
# Pharos University in Alexandria

# Data Science Methodology (DS302)

## DR. Soha Ahmed

## Eng. Nouran Elslisly

## Project: Customer churn

## Content:



---

- **Team info. :**

| Name | ID |
|---|---|
| Mostafa Mohsen Khalifa | 202100289 |
| Ahmed Mohamed Ghanem | 202100544 |
| Khalid Awad Ghonem | 202101220 |
| Anas Gomaa Bnedary | 202100994 |

## Abstract

This study analyzes an E-Commerce customer dataset to identify key factors influencing customer churn using a comprehensive data science approach. The project involved data exploration, detection, and treatment of missing values and outliers detected using Z-score, followed by predictive modeling. Missing values were addressed using difference technique MICE, Logistic Regression Imputation, and KNN, Three machine learning models, Random Forest, Logistic Regression, and Gradient Boosting, were implemented and compared based on their measure performance to evaluate their effectiveness in predicting churn. Visualizations were created using Power BI to extract actionable business insights.

## Introduction (Idea)

Customer churn - the decision by a customer to stop using a company's products or services, is a significant concern for businesses, particularly in the competitive E-Commerce industry. Accurately predicting churn can help companies retain customers, reduce acquisition costs, and improve service strategies.

This project aims to develop a predictive model using machine learning to classify whether a customer is likely to churn based on behavioral and transactional data. By analyzing the dataset, addressing data quality issues, and applying predictive algorithms, the study supports proactive customer retention initiatives.

## Tools usage (language…)

- **pandas:** It Loads the dataset, handles missing values, merges data, drops columns and manipulating structured data.
- **numpy:** Supports large, multi-dimensional arrays and matrices with mathematical operations, used for statistical calculations and conditional filtering.
- **matplotlib:** Plotting and visualization, for plotting missing values and outlier distributions.
- **sklearn:** Filling in missing values using statistical or ML-based techniques.
  - **KNNImputer:** Imputes missing values using k-nearest neighbors.
  - **SimpleImputer:** Imputes using mean/median.

    Used for KNN and regression imputation.
  - **LinearRegression:** Used to predict missing values using linear regression.
  - **Experimental and IterativeImputer:** Advanced imputation using multiple iterations, handles missing data by modeling each feature as a function of the others.
  - **StandardScaler:** Feature scaling (standardization: mean = 0, std = 1), used before training models to normalize features.
  - **train_test_split:** Splits data into training and testing sets.
  - **RandomForestClassifier:** Ensemble learning model using decision trees, used for classification after preprocessing.
  - **Metrics:** Model evaluation metrics.

- **stats:**
    - Z-Score: Identifies outliers that are a certain number of standard deviations away from the mean.

## Data source (Dataset)

The dataset used in this project is an E-Commerce Customer Behavior Dataset that contains information about 5630 customers and includes 20 features related to customer like tenure, hourSpendOnApp, gender, and churn status, simple image of it.

| CustomerID | Churn | Tenure | PreferredLoginDev | CityTier |
|---|---|---|---|---|
| 50001 | 1 | 4 | Mobile Phone | 3 |
| 50002 | 1 | | Mobile Phone | 1 |
| 50003 | 1 | | Mobile Phone | 1 |
| 50004 | 1 | 0 | Mobile Phone | 3 |
| 50005 | 1 | 0 | Mobile Phone | 1 |
| 50006 | 1 | 0 | Computer | 1 |
| 50007 | 1 | | Mobile Phone | 3 |
| 50008 | 1 | | Mobile Phone | 1 |
| 50009 | 1 | 13 | Mobile Phone | 3 |

Link dataset: https://www.kaggle.com/datasets/ankitverma2010/ecommerce-customer-churn-analysis-and-prediction

## Code

Now we will explain stage of the code with It's explanation,

- **Importing Required Libraries:**

  First we need to import libraries we will use and will help us in the project implementations, we was explain and describe them in part of Tools usage.

- **Preprocessing:**

  Now first compute the missing value in the data to solve it with difference strategies:

```
[14] # Basic exploration
     print("Dataset Shape:", df.shape)
     print("\nMissing Values Summary:")
     print(df.isnull().sum())
     print("\nData Types:")
     print(df.dtypes)
```

  Load the dataset and then we need to explore it:
  **Shape:** this to see the shape of dataset -> (5630,20).
  Then in next line we will count the missing value for each column using **isnull().sum()** to explore missing value.
  **Dtypes**: Shows data types.

```
Dataset Shape: (5630, 20)

Missing Values Summary:
CustomerID                    0
Churn                         0
Tenure                      264
PreferredLoginDevice          0
CityTier                      0
WarehouseToHome             251
```

```
# Calculate the percentage of missing values in each column
missing_percent = df.isnull().mean() * 100
missing_cols = missing_percent[missing_percent > 0].index.tolist()
print("Columns with missing values:", missing_cols)
```

Now analysis missing value:

Calculates the percentage of missing values in each column.

Lists columns that contain any missing values  when it bigger than 0.

Now this is column with missing value:

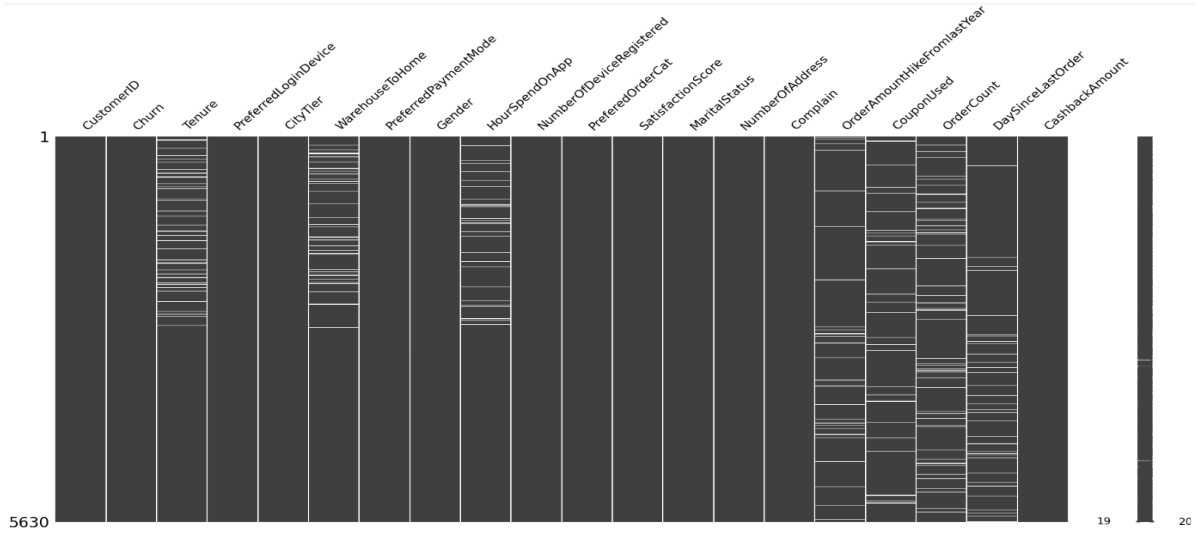
```
Columns with missing values: ['Tenure', 'WarehouseToHome', 'HourSpendOnApp', 'OrderAmountHikeFromlastYear', 'CouponUsed', 'OrderCount'
```

