Install Required Packages ¶

In [1]:

```
!pip install xgboost
!pip install catboost
!pip install imblearn
!pip install -U imbalanced-learn
!pip install imblearn==0.0
!pip install -U scikit-learn
!pip install shap
```

```
Requirement already satisfied: pytz>=2017.3 in c:\users\gowthami nagappan
\anaconda\lib\site-packages (from pandas->shap) (2021.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\gowthami
nagappan\anaconda\lib\site-packages (from pandas->shap) (2.8.2)
Requirement already satisfied: joblib>=1.1.1 in c:\users\gowthami nagappan
\anaconda\lib\site-packages (from scikit-learn->shap) (1.2.0)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\gowthami n
agappan\anaconda\lib\site-packages (from scikit-learn->shap) (2.2.0)
Requirement already satisfied: six>=1.5 in c:\users\gowthami nagappan\anac
onda\lib\site-packages (from python-dateutil>=2.7.3->pandas->shap) (1.16.
0)
```

DATA PRE-PROCESSING

Import Packages and CSV

In [2]:

```
#import required packages
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings("ignore")
pd.pandas.set_option("display.max_columns", None)
```

In [3]:

```
# Create Dataframe
dataset = pd.read_excel(r'Telco_customer_churn.xlsx', engine='openpyxl')
#Print shape of dataset
print(dataset.shape)
```

```
(7043, 33)
```

In [4]:

```
# First 5 rows of the dataset
dataset.head()
```

Out[4]:

	CustomerID	Count	Country	State	City	Zip Code	Lat Long	Latitude	Longitud
0	3668-QPYBK	1	United States	California	Los Angeles	90003	33.964131, -118.272783	33.964131	-118.27278
1	9237-HQITU	1	United States	California	Los Angeles	90005	34.059281, -118.30742	34.059281	-118.30742
2	9305-CDSKC	1	United States	California	Los Angeles	90006	34.048013, -118.293953	34.048013	-118.29395
3	7892-POOKP	1	United States	California	Los Angeles	90010	34.062125, -118.315709	34.062125	-118.31570
4	0280-XJGEX	1	United States	California	Los Angeles	90015	34.039224, -118.266293	34.039224	-118.26629

In [5]:

```
dataset["Churn Value"].value_counts()
```

Out[5]:

```
0    5174
1    1869
Name: Churn Value, dtype: int64
```

DATA CLEANING

As per EDA we will change the total charges to numeric

In [6]:

```
dataset['Total Charges'] = pd.to_numeric(dataset['Total Charges'], errors='coerce')
```

Check Null Values

In [7]:

```
##these are the features with nan value
features_with_nan=[column for column in dataset.columns if dataset[column].isnull().sum()
for feature in features_with_nan:
    print(feature,np.round(dataset[feature].isnull().mean()*100,5), '% in the dataset')
```

```
Total Charges 0.15618 % in the dataset
Churn Reason 73.46301 % in the dataset
```

Missing Values in a DataFrame is called "NaN"

Why might a value be missing?

When considering the potential impact of the missing data on the Dataset, it is essential to consider the underlying reasons for why the data are missing Missing data are of 3 types of categories:

Here it is MCAR type missing values in dataset which are Missing completely at Random.

When we make this assumption, we are assuming this missing data is fully unrelated to the other information in the data looks like churn reasons were not given by the customers

Other Data Cleaning steps

Handling Duplicates

In [8]:

```
# Checking for duplicates
dataset.duplicated().sum()
```

Out[8]:

```
0
```

Remove ID from the dataset as it cannot used in Model Training

As a Report form the EDA we can drop some features

In [9]:

```
dataset.drop(['CustomerID','Zip Code','Count', 'Country' , 'State' , 'Lat Long','Latitude
```

FEATURE ENGINEERING

Type of Features

Numeric Features

In [10]:

```
Telco_numeric_features = [colu for colu in dataset.columns if dataset[colu].dtype != 'O']
print('Num of Numerical Features in the Dataset :', len(Telco_numeric_features))
Telco_numeric_features
```

```
Num of Numerical Features in the Dataset : 5
```

Out[10]:

```
['Tenure Months',
 'Monthly Charges',
 'Total Charges',
 'Churn Value',
 'Churn Score']
```

Categorical Features

In [11]:

```
categorical_feature = [column for column in dataset.columns if dataset[column].dtype == '
print('Num of Categorical Features :', len(categorical_feature))
categorical_feature
```

```
Num of Categorical Features : 15
```

Out[11]:

```
['City',
 'Senior Citizen',
 'Partner',
 'Dependents',
 'Multiple Lines',
 'Internet Service',
 'Online Security',
 'Online Backup',
 'Device Protection',
 'Tech Support',
 'Streaming TV',
 'Streaming Movies',
 'Contract',
 'Paperless Billing',
 'Payment Method']
```

Discrete Features

In [12]:

```
discrete_features=[column for column in Telco_numeric_features if (len(dataset[column].ur
print('Num of Discrete Features :',len(discrete_features))
discrete_features
```

```
Num of Discrete Features : 1
```

Out[12]:

```
['Churn Value']
```

Continuous features

In [13]:

```
continuous_features=[column for column in Telco_numeric_features if len(dataset[column].u
print('Num of Continuous Features :',len(continuous_features))
continuous_features
```

```
Num of Continuous Features : 4
```

Out[13]:

```
['Tenure Months', 'Monthly Charges', 'Total Charges', 'Churn Score']
```

Multicollinearity Check

Multicollinearity can be a concern in logistic regression as it can lead to instability in the estimation of the model parameters and affect the interpretation of the coefficients.

Multicollinearity arises if there is high correlation between 2 (or) more independent variables in logistic regression model. This can lead to difficulties in interpreting the effect of individua variables on the outcome variables because their effects may become indistinguishable. Additionally, it can lead to inflated standard errors and unstable estimates of the model coefficients.

There are several methods for detecting multicollinearity including the Variance Inflation Factor.

In [14]:

```
from statsmodels.stats.outliers_influence import variance_inflation_factor
def vif_calculations(selected_features, dataset):

    X = dataset[selected_features]

    X['intercept'] = 1

    # Generate dataframe to keep VIF values
    vifvalues = pd.DataFrame()
    vifvalues["Variable"] = X.columns
    vifvalues["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
    vifvalues = vifvalues[vifvalues['Variable']!='intercept']
    return vifvalues
```

In [15]:

```
# Create copy of dataframe to check variance inflation factor
dfcopy = dataset.copy()
for i in continuous_features:
    dfcopy[i].fillna(dfcopy[i].median(), inplace=True)
```

In [16]:

```
vif_calculations(continuous_features, dfcopy)
```

Out[16]:

	Variable	VIF
0	Tenure Months	5.777478
1	Monthly Charges	3.308527
2	Total Charges	9.472717
3	Churn Score	1.106730

The VIF (variance inflation factor) establishes how strongly the independent variables are correlated. By regressing a variable against every other variable, it is predicted.

