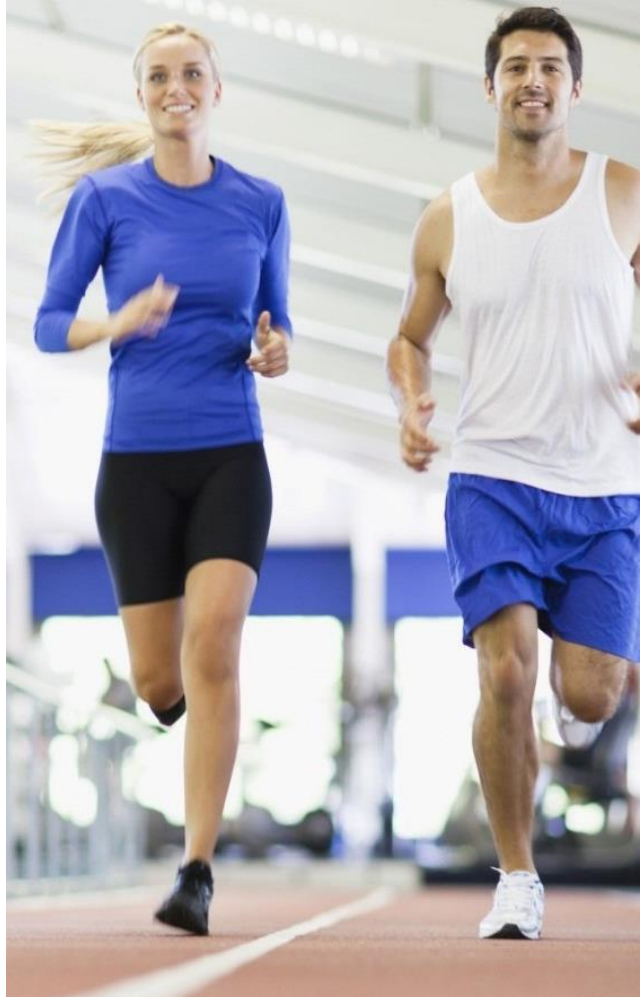




Analysis of Gym Customer Churn

PREDICTIVE MODELING AND INSIGHTS

Introduction

- Overview of the project
- Objective: To predict customer churn using various supervised learning models
- Importance of predicting customer churn for business decision-making

Dataset Description

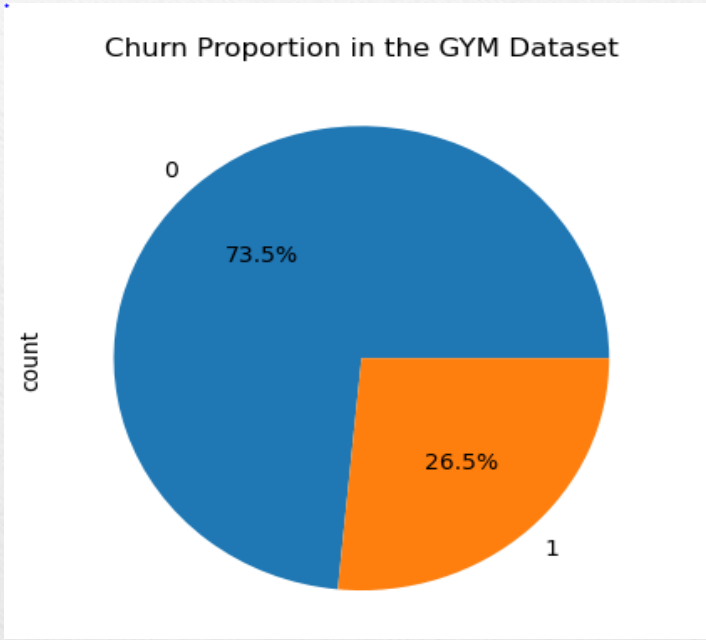
- Source of the dataset: The dataset being used is from Kaggle Dataset link: https://www.kaggle.com/datasets/adrianvinueza/gym-customers-features-and-churn
- Brief description of the dataset: The dataset has 4000 rows, 14 columns
- Target variable: Churn (Yes/No)

GYM Churn Proportion

The GYM customer churn percentage is 26.5%

For every 100 customers around 26 members are leaving GYM.

The above results are based on the provided data set of 4000 records.