Now we will start to plot and solve missing value,

```
[5]  # Function to visualize missing data patterns
     def visualize_missing_data(df, method_name):
         print(f"\n Missing Data Visualization - {method_name}")
         plt.figure(figsize=(10, 6))
         msno.matrix(df)
         plt.title(f"Missing Data Matrix - {method_name}")
         plt.tight_layout()
         plt.show()

     # Visualize missing data patterns
     visualize_missing_data(df, "Raw Data")
```

Plot the missing value dataset using **matrix plot**:



```python
# Function to analyze missingness patterns in relation to other variables
def analyze_missingness_patterns(df, target_col='Churn'):
    for col in df.columns:
        if df[col].isnull().sum() > 0:
            print(f"\n Missing Analysis for: {col}")
            df['Missing_' + col] = df[col].isnull().astype(int)

            if target_col in df.columns:
                print(f"\n Comparing missing values in '{col}' vs mean '{target_col}':")
                print(df.groupby('Missing_' + col)[target_col].mean())

            # Compare with other numerical columns
            for num_col in df.select_dtypes(include='number').columns:
                if num_col != target_col and num_col != 'Missing_' + col:
                    mean_values = df.groupby('Missing_' + col)[num_col].mean()
                    print(f" {num_col} by missing status in {col}:\n{mean_values}")

            # Remove temporary columns
            df.drop(columns=['Missing_' + col], inplace=True)

# Apply the function
analyze_missingness_patterns(df, target_col='Churn')
```

This function is used to analyze the pattern of missing values by checking how they relate to the target variable (Churn) and other numeric features.
It helps determine whether the messiness is random or related to other values (which is important when choosing an imputation method), to identify MAR or MCAR or MNAR.

**EX:**

**1. Tenure**

- **Churn Rate:** Higher with missing data (30.7% vs. 16.1%).
- **Behavior of Users with Missing Data:**
  - Spend less time on the app.
  - Use fewer coupons.
  - Fewer orders and lower overall engagement.
- **Insight:** Missing values in Tenure may indicate inactive or new users. The missingness is **not random** (not MCAR) and is linked to customer behavior. Removing these rows may introduce bias.
- **Recommendation:**
  - Use imputation models like Regression, MICE, or KNN imputation.

We will start to solve the missing value using knn,

```
# 1. KNN Imputation
knn_numeric = df_knn.select_dtypes(include='number')
imputer_knn = KNNImputer(n_neighbors=5)
df_knn[knn_numeric.columns] = imputer_knn.fit_transform(knn_numeric)
```

Selects numeric columns.
Uses K-Nearest Neighbors (KNN) to fill in missing values based on the 5 nearest records.
Imputes values in df_knn.
KNN is good for local approximation, especially if missing data is not random.

```
# ✅ f. MICE (Iterative Imputer)
mice_numeric = df_mice.select_dtypes(include='number')
imputer_mice = IterativeImputer(random_state=0)
df_mice[mice_numeric.columns] = imputer_mice.fit_transform(mice_numeric)
```

The same in knn brut use different strategy, Iteratively models each variable with missing values using other variables, more robust and accurate for complex datasets.

```
# 2. Regression Imputation (manual implementation for each column with missing values)
for col in missing_cols:
    features = df_regression.drop(columns=[col]).select_dtypes(include='number').columns.tolist()

    # Separate rows where the value is not NaN
    df_complete = df_regression[df_regression[col].notna()]
    df_missing = df_regression[df_regression[col].isna()]

    if not df_missing.empty:
        # Temporarily impute other features with their means
        imputer_temp = SimpleImputer(strategy='mean')
        X_train = imputer_temp.fit_transform(df_complete[features])
        y_train = df_complete[col]
        X_missing = imputer_temp.transform(df_missing[features])

        # Train the model
        model = LinearRegression()
        model.fit(X_train, y_train)

        # Predict and impute
        predicted = model.predict(X_missing)
        df_regression.loc[df_regression[col].isna(), col] = predicted
```

This for loop performs missing value imputation using regression, where missing values in a column are predicted using a regression model trained on other numerical features.
Before training the regression model, it **fills missing values** in the other features using **mean imputation** (to avoid errors in model training).

```
# Display samples after imputation
print("KNN Imputation Sample:\n", df_knn.head())
print("\n Regression Imputation Sample:\n", df_regression.head())
print("\n MICE Imputation Sample:\n", df_mice.head())
```

Now print data after fill it, this is simple data:

```
KNN Imputation Sample:
    CustomerID  Churn  Tenure PreferredLoginDevice
0      50001.0    1.0     4.0         Mobile Phone
1      50002.0    1.0     2.6                Phone
2      50003.0    1.0     2.6                Phone
3      50004.0    1.0     0.0                Phone
4      50005.0    1.0     0.0                Phone
```

Now we will compute the correlation to select best feature and outliers to remove it from data

```
# Helper function: Get continuous numerical columns only
def get_continuous_numerical(df):
    num_cols = df.select_dtypes(include=np.number).columns
    continuous_cols = [col for col in num_cols if df[col].nunique() > 10]
    return continuous_cols
```

This function to extract column with continues value for compute the outliers.

```python
# Calculate Z-scores to detect outliers
# Detect outliers
def detect_outliers(df_input, method_name):
    cont_cols = get_continuous_numerical(df_input)

    # Z-score method
    z_scores = np.abs((df_input[cont_cols] - df_input[cont_cols].mean()) / df_input[cont_cols].std())
    outliers_z = (z_scores > 3).sum(axis=0)
    print(f"\n🔎 Outliers detected using Z-score for {method_name}:\n", outliers_z)
    return outliers_z
```

This function detects outliers in the continuous numerical columns of a given DataFrame using Z-score method.