A commonly used rule of is that a VIF value greater than 5 or 10 means a problematic degree of multicollinearity.

This Dataset has Multicollinearity mostly monthly charges and Total charges have more chance of being highly correlated

lets drop anyone of them and check again

In [17]:

```
dfcopy.drop('Monthly Charges', inplace=True, axis=1)
```

In [18]:

```
continuous_columns = continuous_features.remove('Monthly Charges')
vif_calculations(continuous_features, dfcopy)
```

Out[18]:

	Variable	VIF
0	Tenure Months	3.296786
1	Total Charges	3.178741
2	Churn Score	1.066775

Now this dataset doesn't have any Multicollinearity so let's remove "Monthly Charges" Column as it might be using Logistic Regression Model.

In [19]:

```
dataset.drop('Monthly Charges', axis=1, inplace=True)
```

Check Outlier and Capping it

Why outliers?

Data Entry error : Human error.

Measurement error: Instrument error.

Natural error: it will be Real life data.

Intentional error: People give wrong inputs

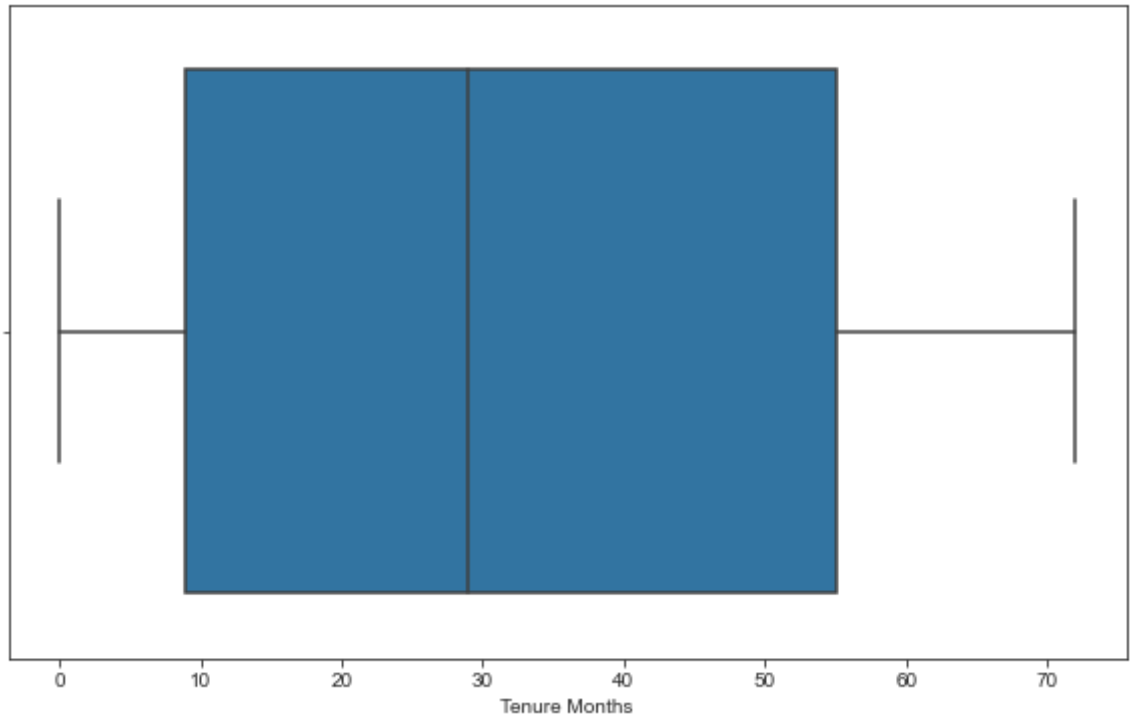
Impact of Outliers ?

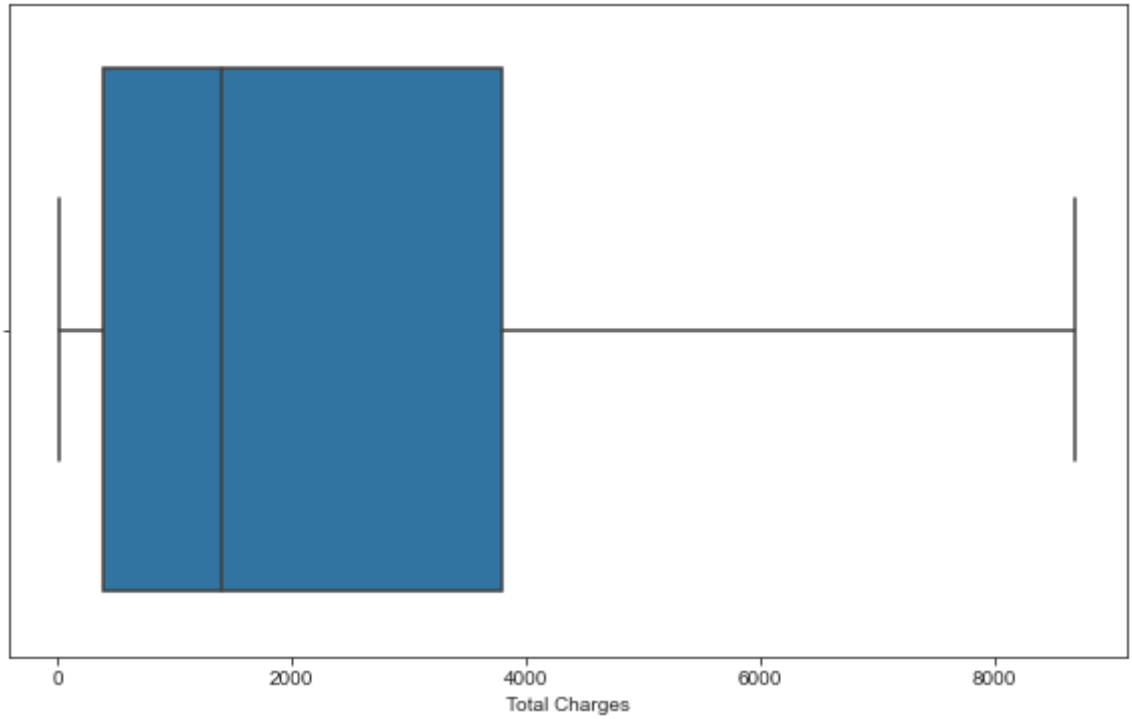
Ouliers can very high impact on few Machine learning models.