Data Exploration

- The dataset has 4000 records 14 columns and clean data.
- There are no null values
- There are no duplicates
- Target value is Churn it is a NUMERICAL value “0” and “1”. Its data type is INT.

```
#To find missing values
data.isnull().sum()
```

```
gender                             0
Near_Location                      0
Partner                            0
Promo_friends                      0
Phone                              0
Contract_period                    0
Group_visits                       0
Age                                0
Avg_additional_charges_total       0
Month_to_end_contract              0
Lifetime                           0
Avg_class_frequency_total          0
Avg_class_frequency_current_month  0
Churn                              0
dtype: int64
```

```
#Identifying duplicate records
print(data.duplicated().sum())
```

```
0
```

Data Cleaning

```
# Handling missing values, encoding categorical variables, scaling numerical variables

# Identify columns with missing values
missing_values = data.isnull().sum()
missing_values.info()
# Impute missing values for numerical columns with median
num_cols = data.select_dtypes(include=['float64', 'int64']).columns
data[num_cols] = data[num_cols].fillna(data[num_cols].median())

# Impute missing values for categorical columns with mode
cat_cols = data.select_dtypes(include=['object']).columns
for col in cat_cols:
    if data[col].isnull().any():
        mode_value = data[col].mode()
        if not mode_value.empty:
            data[col].fillna(mode_value[0], inplace=True)

# Encoding categorical variables using OneHotEncoder
data = pd.get_dummies(data, columns=cat_cols, drop_first=True)
```

```
<class 'pandas.core.series.Series'>
Index: 14 entries, gender to Churn
Series name: None
Non-Null Count  Dtype
--------------  -----
14 non-null     int64
dtypes: int64(1)
```

Data Preprocessing

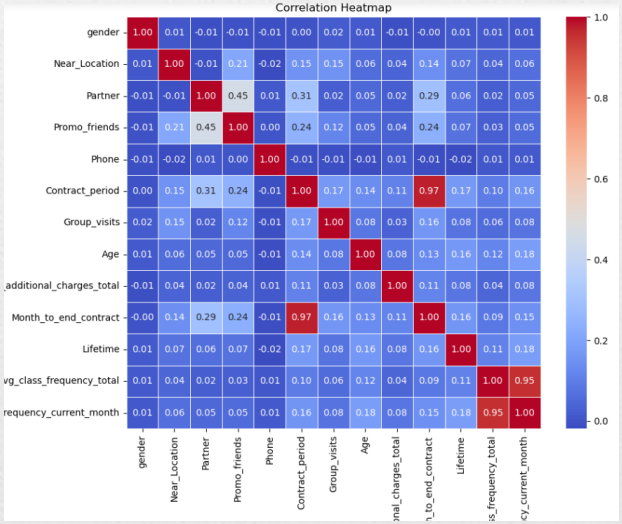
- Steps taken for data cleaning (e.g., handling missing values, encoding categorical variables)
- Feature engineering (e.g., creating new features, scaling)
- Splitting the data into training and test sets

```
#identify datatypes
data.dtypes
```

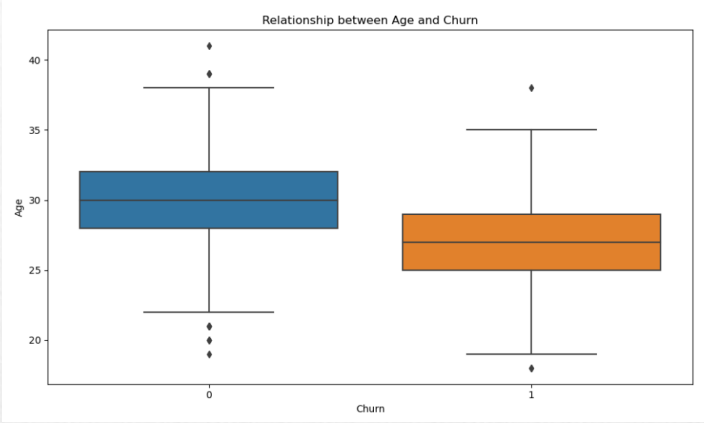
```
]: gender                              int64
   Near_Location                       int64
   Partner                             int64
   Promo_friends                       int64
   Phone                               int64
   Contract_period                     int64
   Group_visits                        int64
   Age                                 int64
   Avg_additional_charges_total      float64
   Month_to_end_contract             float64
   Lifetime                            int64
   Avg_class_frequency_total         float64
   Avg_class_frequency_current_month float64
   Churn                               int64
   dtype: object
```