**z_scores:** Calculates Z-scores for each value in the continuous columns.

Z-score > 3 is considered an **outlier**.

It counts how many outliers are found per column and prints the result.

```python
# Plot outliers using Z-Score
def plot_outliers_z(df_input, method_name):
    cont_cols = get_continuous_numerical(df_input)
    z_scores = np.abs((df_input[cont_cols] - df_input[cont_cols].mean()) / df_input[cont_cols].std())

    plt.figure(figsize=(12, 6))
    sns.boxplot(data=z_scores)
    plt.xticks(rotation=45)
    plt.title(f'Outlier Detection using Z-Score - {method_name}')
    plt.tight_layout()
    plt.show()
```

Start to plot for z-score.

Now we need to remove the outliers,

```python
# Calculate Z-scores to detect outliers
# Handle outliers
def handle_outliers(df_input, method_name):
    df = df_input.copy()

    # Keep 'Churn' to avoid changing it
    churn_column = df['Churn']
    df = df.drop('Churn', axis=1)

    # Numerical columns only
    numeric_cols = df.select_dtypes(include=np.number).columns

    # Remove outliers using Z-score
    z_scores = np.abs((df[numeric_cols] - df[numeric_cols].mean()) / df[numeric_cols].std())
    df = df[(z_scores <= 3).all(axis=1)]  # Remove rows containing outliers

    # Add 'Churn' column back after removing outliers
    df['Churn'] = churn_column[df.index]

    print(f" Handled outliers for {method_name}: Removed {len(df_input) - len(df)} rows ")
    return df
```

This function removes outliers from the numerical columns of a dataset using the Z-score method, and it does not apply normalization.
It also ensures the Churn target column is preserved during this process.
Now we will compute the correlation as we said to select best feature,

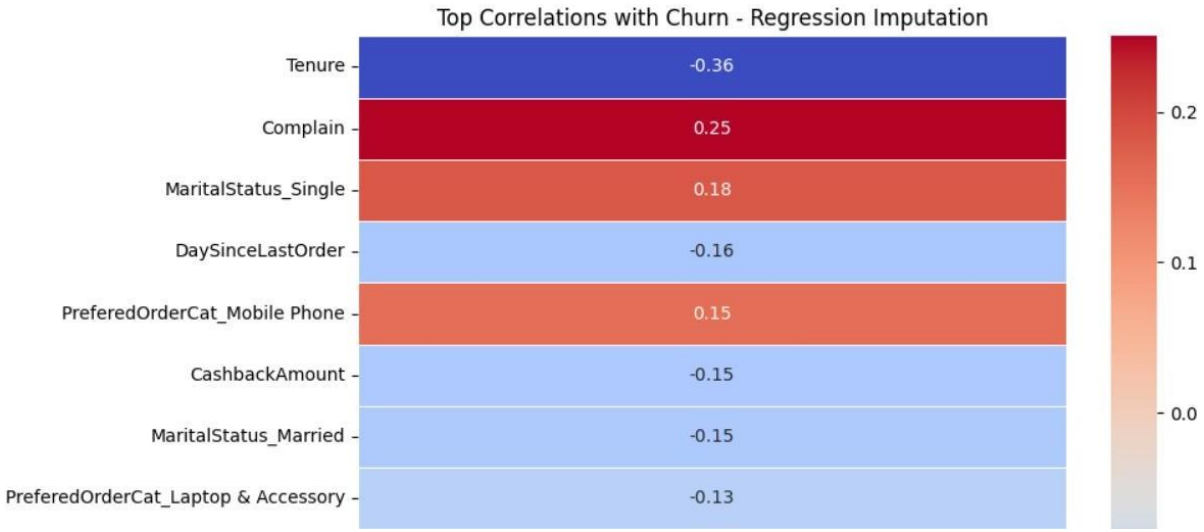
```python
# Calculate correlation with Churn
def correlation_analysis(df_input, method_name):
    df_corr = df_input.copy()
    df_corr = pd.get_dummies(df_corr, drop_first=True)
    corr_matrix = df_corr.corr()
    churn_corr = corr_matrix['Churn'].sort_values(ascending=False)
    print(f"\n🔍 Correlation with Churn - {method_name}:")
    print(churn_corr[1:6])  # Top 5
    return churn_corr
```

To calculate and display the correlation between all features and the target column Churn, and print the top 5 most positively correlated features.

```python
# Plot correlation with Churn
def plot_churn_correlations(df_input, method_name):
    df_corr = df_input.copy()
    df_corr = pd.get_dummies(df_corr, drop_first=True)
    corr_matrix = df_corr.corr()
   #use comment
    churn_corr = corr_matrix['Churn'].drop('Churn').sort_values(key=lambda x: abs(x), ascending=False)
    top_features = churn_corr[:15]  # Top 15 features

    plt.figure(figsize=(10, 8))
    sns.heatmap(top_features.to_frame(), annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, cbar=True)
    plt.title(f'Top Correlations with Churn - {method_name}')
    plt.tight_layout()
    plt.show()
```

Then plot the correlation:



Now select best columns depend on correlation values:

```
# Define the most important features based on correlation analysis
top_features = [
    'Tenure',
    'Complain',
    'DaySinceLastOrder',
    'PreferedOrderCat',
    'CashbackAmount',
    'MaritalStatus',
    'PreferredPaymentMode',
    'PreferredLoginDevice',
    'NumberOfDeviceRegistered',
    'SatisfactionScore',
]

# Select top features
def select_top_features(df_input):
    return df_input[top_features + ['Churn']]

# Apply feature selection
df_regression_selected = select_top_features(df_regression_cleaned)
df_mice_selected = select_top_features(df_mice_cleaned)
df_knn_selected = select_top_features(df_knn_cleaned)
```

We see there is 10 column is good so we will apply the models on it we was explained them.