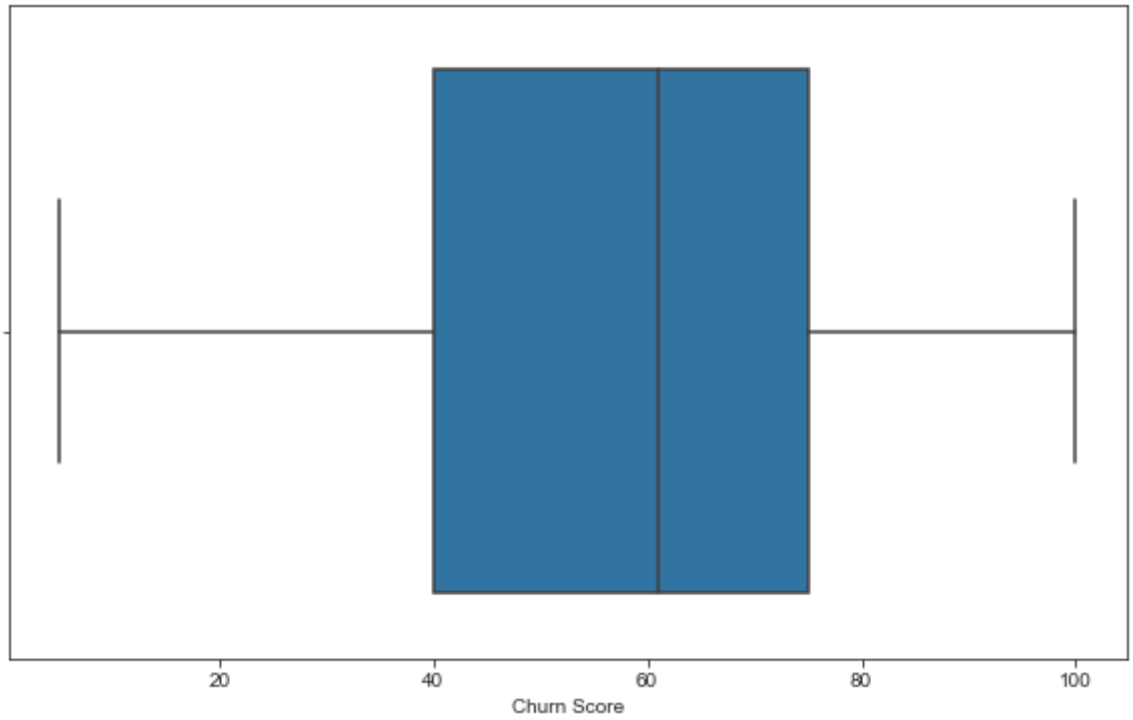
Can Cause bias in the output.

In [20]:

```
for i in continuous_features:
        plt.figure(figsize=(10,6))
        sns.set_style('ticks')
        ax = sns.boxplot(dataset[i])
```








```
dataset[continuous_features].skew(axis=0)
```

Out[21]:

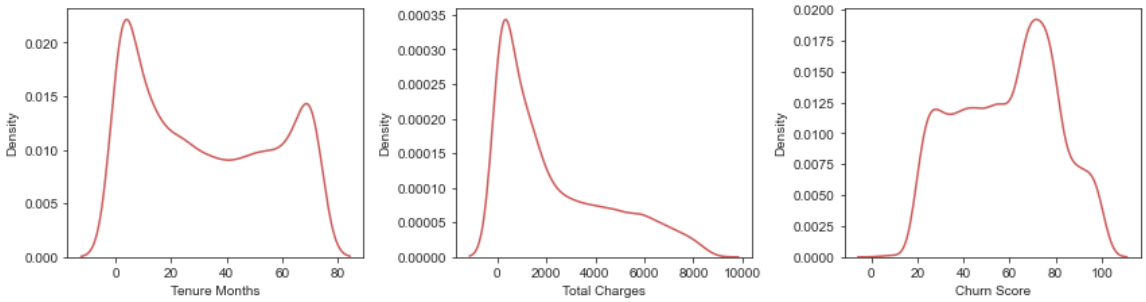
```
Tenure Months    0.239540
Total Charges    0.961642
Churn Score     -0.089840
dtype: float64
```

There is no specific threshold for skewness that is universally accepted as a definitive indication of whether a distribution is significantly skewed or not. However, some general guidelines are often used to interpret the magnitude of the skewness value.

Typically, a distribution is seen to be considerably skewed if its skewness value is larger than +1 or less than -1. Moderate skewness is defined as a skewness value among +0.5 and +1 or between -0.5 and -1. Last but not least, skewness values among -0.5 and +0.5 are regarded as symmetric roughly.

In [22]:

```
# distribution of data before scaling
plt.figure(figsize=(12, 6))
for i, col in enumerate(continuous_features):
    plt.subplot(2, 3, i+1)
    sns.kdeplot(x=dataset[col], color='indianred')
    plt.xlabel(col)
    plt.tight_layout()
```



Total Charges is skewed so we can apply Power Transformer to make it normally distributed.

SPLIT X and Y

In [23]:

```
from sklearn.model_selection import train_test_split
X = dataset.drop(['Churn Value'], axis=1)
y = dataset['Churn Value']
```

Divide the Data frame into X and Y.

Here, the independent Columns are set as Variable X and the Dependent Column (Churn Value) is set as Variable Y.

In [24]:

```
for feature in categorical_feature:
    print(feature,':', X[feature].nunique())
```

```
City : 1129
Senior Citizen : 2
Partner : 2
Dependents : 2
Multiple Lines : 3
Internet Service : 3
Online Security : 3
Online Backup : 3
Device Protection : 3
Tech Support : 3
Streaming TV : 3
Streaming Movies : 3
Contract : 3
Paperless Billing : 2
Payment Method : 4
```

FEATURE ENCODING AND SCALING

One Hot Encoding for Columns which had lesser unique values and not ordinal

In [25]:

```
for i in continuous_features:
    print(f'{i}: mean = {dataset[i].mean():.2f}, median = {dataset[i].median():.2f}')
```

```
Tenure Months: mean = 32.37, median = 29.00
Total Charges: mean = 2283.30, median = 1397.47
Churn Score: mean = 58.70, median = 61.00
```

Mean imputer: Since we handeled outlier for the continuous features the mean and median of the features are nearly same. When there are no outliers mean performs a better imputer.

Standardscaler: As the features distribution are nearly normal we use standard scaler.