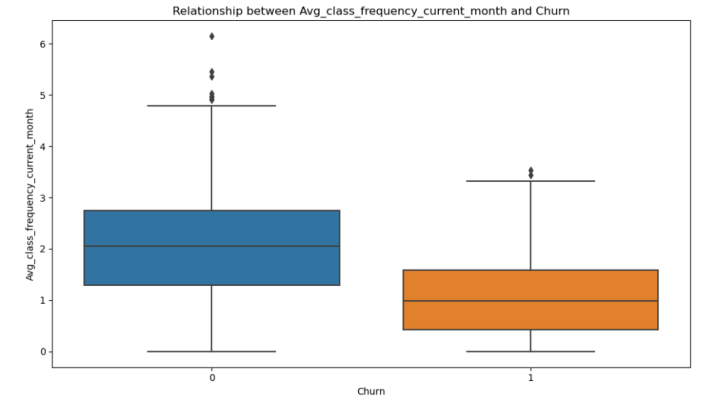
Exploratory Data Analysis

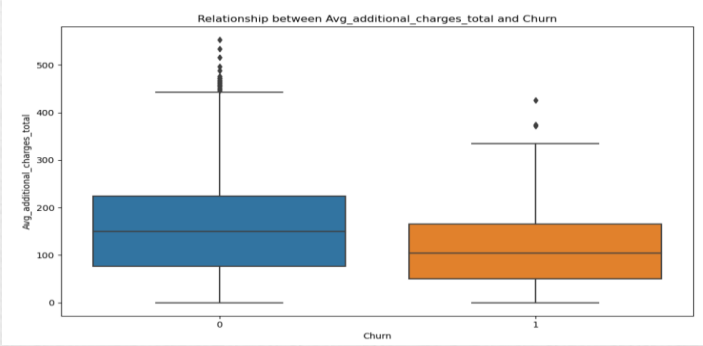
- Key insights from EDA
- Distribution of the target variable
- Important features and their distributions contract period, Avg class frequency, monthly Avg class frequency.
- Correlation between features and the target variable
- Visualizations: Histograms, bar plots, correlation heatmaps

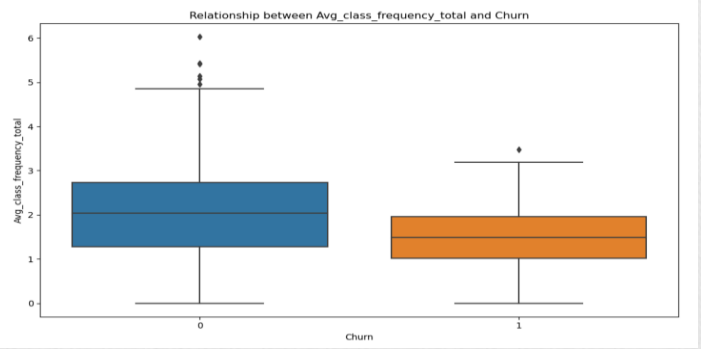


Relationship between target and features









Model Selection

Overview of the models used:

- **K-Nearest Neighbors (KNN)**
- **Logistic Regression**
- **Random Forest**
- **Support Vector Machine (SVM)**
- **XGBoost**

Test-Train

- 80% of data for training
- 20% of data for testing
- StandardScaler used to normalize features
- Applied PCA to the pipeline with 95% variance

```
# Select relevant features (dropping the )
X = data.drop(['Churn'], axis=1)
y = data['Churn']

# Churn is the target variable
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=30)
```

```
#Applying PCA to the scaled features
for name, model in models.items():
    pipe = Pipeline([('scaler', StandardScaler()), ('pca', PCA(n_components=0.95)), (name, model)])
    pipe.fit(X_train_scaled, y_train_encoded)
```

Model Comparison - Classification Report

Model	Accuracy	Precision	Recall	F1 Score
KNN	0.85	0.769953052	0.732142857	0.750572082
Logistic Regression	0.9	0.899521531	0.839285714	0.868360277
Random Forest	0.87	0.89	0.794642857	0.839622642
SVM	0.89	0.905472637	0.8125	0.856470588
XGBoost	0.87	0.944444444	0.834821429	0.886255924

K-Nearest Neighbors (KNN):

- KNN showed significant improvement after hyperparameter tuning, achieving an accuracy of 0.87125.

Logistic Regression:

- Logistic Regression exhibited a higher accuracy of around 90% compared to KNN, with better precision, recall, and F1-score for predicting churn.

Random Forest:

- The Random Forest model achieved an accuracy of around 88%, with precision, recall, and F1-score for predicting churn.

Support Vector Machine (SVM):

- SVM exhibited an accuracy of about 89% with relatively balanced precision and recall for predicting churn.

XGBoost:

- XGBoost, a gradient boosting algorithm, achieved an accuracy of about 87% with similar precision, recall, and F1-score for predicting churn as other models.

Conclusion

- Logistic Regression and SVM seem to be the best performers on this dataset. Given their high accuracy and balanced classification reports, they are good candidates for further tuning and evaluation.
- Random Forest is a solid performer and might benefit from further hyperparameter tuning. KNN showed good improvement after tuning, making it a viable option.
- XGBoost is also a strong candidate, particularly for its ability to handle complex patterns, but it might need more extensive tuning to match the performance of Logistic Regression and SVM.

Q&A