## ➢ Train model

This part is for training models and choose the best one, to train a model we need to extract and separate target column from other columns.

```
# Prepare data for modeling
def prepare_data(df_input):
    df_input = pd.get_dummies(df_input, drop_first=True)
    X = df_input.drop('Churn', axis=1)
    y = df_input['Churn']
    return X, y
```

This function will return the column target and save it in y variable while the x variable keep all columns except target (Churn).

```
# Define models
models = {
    "Logistic Regression": LogisticRegression(max_iter=1000),
    "Random Forest": RandomForestClassifier(),
    "Gradient Boosting": GradientBoostingClassifier()
}
```

Now define the models, we will use 3 model and compare between them, LR, RF, GB.

```
# Apply evaluation to each dataset
results_regression = run_all_evaluations(df_regression_selected, "Regression")
results_mice = run_all_evaluations(df_mice_selected, "MICE")
results_knn = run_all_evaluations(df_knn_selected, "KNN")
```

In this code we need to train each model in all 3 strategies for missing value that was explained, these information required for **run_all_evaluations** we will explained it later.

```python
# Run evaluations for all datasets
def run_all_evaluations(df, label):
    print(f"\n=== Evaluating models with {label} imputation ===")
    X, y = prepare_data(df)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    return evaluate_models(X_train, X_test, y_train, y_test, models)
```

Split the data to train and test with 80% train and 20% for testing, then send the information to **evaluate_models** this function will explained later.

```python
# Comprehensive evaluation function
def evaluate_models(X_train, X_test, y_train, y_test, model_dict):
    results = {}
    for model_name, model in model_dict.items():
        print(f"Training {model_name}...")
        model.fit(X_train, y_train)
        y_pred = model.predict(X_test)

        results[model_name] = {
            "Accuracy": accuracy_score(y_test, y_pred),
            "Precision": precision_score(y_test, y_pred),
            "Recall": recall_score(y_test, y_pred),
            "F1 Score": f1_score(y_test, y_pred)
        }

        print(f"{model_name} metrics:")
        for metric, value in results[model_name].items():
            print(f"  - {metric}: {value:.4f}")
    return results
```

Here will start to train each model with splitting data that explained in previous function and extract the prediction and result for all models.

```python
# Combine results into a structured DataFrame
combined_results = []
for model_name in models.keys():
    combined_results.append({
        "Model": model_name,
        "Imputation": "Regression",
        **results_regression[model_name]
    })
    combined_results.append({
        "Model": model_name,
        "Imputation": "MICE",
        **results_mice[model_name]
    })
    combined_results.append({
        "Model": model_name,
        "Imputation": "KNN",
        **results_knn[model_name]
    })

df_results = pd.DataFrame(combined_results)
```

Here combine all result from all model in difference strategies apply in data and covert it to data frame.

```python
# Save results to CSV
df_results.to_csv("model_metrics_results.csv", index=False)
```

Then solve the data in csv file.

```python
# Create a comprehensive visualization of model performance
metrics = ["Accuracy", "Precision", "Recall", "F1 Score"]

for metric in metrics:
    plt.figure(figsize=(14, 8))

    # Reshape data for grouped bar chart
    pivot_data = df_results.pivot(index='Model', columns='Imputation', values=metric)

    # Plot grouped bar chart
    ax = pivot_data.plot(kind='bar', figsize=(14, 8))

    plt.title(f'Model {metric} by Imputation Method', fontsize=16)
    plt.ylabel(metric, fontsize=14)
    plt.xlabel('Model', fontsize=14)
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.legend(title='Imputation Method')

    # Add value labels on bars
    for container in ax.containers:
        ax.bar_label(container, fmt='%.3f', fontsize=10)

    plt.tight_layout()
    plt.show()
```

We will plot the result performance of each model for compare.

## ➢ Model evaluation

To evaluate the performance of each model we use metrics that have Accuracy, Precision, Recall (Sensitivity) and F1 Score.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

**Accuracy:** The ratio of correctly predicted samples to total samples.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

**Precision:** Out of all the positive predictions, how many were actually positive, high precision means the model made very few false positives.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

**Recall (Sensitivity):** Out of all actual positives, how many did the model correctly identify, High recall means the model is good at catching all true cases..

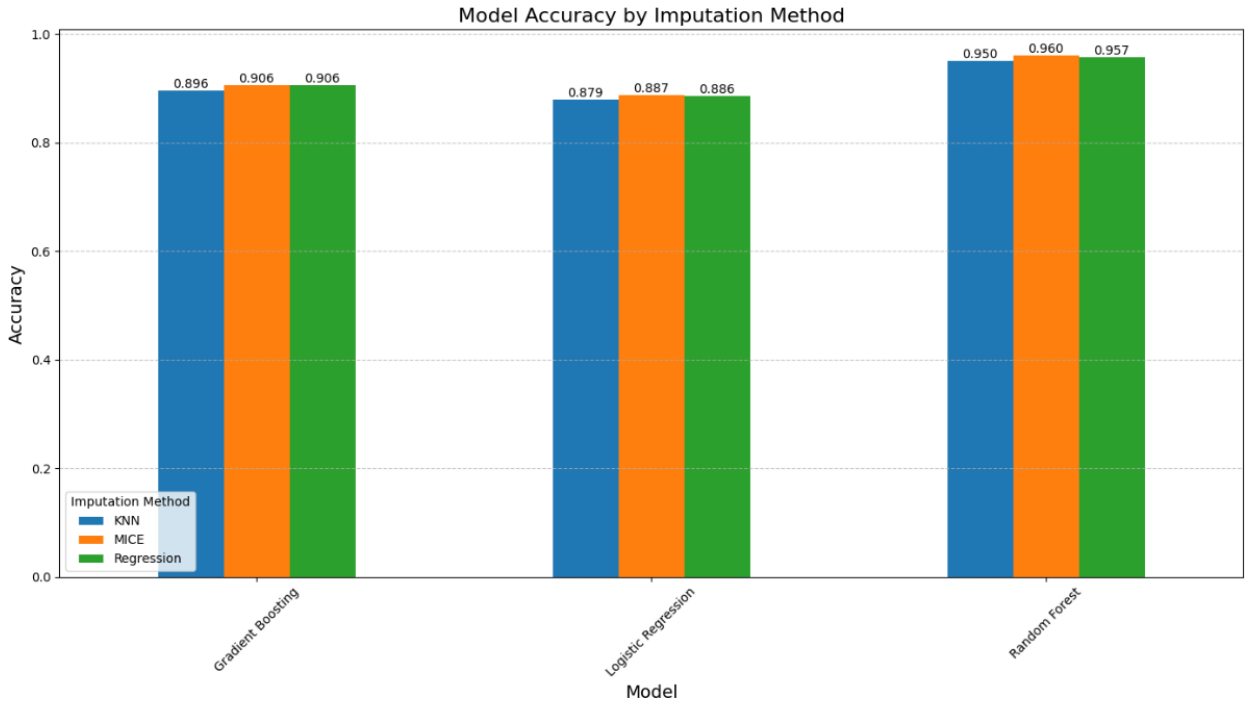
$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

**F1 Score:** The harmonic mean of Precision and Recall, It punishes extreme values more, a model with high precision and very low recall will still get a low F1 score.