Power Transformer: Since Total Charges are skewed we use Power Transformer on them

In [26]:

```
# Create Column Transformer using 3 type of transformers
categorical_feature = X.select_dtypes(include="object").columns
num_feature = Telco_numeric_features.copy()
num_feature.remove('Churn Value')
num_feature.remove('Monthly Charges')
num_feature.remove('Total Charges')

transform_features=['Total Charges']

from sklearn.preprocessing import OneHotEncoder, StandardScaler, PowerTransformer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer


transform_pipe = Pipeline(steps=[
    ('transformer', PowerTransformer()),
    ('imputer', SimpleImputer(strategy='mean')),
    ('scaler', StandardScaler(with_mean=False))
])

preprocessor = ColumnTransformer(
    [
        ("numeric_Pipeline",StandardScaler(),num_feature),
        ("Categorical_Pipeline",OneHotEncoder(), categorical_feature),
        ("Power_Transformation", transform_pipe, transform_features)
]
)
```

In [27]:

```
num_feature
```

Out[27]:

```
['Tenure Months', 'Churn Score']
```

In [28]:

```
X= preprocessor.fit_transform(X)
```

HANDLING IMBALANCED DATASET

Dealing with an unbalanced target variable.

SMOTE, also known as synthetic minority oversampling technique, is another approach to oversampling the minority class. There are many situations when adding duplicate minority class entries to the model does not result in the addition of additional data.

Class imbalance is effectively addressed using the well-known oversampling strategy known as SMOTE. The purpose of integrating SMOTE with other undersampling algorithms (ENN, Tomek) is to increase the efficiency of managing the unbalanced class.

Why Smote tomek?

A hybrid method called SMOTE-Tomek tries to remove overlapping data points for each classes scattered in sample space. This approach employs Tomek Links to eliminate overlapping observations of the two classes and combines the SMOTE capability to produce synthetic data to the minority class.

To add new data of minority class:

1) Pick random information from the minority class. 2) Determine the separation between randomly generated data & its k closest neighbours. 3) As a synthetic sample, combine the difference with a random numb between 0 & 1, then its will adds the results to minority class. 4) Keep going back to steps 2-3 until the necessary percentage(%) of the minority class reach.

To remove the overlapping data points of both classes:

1) In the dataset, look for Tomek Links—pairs of samples that are closest neighbours but belong to distinct classes. 2) Remove the sample from each Tomek Link that belongs to the dominant class. 3) Repeat step 1–2 until there are no more Tomek Links in the dataset.

MODEL SELECTION

Here should understand the Various Classification models with default values from these models we can choose top 4 with Highest Accuracy score and proceed with HyperParameter Tuning

TRAIN & TEST SPLIT

The Train and Test split technique are using to Train the Models with dataset then we can evaluate the ML models performance with Test Data.

Here we can split the dataset into 80-20 Combination. 80 % of Data from dataset to Train the model & 20% Data from the dataset to Test.

Import Required packages for model training

In [29]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, AdaBoost
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, ConfusionMatrixDisplay
                            precision_score, recall_score, f1_score, roc_auc_score, roc_c
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
import time
```

In [30]:

```
def evaluate_clf(true, predicted):
    acc = accuracy_score(true, predicted) # Calculate Accuracy
    f1 = f1_score(true, predicted) # Calculate F1-score
    precision = precision_score(true, predicted) # Calculate Precision
    recall = recall_score(true, predicted)  # Calculate Recall
    roc_auc = roc_auc_score(true, predicted) #Calculate Roc
    return acc, f1 , precision, recall, roc_auc
```

In [31]:

```
import random
seed_value = 1234
random.seed(seed_value)
```

In [32]:

```
# Initialize models which are required for models
models = {
    "Logistic Regression": LogisticRegression(),
    "Gradient Boosting": GradientBoostingClassifier(),
    "CatBoosting Classifier": CatBoostClassifier(verbose=False),
    "Random Forest": RandomForestClassifier(),
    "Decision Tree": DecisionTreeClassifier(),
    "K-Neighbors Classifier": KNeighborsClassifier(),
    "XGBClassifier": XGBClassifier(),
    "AdaBoost Classifier": AdaBoostClassifier()
}
```

Why it is not really a good idea to perform upsampling (For e.g, using SMOTE) before the train-test split in the context of imbalanced classification problems.

This approach can lead to two problems:

- **Data leakage:** When upsampling is conducted prior to the train-test split, some of the duplicated or synthetic samples generated may end up in both the training & test sets, which means that the model is being evaluated on some samples that it has already seen during training. This violates the fundamental requirement of having a test set that is independent of the training dataset and can lead to over-optimistic performance estimates.
- **Biased performance measures:** Upsampling before the train-test split can also lead to artificially balanced test sets, which can result in misleading performance estimates. The goal of the test set is to obtain performance estimates that are representative of the real, unseen data, which is imbalanced in this case. By artificially balancing the test set, The estimated performance may not accurately reflect the model's capacity to generalise to previously unknown data.

Although SMOTE generates synthetic samples Which are "close" to real minority classes. The issues outlined above still apply to some extent. To avoid these problems, we are using a pipeline approach where SMOTE is applied only to the training set after the train-test split.

In [33]:

```
from imblearn.combine import SMOTETomek

def evaluate_models(X, y, models):
    X_trains, X_tests, y_trains, y_tests = train_test_split(X, y, test_size=0.2, random_s

    # Applying SMOTE Tomek to the training set only
    smt = SMOTETomek(random_state=42,sampling_strategy='minority')
    X_trains, y_trains = smt.fit_resample(X_trains, y_trains)

    models_list = []
    accu_list = []
    f1_list = []
    preci_list = []
    recall_list = []
    rocauc_list = []
    execution_time = []

    for name, model in models.items():
        start_time = time.time() # Start the timer
        model.fit(X_trains, y_trains) # Train model
        end_time = time.time() # End the timer

        # Make the predictions
        y_train_preds = model.predict(X_trains)
        y_test_preds = model.predict(X_tests)

        # Trainingset performance
        train_accuracys, trains_f1, train_precisions, train_recalls, train_rocaucs = eval

        # Testset performance
        test_accuracys, tests_f1, test_precisions, test_recalls, test_rocaucs = evaluate_

        print("Model :" ,name)

        print('Performance of the Model')
        print("TrainSet - Accuracy: {:.4f}".format(train_accuracys))
        print('TrainSet - F1 score: {:.4f}'.format(trains_f1))
        print('TrainSet - Precision: {:.4f}'.format(train_precisions))
        print('TrainSet - Recall: {:.4f}'.format(train_recalls))
        print('TrainSet - Roc Auc Score: {:.4f}'.format(train_rocaucs))

        print('----------------------------------')

        print('Performance of the Model')
        print('TestSet - Accuracy: {:.4f}'.format(test_accuracys))
        print('TestSet - F1 score: {:.4f}'.format(tests_f1))
        print('TestSet - Precision: {:.4f}'.format(test_precisions))
        print('TestSet - Recall: {:.4f}'.format(test_recalls))
        print('TestSet - Roc Auc Score: {:.4f}'.format(test_rocaucs))
        print('='*35)
        print('\n')
        # Append evaluation metrics to lists
        models_list.append(name)
        accu_list.append(test_accuracys)
        f1_list.append(tests_f1)
        preci_list.append(test_precisions)
        recall_list.append(test_recalls)
        rocauc_list.append(test_rocaucs)
        execution_time.append(end_time - start_time) # Append the execution time to the l
```