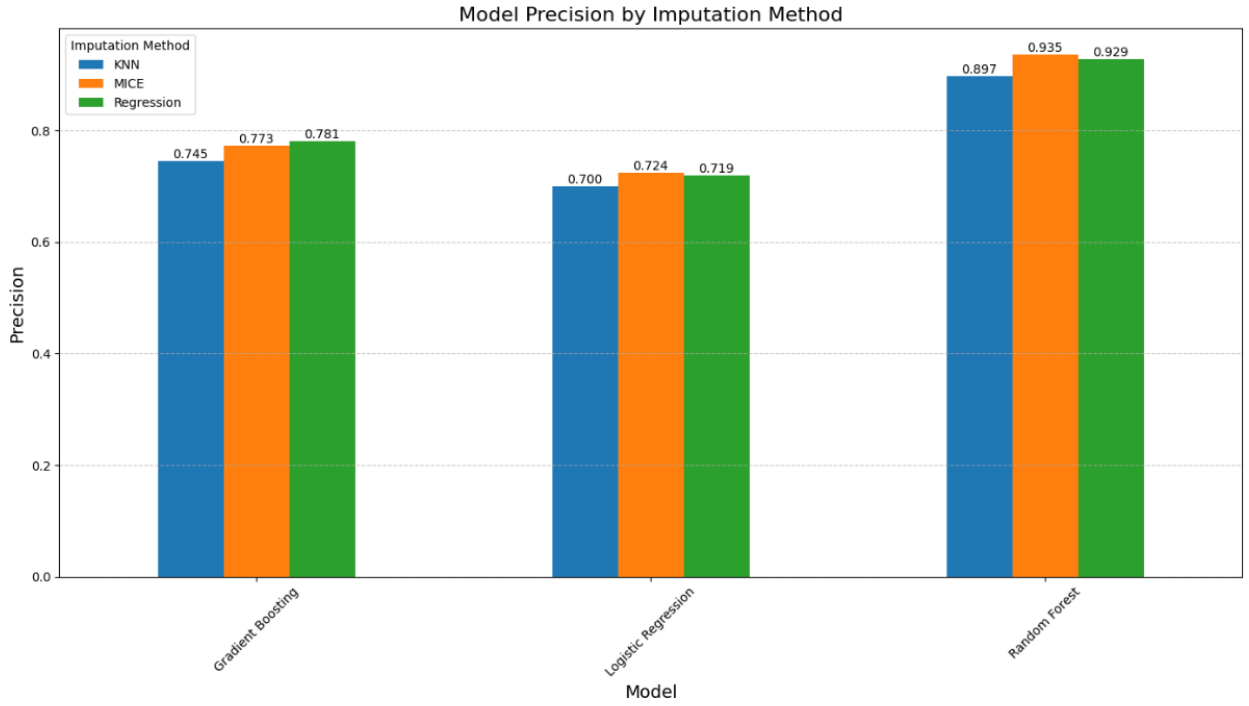
## ➢ Output Result

### Model performance:

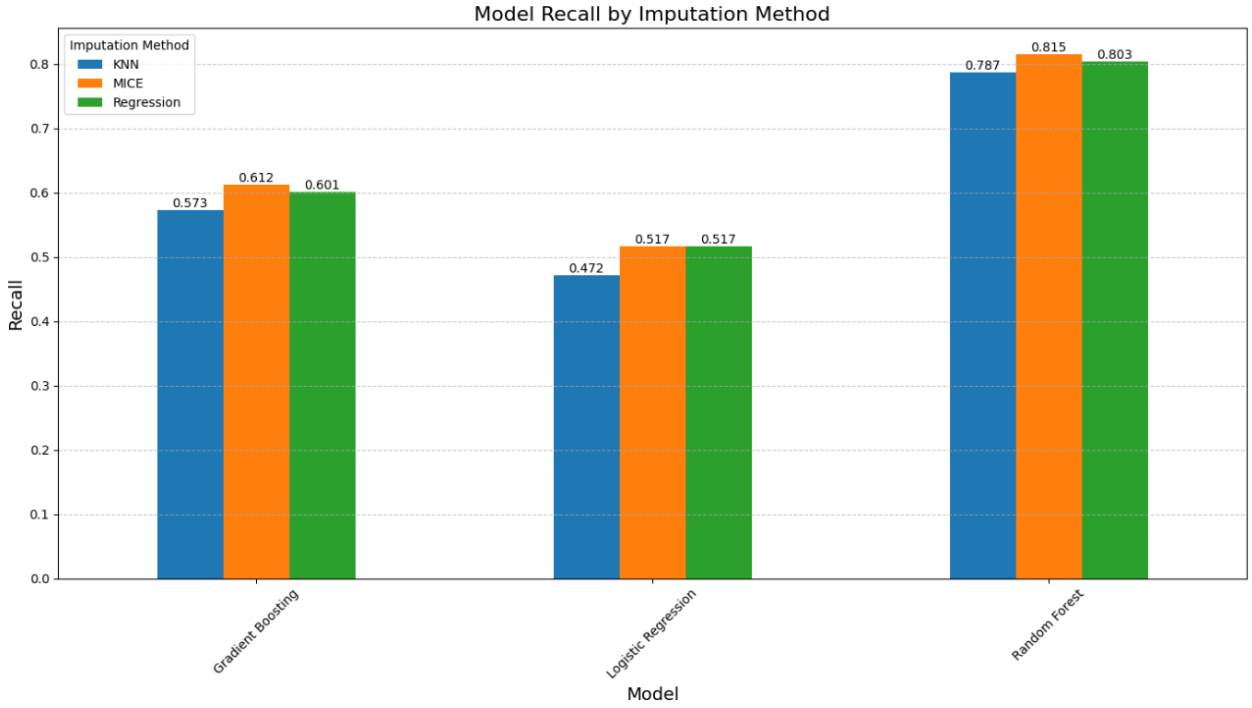
- Accuracy:



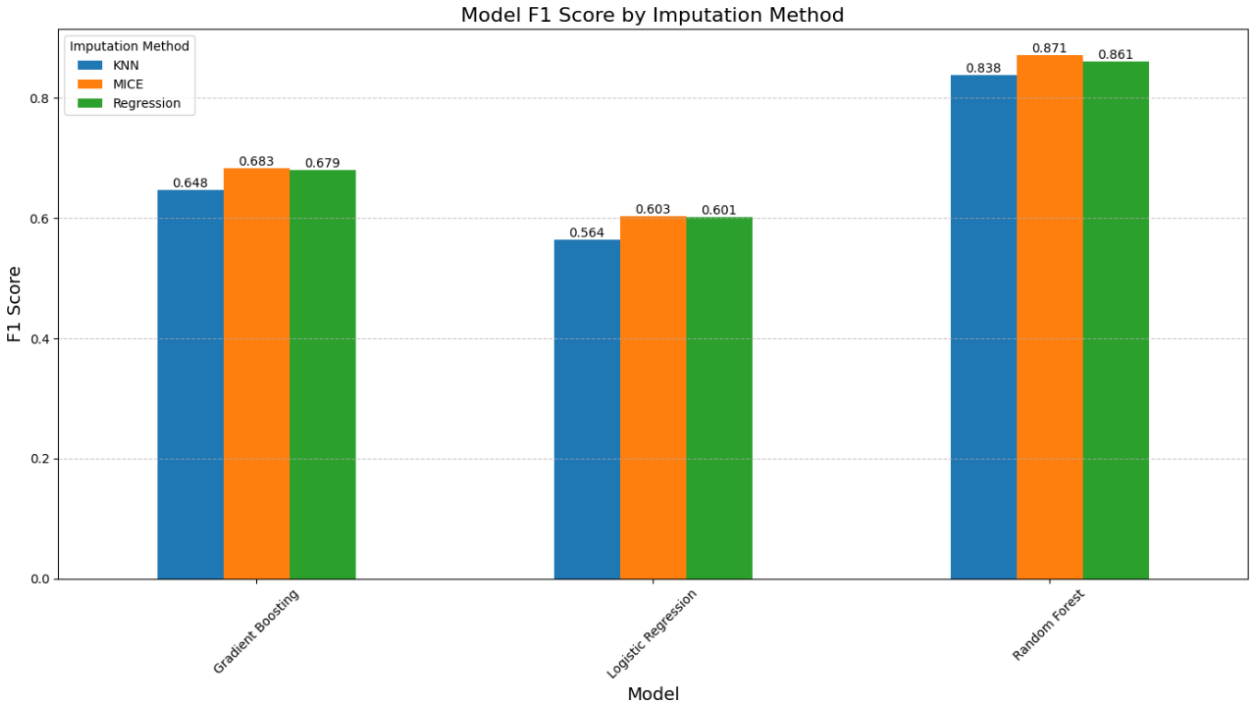

- Precision:



- Recall:




- F1 Score:



**Statistical models performance:**

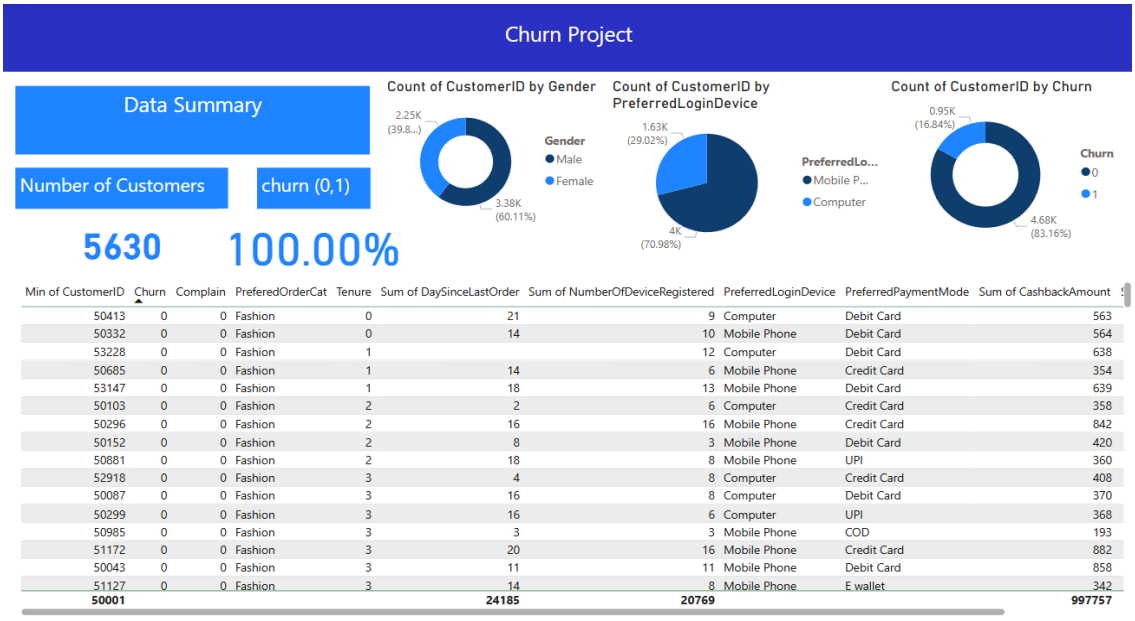
=== Model Performance Comparison ===

| | Model | Imputation | Accuracy | Precision | Recall | F1 Score |
|---|---|---|---|---|---|---|
| 0 | Logistic Regression | Regression | 0.885981 | 0.718750 | 0.516854 | 0.601307 |
| 1 | Logistic Regression | MICE | 0.886916 | 0.724409 | 0.516854 | 0.603279 |
| 2 | Logistic Regression | KNN | 0.878505 | 0.700000 | 0.471910 | 0.563758 |
| 3 | Random Forest | Regression | 0.954206 | 0.916129 | 0.797753 | 0.852853 |
| 4 | Random Forest | MICE | 0.951402 | 0.920000 | 0.775281 | 0.841463 |
| 5 | Random Forest | KNN | 0.956075 | 0.922581 | 0.803371 | 0.858859 |
| 6 | Gradient Boosting | Regression | 0.904673 | 0.775362 | 0.601124 | 0.677215 |
| 7 | Gradient Boosting | MICE | 0.905607 | 0.773050 | 0.612360 | 0.683386 |
| 8 | Gradient Boosting | KNN | 0.896262 | 0.744526 | 0.573034 | 0.647619 |

**Note:** depend on performance the best model was Random Forest.

## Power Pi visualization:

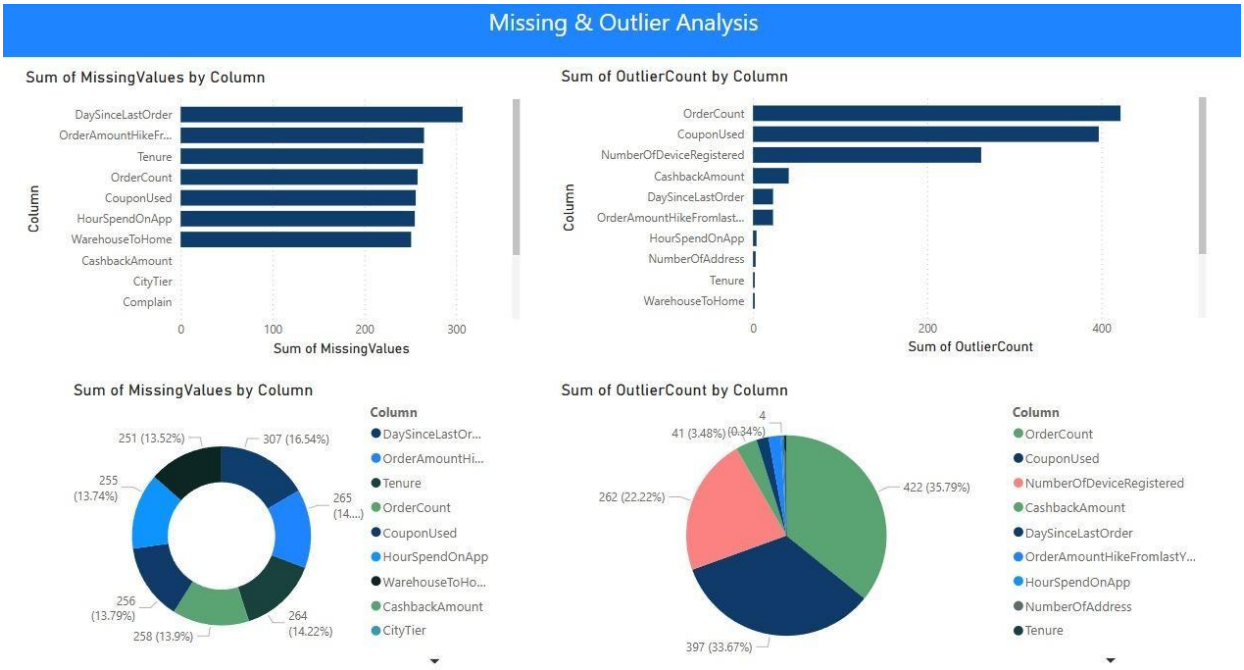
- Data summary before preprocessing:




| Min of CustomerID | Churn | Complain | PreferedOrderCat | Tenure | Sum of DaySinceLastOrder | Sum of NumberOfDeviceRegistered | PreferredLoginDevice | PreferredPaymentMode | Sum of CashbackAmount |
|---|---|---|---|---|---|---|---|---|---|
| 50413 | 0 | 0 | Fashion | 0 | 21 | 9 | Computer | Debit Card | 563 |
| 50332 | 0 | 0 | Fashion | 0 | 14 | 10 | Mobile Phone | Debit Card | 564 |
| 53228 | 0 | 0 | Fashion | 1 | | 12 | Computer | Debit Card | 638 |
| 50685 | 0 | 0 | Fashion | 1 | 14 | 6 | Mobile Phone | Credit Card | 354 |
| 53147 | 0 | 0 | Fashion | 1 | 18 | 13 | Mobile Phone | Debit Card | 639 |
| 50103 | 0 | 0 | Fashion | 2 | 2 | 6 | Computer | Credit Card | 358 |
| 50296 | 0 | 0 | Fashion | 2 | 16 | 16 | Mobile Phone | Credit Card | 842 |
| 50152 | 0 | 0 | Fashion | 2 | 8 | 3 | Mobile Phone | Debit Card | 420 |
| 50881 | 0 | 0 | Fashion | 2 | 18 | 8 | Mobile Phone | UPI | 360 |
| 52918 | 0 | 0 | Fashion | 3 | 4 | 8 | Computer | Credit Card | 408 |
| 50087 | 0 | 0 | Fashion | 3 | 16 | 8 | Computer | Debit Card | 370 |
| 50299 | 0 | 0 | Fashion | 3 | 16 | 6 | Computer | UPI | 368 |
| 50985 | 0 | 0 | Fashion | 3 | 3 | 3 | Mobile Phone | COD | 193 |
| 51172 | 0 | 0 | Fashion | 3 | 20 | 16 | Mobile Phone | Credit Card | 882 |
| 50043 | 0 | 0 | Fashion | 3 | 11 | 11 | Mobile Phone | Debit Card | 858 |
| 51127 | 0 | 0 | Fashion | 3 | 14 | 8 | Mobile Phone | E wallet | 342 |
| **50001** | | | | | **24185** | **20769** | | | **997757** |

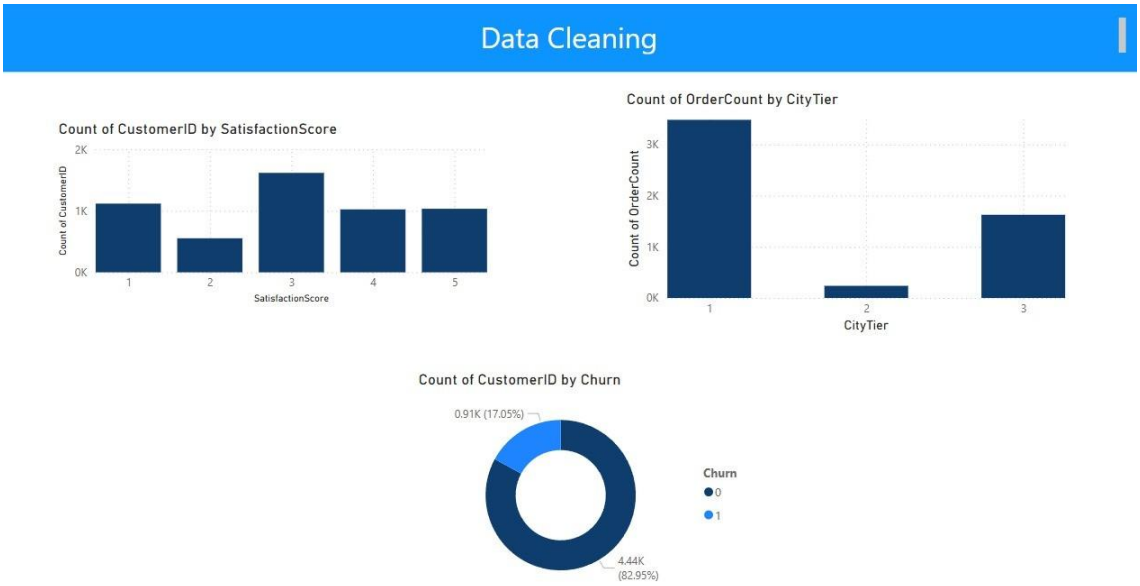

This visualization shows the original state of the dataset including missing values and possible inconsistencies.
It's useful for understanding the raw data quality before cleaning.