```
    # Create and return a dataframe with evaluation results
    report = pd.DataFrame({
        'Model Name': models_list,
        'Accuracy': accu_list,
        'F1 Score': f1_list,
        'Precision': preci_list,
        'Recall': recall_list,
        'ROC AUC Score': rocauc_list,
        'Execution Time': execution_time
    }).sort_values(by=['Accuracy'], ascending=False).reset_index(drop=True)

    return report
```

EVALUATE ALL BASE MODELS

In [34]:

```
base_report =evaluate_models(X=X, y=y, models=models)
```

```
Model : Logistic Regression
Performance of the Model
TrainSet - Accuracy: 0.9504
TrainSet - F1 score: 0.9511
TrainSet - Precision: 0.9383
TrainSet - Recall: 0.9643
TrainSet - Roc Auc Score: 0.9504
----------------------------------
Performance of the Model
TestSet - Accuracy: 0.9070
TestSet - F1 score: 0.8471
TestSet - Precision: 0.7943
TestSet - Recall: 0.9075
TestSet - Roc Auc Score: 0.9072
==================================


Model : Gradient Boosting
Performance of the Model
```

In [35]:

```
base_report
```

Out[35]:

	Model Name	Accuracy	F1 Score	Precision	Recall	ROC AUC Score	Execution Time
0	XGBClassifier	0.923350	0.866995	0.854369	0.8800	0.910268	0.658550
1	AdaBoost Classifier	0.918382	0.863905	0.820225	0.9125	0.916607	0.771366
2	CatBoosting Classifier	0.914123	0.855090	0.820690	0.8925	0.907598	12.100657
3	Gradient Boosting	0.913414	0.856471	0.808889	0.9100	0.912384	2.946455
4	Decision Tree	0.911285	0.847747	0.826603	0.8700	0.898826	0.158425
5	Logistic Regression	0.907026	0.847141	0.794311	0.9075	0.907169	0.148482
6	Random Forest	0.902768	0.823226	0.850667	0.7975	0.871000	5.802184
7	K-Neighbors Classifier	0.832505	0.760163	0.640411	0.9350	0.863437	0.001676

From this report, We analyzed several ML models to predict customer Attrition for a telecom company. The dataset used in this study was imbalanced, with the minority class representing the customers who churn and the majority class representing customers who do not churn.

We evaluated the performance of the following models: CatBoosting Classifier, Gradient Boosting, XGBClassifier, AdaBoost Classifier, Logistic Regression, Random Forest, Decision Tree as well as K-Neighbors Classifier. Here, We utilized accuracy, F1 score, precision, recall, ROC AUC score, and execution time as evaluation metrics.

Based on the evaluation, we recommend using the `CatBoosting Classifier` or `Gradient Boosting` models as they performed the best across all evaluation metrics. However, it is important to note that these models had relatively longer execution times compared to other models. This may impact the hyperparameter tuning process and should be considered when selecting a model for practical deployment.

Since the execution time of the CatBoosting Classifier is much higher than that of the Gradient Boosting model (6 seconds vs 2 seconds), the Gradient Boosting model might be a better choice if we consider the execution time factor and AUC score.

Future experiment can focus on further improving the performance of the models by hyperparameter tuning with 3 best models.

Here we can use Gradient Boosting , XGBClassifier, AdaBoost Classifier for Hyper Parameter Tuning

Best Base Model is Gradient Boosting with 0.914513 ROC AUC Score

In [36]:

```
param_grid_gbc = {
    "n_estimators":[5,50,100,150,200,250],
    "max_depth":[1,3,5,7,9,10,12],
    "learning_rate":[(0.96 + x / 100) for x in range(0, 6)]
}

param_grid_xgb = {
    "learning_rate": [0.1, 0.01],
    "max_depth": [5, 8, 12, 20, 30, 40],
    "n_estimators": [150, 250, 350],
    "colsample_bylevel": [0.5, 0.8, 1, 0.3, 0.4]
}

param_grid_abc = {
    'n_estimators': [2, 3, 4, 5, 6, 7, 8, 9, 10],
    'learning_rate': [(0.96 + x / 100) for x in range(0, 7)],
    'algorithm': ['SAMME', 'SAMME.R']
}
```

In [37]:

```
# Models list for Hyperparameter tuning
randomcv_models = [
    ('XGBClassifier', XGBClassifier(), param_grid_xgb),
    ('Gradient Boosting', GradientBoostingClassifier(), param_grid_gbc),
    ("AdaBoost Classifier", AdaBoostClassifier(), param_grid_abc),
]
```

Its Commonly Recommendable to balance the dataset before "Hyper parameter Tuning" because hyper parameter tuning is used to optimize the performance of the given ML algorithm on a selected dataset. However, if the dataset is imbalanced, the performance metrics like Accuracy, Precision and Recall can be misleading. Therefore , its good to handle the imbalanced dataset before tuning the hyperparameters to make sure that the model performance is optimized on a balanced dataset.

In [38]:

```
from imblearn.combine import SMOTETomek
from sklearn.model_selection import RandomizedSearchCV

# Define the samplers
smt = SMOTETomek()

X_res, y_res = smt.fit_resample(X, y)

randomcv_param = {}
for modelname, modelobj, parameters in randomcv_models:
    random = RandomizedSearchCV(estimator=modelobj,
                                param_distributions=parameters,
                                n_iter=120,
                                cv=2,
                                verbose=2,
                                n_jobs=-1)
    random.fit(X_res, y_res)
    randomcv_param[modelname] = random.best_params_

for model_name in randomcv_param:
    print(f"---------------- Best Parameters for the {model_name} is -------------------"
    print(randomcv_param[model_name])
```

```
Fitting 2 folds for each of 120 candidates, totalling 240 fits
Fitting 2 folds for each of 120 candidates, totalling 240 fits
Fitting 2 folds for each of 120 candidates, totalling 240 fits
---------------- Best Parameters for the XGBClassifier is ----------------
---
{'n_estimators': 350, 'max_depth': 8, 'learning_rate': 0.01, 'colsample_by
level': 0.3}
---------------- Best Parameters for the Gradient Boosting is ------------
-------
{'n_estimators': 100, 'max_depth': 12, 'learning_rate': 0.97}
---------------- Best Parameters for the AdaBoost Classifier is ----------
---------
{'n_estimators': 7, 'learning_rate': 0.98, 'algorithm': 'SAMME.R'}
```

Retraining the Model with Parameters

In [39]:

```
best_models = {
    "XGBClassifier": XGBClassifier(**randomcv_param['XGBClassifier'],n_jobs=-1),
    "AdaBoost Classifier" : AdaBoostClassifier(**randomcv_param['AdaBoost Classifier']),
    "Gradient Boosting" : GradientBoostingClassifier(**randomcv_param['Gradient Boosting'
}
tuned_report =evaluate_models(X=X, y=y, models=best_models)
```

```
Model : XGBClassifier
Performance of the Model
TrainSet - Accuracy: 0.9619
TrainSet - F1 score: 0.9624
TrainSet - Precision: 0.9489
TrainSet - Recall: 0.9764
TrainSet - Roc Auc Score: 0.9619
----------------------------------
Performance of the Model
TestSet - Accuracy: 0.9198
TestSet - F1 score: 0.8656
TestSet - Precision: 0.8254
TestSet - Recall: 0.9100
TestSet - Roc Auc Score: 0.9168
==================================


Model : AdaBoost Classifier
Performance of the Model
TrainSet - Accuracy: 0.9325
TrainSet - F1 score: 0.9338
TrainSet - Precision: 0.9162
TrainSet - Recall: 0.9520
TrainSet - Roc Auc Score: 0.9325
----------------------------------
Performance of the Model
TestSet - Accuracy: 0.9141
TestSet - F1 score: 0.8620
TestSet - Precision: 0.7925
TestSet - Recall: 0.9450
TestSet - Roc Auc Score: 0.9234
==================================


Model : Gradient Boosting
Performance of the Model
TrainSet - Accuracy: 1.0000
TrainSet - F1 score: 1.0000
TrainSet - Precision: 1.0000
TrainSet - Recall: 1.0000
TrainSet - Roc Auc Score: 1.0000
----------------------------------
Performance of the Model
TestSet - Accuracy: 0.9099
TestSet - F1 score: 0.8453
TestSet - Precision: 0.8242
TestSet - Recall: 0.8675
TestSet - Roc Auc Score: 0.8971
==================================
```

In [40]:

```
tuned_report
```

Out[40]:

	Model Name	Accuracy	F1 Score	Precision	Recall	ROC AUC Score	Execution Time
0	XGBClassifier	0.919801	0.865636	0.825397	0.9100	0.916843	2.094381
1	AdaBoost Classifier	0.914123	0.862030	0.792453	0.9450	0.923442	0.135191
2	Gradient Boosting	0.909865	0.845311	0.824228	0.8675	0.897080	9.615283

These 3 models such as Gradient Boosting, XGBClassifier and AdaBoost Classifier were trained and evaluated on the selected dataset. All these 3 models were trained with the Tuned Hyperparameter.

Based on the results, all three models have similar performance, with accuracy and F1 scores ranging from 0.909 to 0.920. However, when dealing with an imbalanced dataset, we need to pay special attention to the recall metric, which measures the ability of the model to correctly identify positive cases (in this case, customers who churn). Therefore, we should prioritize the model with the highest recall score.

However, it's very essential to note, the choice of model should not be based on a single metric, but rather on a combination of metrics and other factors such as interpretability, execution time, and computational resources required. In this case, all three models have similar execution times, and the XGBClassifier has the highest score of 0.92, which indicates good overall performance in terms of ranking the positive and negative classes.

XGBClassifier is an implementation of the gradient boosting algorithm with an efficient tree-based implementation. It's a supervised learning algorithm which is used for both regression & classification problems. The algorithm works based on by building an ensemble of weak decision tree models and improving those performance by adjusting the weights of misclassified observations in each iteration.

Conclusion: In conclusion, the XGBClassifier is the best model for predicting customer churn in this imbalanced dataset, based on the recall metric, ROC AUC Score, and overall performance.

With Imbalanced dataset

In [41]:

```
X_1 = dataset.drop(['Churn Value'], axis=1)
y_1 = dataset['Churn Value']

X_1 = preprocessor.fit_transform(X_1)
```

In [42]:

```
X1_trains, X1_tests, y1_trains, y1_tests = train_test_split(X_1, y_1, test_size=0.2, rand
```

In [43]:

```
# class weights to account for imbalance
class_weights = len(y1_trains) / (2 * pd.Series(y1_trains).value_counts())
```

In [44]:

```
# Train an XGBoost model with class weights
model = XGBClassifier(scale_pos_weight=class_weights[1])
model.fit(X1_trains, y1_trains)
```

Out[44]:

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types
=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_typ
e=None,
              interaction_constraints=None, learning_rate=None, max_bin
=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
```

In [45]:

```
# Get the original feature names
from sklearn.preprocessing import OneHotEncoder

numeric_feature_names = list(dataset[num_feature].columns)
categorical_feature_names = list(preprocessor.named_transformers_['Categorical_Pipeline']
#categorical_feature_names = list(preprocessor.named_transformers_['Categorical_Pipeline'

transform_feature_names = ['_'.join([f, 'transformed']) for f in transform_features]
feature_names = numeric_feature_names + categorical_feature_names + transform_feature_nam
```