- Missing value detected and outliers:



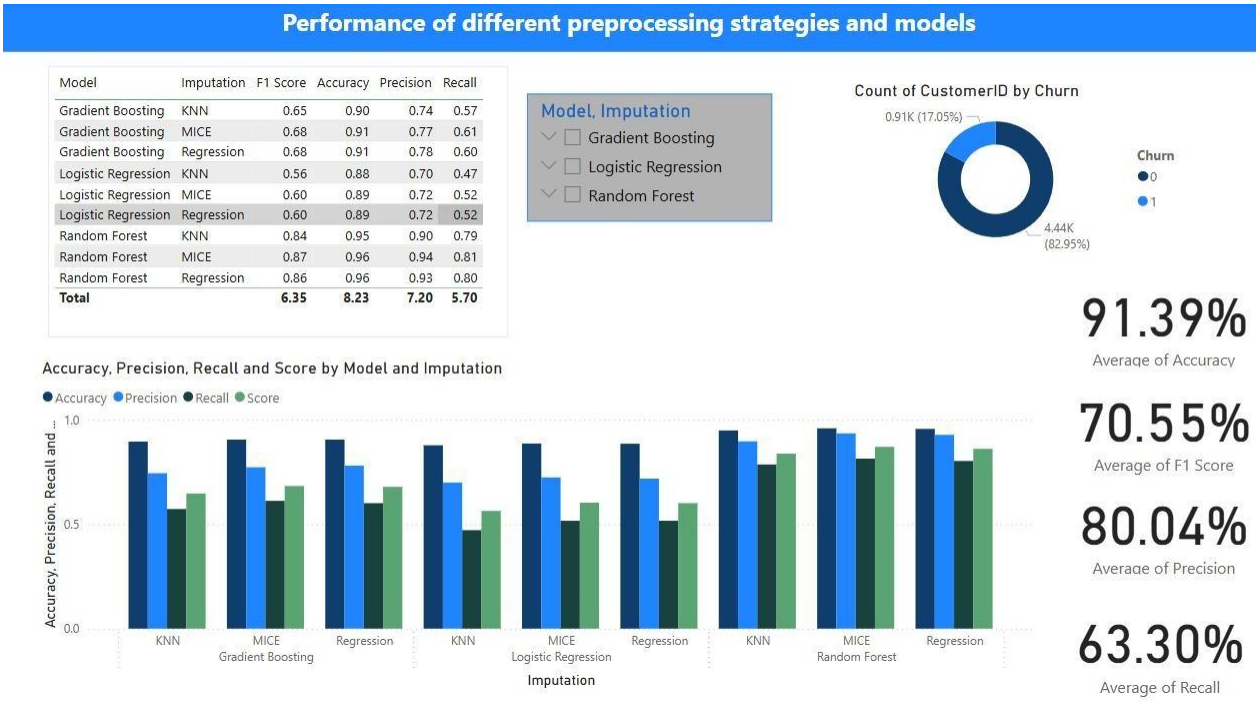

These visuals identify columns with missing data and highlight numeric features with potential outliers using Z-score method.

- Data summary after preprocessing:



This view confirms that the data has been cleaned, missing values handled, outliers removed, and dataset balanced or transformed.

- Models performance and comparison:




| Model | Imputation | F1 Score | Accuracy | Precision | Recall |
|---|---|---|---|---|---|
| Gradient Boosting | KNN | 0.65 | 0.90 | 0.74 | 0.57 |
| Gradient Boosting | MICE | 0.68 | 0.91 | 0.77 | 0.61 |
| Gradient Boosting | Regression | 0.68 | 0.91 | 0.78 | 0.60 |
| Logistic Regression | KNN | 0.56 | 0.88 | 0.70 | 0.47 |
| Logistic Regression | MICE | 0.60 | 0.89 | 0.72 | 0.52 |
| Logistic Regression | Regression | 0.60 | 0.89 | 0.72 | 0.52 |
| Random Forest | KNN | 0.84 | 0.95 | 0.90 | 0.79 |
| Random Forest | MICE | 0.87 | 0.96 | 0.94 | 0.81 |
| Random Forest | Regression | 0.86 | 0.96 | 0.93 | 0.80 |
| **Total** | | **6.35** | **8.23** | **7.20** | **5.70** |

This dashboard compares model performance (Accuracy, Precision, Recall, F1) across Random Forest, Logistic Regression, and Gradient Boosting.

- ➢ Random Forest outperforms the other models in most metrics.
- ➢ Visualization helps stakeholders easily pick the best model.

## ➢ Conclusion:

This project successfully applied a data science workflow to predict customer churn in the E-Commerce domain. Beginning with thorough data preprocessing, including handling missing values using KNN, regression, and MICE imputation methods, and detecting outliers with Z-score, we ensured a clean and reliable dataset for modeling.

Feature selection based on correlation analysis improved model efficiency and interpretability. Three machine learning models, Logistic Regression, Random Forest, and Gradient Boosting, were trained and evaluated. Among them, Random Forest consistently demonstrated the best performance across accuracy, precision, recall, and F1 score metrics, making it the most suitable model for churn prediction in this scenario.

Power BI visualizations were instrumental in summarizing the data journey, from raw input through preprocessing to model comparison, providing clear, actionable insights for business stakeholders. These visual tools support data-driven decisions to proactively manage and reduce customer churn.

---

***The end…Thanks***

## References:

### Data set:

https://www.kaggle.com/datasets/ankitverma2010/ecommerce-customer-churn-analysis-and-prediction