In [46]:

```
y_pred = model.predict(X_1[0])

# Obtain the feature importance scores
feature_importance = model.get_booster().get_score(importance_type='gain')
feature_importance = {feature_names[int(k[1:])]: v for k, v in feature_importance.items()

# Sort the feature importance scores in descending order
sorted_feature_importance = sorted(feature_importance.items(), key=lambda x: x[1], revers

# Print feature importance scores in descending order
print("Feature Importance Scores:")
for feature, score in sorted_feature_importance:
    print(feature, ":", score)

# Print the feature that influenced the most for the predicted output
most_influential_feature = sorted_feature_importance[0][0]
print("\nMost Influential Feature for the Predicted Output:", most_influential_feature)
```

```
Feature Importance Scores:
Contract_Month-to-month : 51.85902404785156
Churn Score : 38.34318923950195
Dependents_No : 8.17722225189209
Internet Service_Fiber optic : 8.152310371398926
Online Security_No : 4.0171356201171875
City_Running Springs : 3.5099925994873047
Internet Service_DSL : 3.44549298286438
Streaming Movies_Yes : 3.0825510025024414
Internet Service_No : 3.052438497543335
Contract_One year : 2.9162003993988037
Online Backup_No : 2.760613441467285
City_Bakersfield : 2.6751062870025635
Payment Method_Electronic check : 2.5471854209899902
Paperless Billing_No : 2.4795477390289307
Contract_Two year : 2.392561197280884
Tenure Months : 2.311992645263672
City_San Jose : 2.244800090789795
City_Santa Rosa : 2.2219886779785156
Device Protection_Yes : 2.216334819793701
Streaming TV_Yes : 2.087562084197998
City_San Dimas : 2.0003838539123535
City_Santa Ana : 1.9791754484176636
City_San Francisco : 1.7374250888824463
City_Los Angeles : 1.7263695001602173
Tech Support_No : 1.7182656526565552
Multiple Lines_No : 1.6847913265228271
City_Elk Grove : 1.6525471210479736
Senior Citizen_No : 1.6392630338668823
Total Charges_transformed : 1.611040472984314
Online Backup_Yes : 1.5796056985855103
City_Fresno : 1.511548399925232
City_Brea : 1.4681085348129272
Tech Support_Yes : 1.4572086334228516
Multiple Lines_No phone service : 1.4341297149658203
Partner_No : 1.4095302820205688
Payment Method_Credit card (automatic) : 1.3560692071914673
Streaming Movies_No : 1.337954044342041
Online Security_Yes : 1.250910997390747
Device Protection_No : 1.2299853563308716
Multiple Lines_Yes : 1.2244288921356201
City_Sacramento : 1.1712853908538818
Payment Method_Bank transfer (automatic) : 1.1553919315338135
City_Walnut Creek : 1.1012881994247437
City_Hayward : 1.0452685356140137
City_Santa Barbara : 0.9947755336761475
City_Modesto : 0.9246544241905212
City_Pasadena : 0.9182264804840088
Payment Method_Mailed check : 0.907774031162262
City_Inglewood : 0.9072531461715698
Streaming TV_No : 0.8799498677253723
City_Glendale : 0.8703967928886414
City_Lakewood : 0.6543239951133728
City_Stockton : 0.6409362554550171
City_Anaheim : 0.5440173745155334
City_Concord : 0.43693965673446655
City_San Diego : 0.41355445981025696
City_Oakland : 0.3499912619590759
City_Chula Vista : 0.2644942104816437
City_San Bernardino : 0.18940097093582153
```

```
Most Influential Feature for the Predicted Output: Contract_Month-to-month
```

Checking accuracy and Roc of imbalanced data model

In [47]:

```
y1_test_pred = model.predict(X1_tests)

# Training the set performances
train_accuracys, trains_f1, train_precisions, train_recalls, train_rocaucs = evaluate_clf
```

In [48]:

```
train_accuracys
```

Out[48]:

```
0.9198012775017743
```

In [49]:

```
train_rocaucs
```

Out[49]:

```
0.9053715693879382
```

In [50]:

```
output = model.predict(X1_tests[6])
output
```

Out[50]:

```
array([1])
```

SHAP

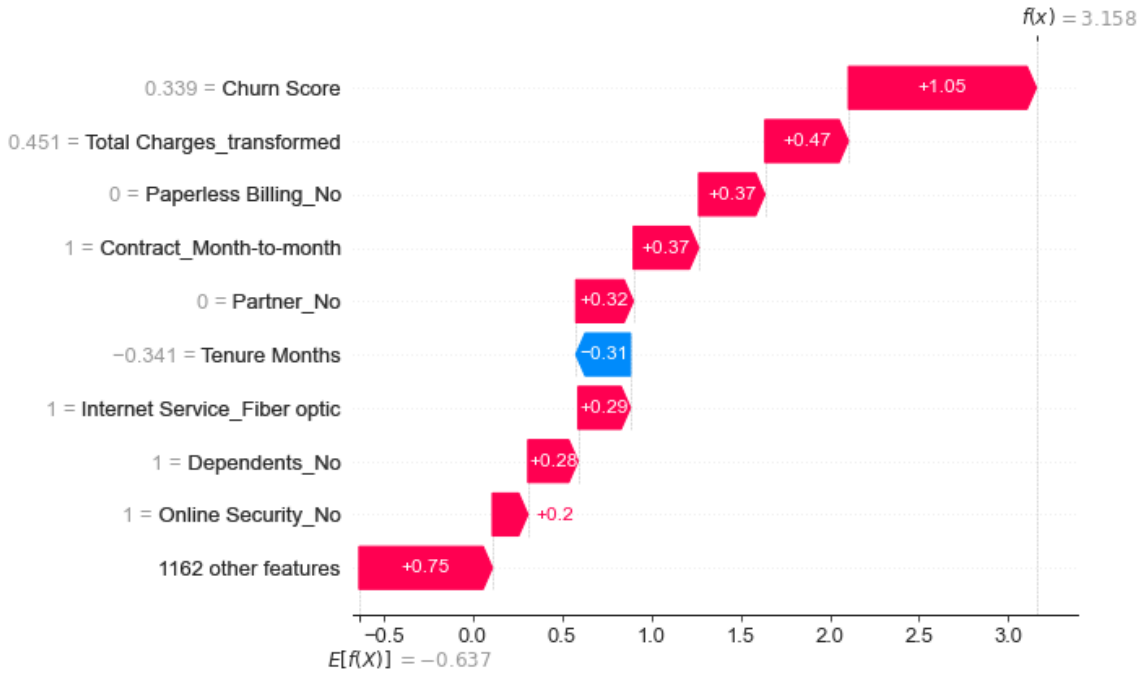
In [51]:

```
import shap

# Create an explainer object for your XGBoost model
explainer = shap.Explainer(model, feature_names=feature_names)

# Calculate 'SHAP values' for a subset of your test data
shap_values = explainer(X1_tests[6])

# Plot the SHAP values for a single instance
shap.plots.waterfall(shap_values[0])
```



In [52]:

```
categorical_colus = dataset.select_dtypes(include=['object']).columns.tolist()

for colu in categorical_colus:
    print(f"Column: {colu}")
    print(f"Unique values: {dataset[colu].unique()}")
```

```
Column: City
Unique values: ['Los Angeles' 'Beverly Hills' 'Huntington Park' ... 'Stand
ish' 'Tulelake'
 'Olympic Valley']
Column: Senior Citizen
Unique values: ['No' 'Yes']
Column: Partner
Unique values: ['No' 'Yes']
Column: Dependents
Unique values: ['No' 'Yes']
Column: Multiple Lines
Unique values: ['No' 'Yes' 'No phone service']
Column: Internet Service
Unique values: ['DSL' 'Fiber optic' 'No']
Column: Online Security
Unique values: ['Yes' 'No' 'No internet service']
Column: Online Backup
Unique values: ['Yes' 'No' 'No internet service']
Column: Device Protection
Unique values: ['No' 'Yes' 'No internet service']
Column: Tech Support
Unique values: ['No' 'Yes' 'No internet service']
Column: Streaming TV
Unique values: ['No' 'Yes' 'No internet service']
Column: Streaming Movies
Unique values: ['No' 'Yes' 'No internet service']
Column: Contract
Unique values: ['Month-to-month' 'Two year' 'One year']
Column: Paperless Billing
Unique values: ['Yes' 'No']
Column: Payment Method
Unique values: ['Mailed check' 'Electronic check' 'Bank transfer (automati
c)'
 'Credit card (automatic)']
```

In [53]:

```
num_columns = dataset.select_dtypes(include='number').columns.tolist()
num_stats = dataset[num_columns].describe().loc[['min', 'max']].transpose()
print(num_stats)
```

```
                min     max
Tenure Months   0.0    72.0
Total Charges  18.8  8684.8
Churn Value     0.0     1.0
Churn Score     5.0   100.0
```

In [54]:

```
city_counts = dataset['City'].value_counts().sort_values(ascending=False)
city_list = city_counts.index.tolist()
```

In [55]:

```
import shap
import pandas as pd
import numpy as np

class XGBoostModel:
    def __init__(self, model=model, preprocessor=preprocessor, feature_names=feature_name
        self.model = model
        self.preprocessor = preprocessor
        self.feature_names = feature_names
        self.city_list = city

    def predict(self, input_dict):
        # Convert input dictionary to dataframe
        input_df = pd.DataFrame.from_dict(input_dict, orient='index').transpose()

        # Preprocess input dataframe using preprocessor
        X = self.preprocessor.transform(input_df)

        # Make prediction using XGBoost model
        y_pred = self.model.predict(X)[0]

        # Compute SHAP values for input data
        explainer = shap.Explainer(self.model, feature_names=self.feature_names)
        shap_values = explainer(X)

        feedback = "No feedback"
        if y_pred == 1:
            # Sort the 'SHAP values' by descending order
            sorted_index = np.argsort(shap_values.values[0])[::-1]

            # Extract the names of the top 2 features with highest SHAP values
            top_2_features =  np.take(feature_names, sorted_index)[:2]

            # Generate feedback based on the negative features
            if len(top_2_features) == 0:
                feedback = "Based on the provided information, we do not see any major is
            else:
                feedback = f"Based on the provided information, the following are top 2 f

        return y_pred, shap_values, feedback
```

Testing SHAP Calculation using Sample Input

In [56]:

```
input_dict = {
'City': 'Los Angeles',
'Senior Citizen': 'No',
'Partner': 'No',
'Dependents': 'No',
'Tenure Months': 2,
'Multiple Lines': 'No',
'Internet Service': 'DSL',
'Online Security': 'Yes',
'Online Backup': 'Yes',
'Device Protection': 'No',
'Tech Support': 'No',
'Streaming TV': 'No',
'Streaming Movies': 'No',
'Contract': 'Month-to-month',
'Paperless Billing': 'Yes',
'Payment Method': 'Mailed check',
'Total Charges': 108.15,
'Churn Score': 86
}
```

In [57]:

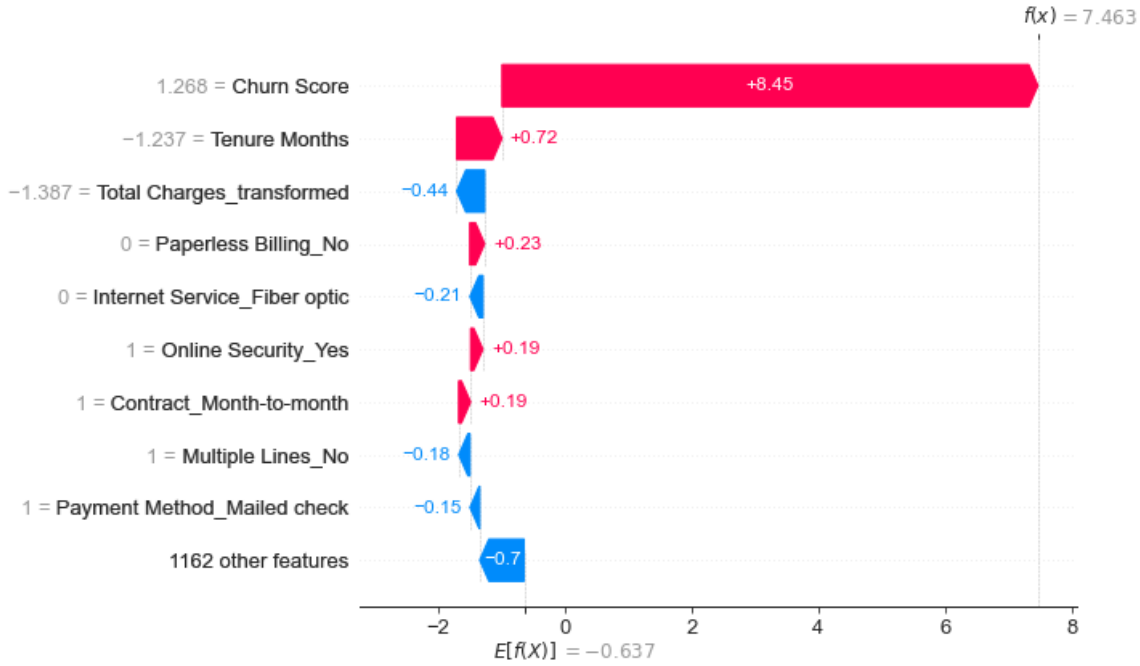
```
final_model = XGBoostModel()
```

In [58]:

```
# Make prediction and generate SHAP waterfall plot
y_pred, shap_plot,feedback = final_model.predict(input_dict)

# Show prediction and SHAP waterfall plot
print(" Prediction:", y_pred, "\n", "Feedback:", feedback)
shap.plots.waterfall(shap_plot[0])
```

```
 Prediction: 1
 Feedback: Based on the provided information, the following are top 2 fact
ors may increase the likelihood of a churn: Churn Score, Tenure Months. Pl
ease consider addressing these issues to improve customer retention.
```



In [59]:

```
# Save the instance to a file using pickle
import pickle
import os
def save_object(file: object, filename: str):
    os.makedirs('artifacts', exist_ok=True)
    with open(f'artifacts/{filename}.pkl', 'wb') as f:
        pickle.dump(file, f)
```

In [60]:

```
save_object(file=city_list, filename='city_list')
save_object(file=model, filename='xgb_model')
save_object(file=preprocessor, filename='preprocessor')
save_object(file=feature_names, filename='feature_names')
```

In [ ]:

```

```