Predicting Customer Churn

Identifying Customer that are susceptible to Churn(leave) in order to enhance retention strategies and boost Sales/Business Growth

Import packages and libriries

In [1]:
```
from ast import literal_eval  #literal_eval allows  computation on strings
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt  #to visualize
import seaborn as sns
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier  #helps predict
from tqdm import tqdm  #helps visualize the progession of loops and action in py

import warnings      #to avoid seeing warings in pandas packages
warnings.filterwarnings('ignore')
```

In [2]:
```
#data collection
#Load the dataset into Python with pandas
df = pd.read_excel("Dataset.xlsx")
```

Data Exploration

In this section:

- understanding the data structure, statictics and quality of the dataset
- visualizing the data to gain insughts
- checking for missing values

In [3]:
```
#show the fisrt 5 rows of the dataset
df.head()
```

Out[3]:

	CustomerID	Name	Age	Gender	Location	Email	
0	1001	Mark Barrett	31	Male	Andrewfort	allison74@example.net	
1	1002	Jeremy Welch	66	Female	Millerhaven	fmiller@example.com	2
2	1003	Brandon Patel	36	Female	Lozanostad	jasonbrown@example.org	
3	1004	Tina Martin	62	Female	South Dustin	matthew62@example.net	0
4	1005	Christopher Rodriguez	68	Female	West James	shannonstrickland@example.org	

5 rows × 21 columns

In [4]:
```
#handling missing vales
missing_values = df.isnull().sum()
missing_values
```

Out[4]:
```
CustomerID                0
Name                      0
Age                       0
Gender                    0
Location                  0
Email                     0
Phone                     0
Address                   0
Segment                   0
PurchaseHistory           0
SubscriptionDetails       0
ServiceInteractions       0
PaymentHistory            0
WebsiteUsage              0
ClickstreamData           0
EngagementMetrics         0
Feedback                  0
MarketingCommunication    0
NPS                       0
ChurnLabel                0
Timestamp                 0
dtype: int64
```

In [5]:
```
#check for duplicate values
df.duplicated().any()
```

Out[5]:
```
False
```

In [6]:
```
#stattistics overview of the  numerial values of the dataset
df.describe()
```

Out[6]:

	CustomerID	Age	NPS	ChurnLabel
count	12483.00000	12483.000000	12483.000000	12483.000000
mean	7242.00000	43.930065	2.973884	0.505808
std	3603.67604	15.341521	2.644623	0.499986
min	1001.00000	18.000000	0.000000	0.000000
25%	4121.50000	31.000000	1.000000	0.000000
50%	7242.00000	44.000000	2.000000	1.000000
75%	10362.50000	57.000000	4.000000	1.000000
max	13483.00000	70.000000	9.000000	1.000000

How the target varible, CHURN LABEL are distributed

In [7]:
```
#setup the figure and axes
fig, ax = plt.subplots(nrows=2, ncols=2, figsize= (15, 10))

#Plot the distribution of the target variable ChurnLabel
sns.countplot(x='ChurnLabel', data=df, ax=ax[0,0], palette=['green', 'pink'])
ax[0, 0].set_title('Distribution of ChurnLabel')
ax[0, 0].set_xticklabels(['No Churn', 'Churn'])
```

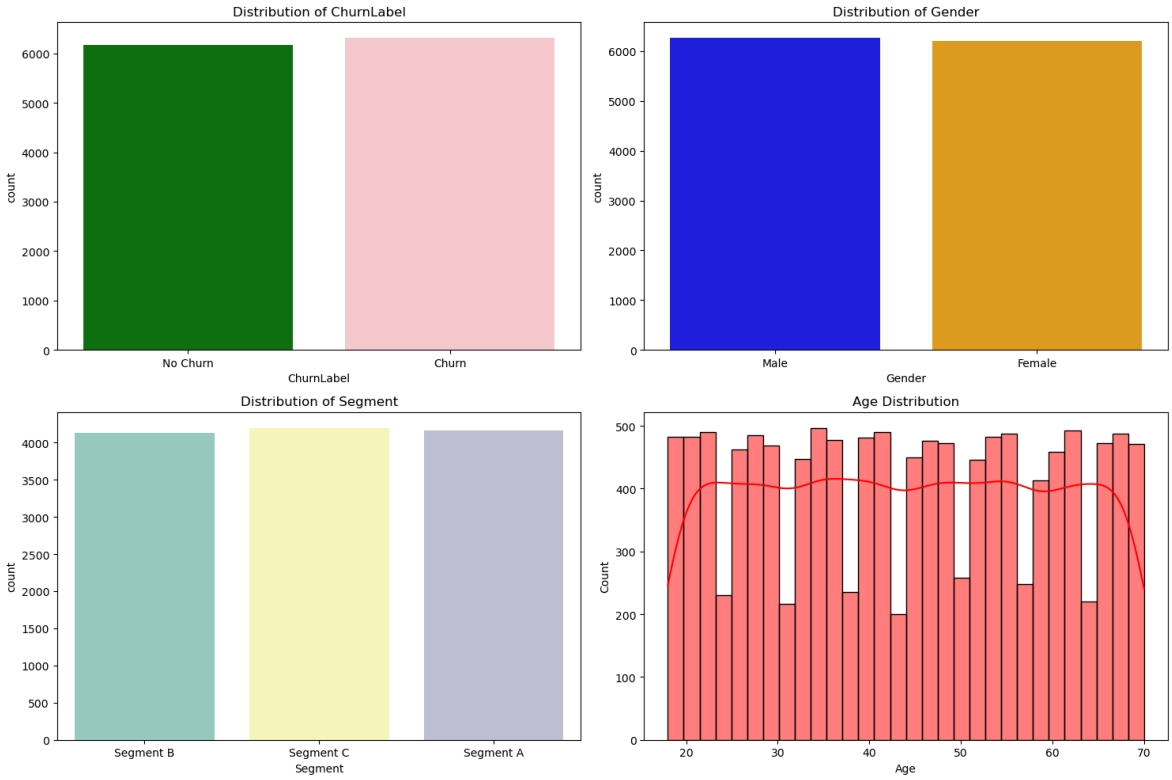
```
#Plot distribution of the Gender
sns.countplot(x='Gender', data=df, ax=ax[0, 1],  palette=['blue', 'orange'])
ax[0, 1].set_title('Distribution of Gender')

#Plot distribution  of Segment
sns.countplot(x='Segment', data=df, ax=ax[1, 0], palette='Set3')
ax[1, 0].set_title('Distribution of Segment')

#Age distribution
sns.histplot(df['Age'], bins=30, ax=ax[1, 1], kde=True, color='red')
ax[1, 1].set_title('Age Distribution')

plt.tight_layout()
plt.show
```

Out[7]: <function matplotlib.pyplot.show(close=None, block=None)>

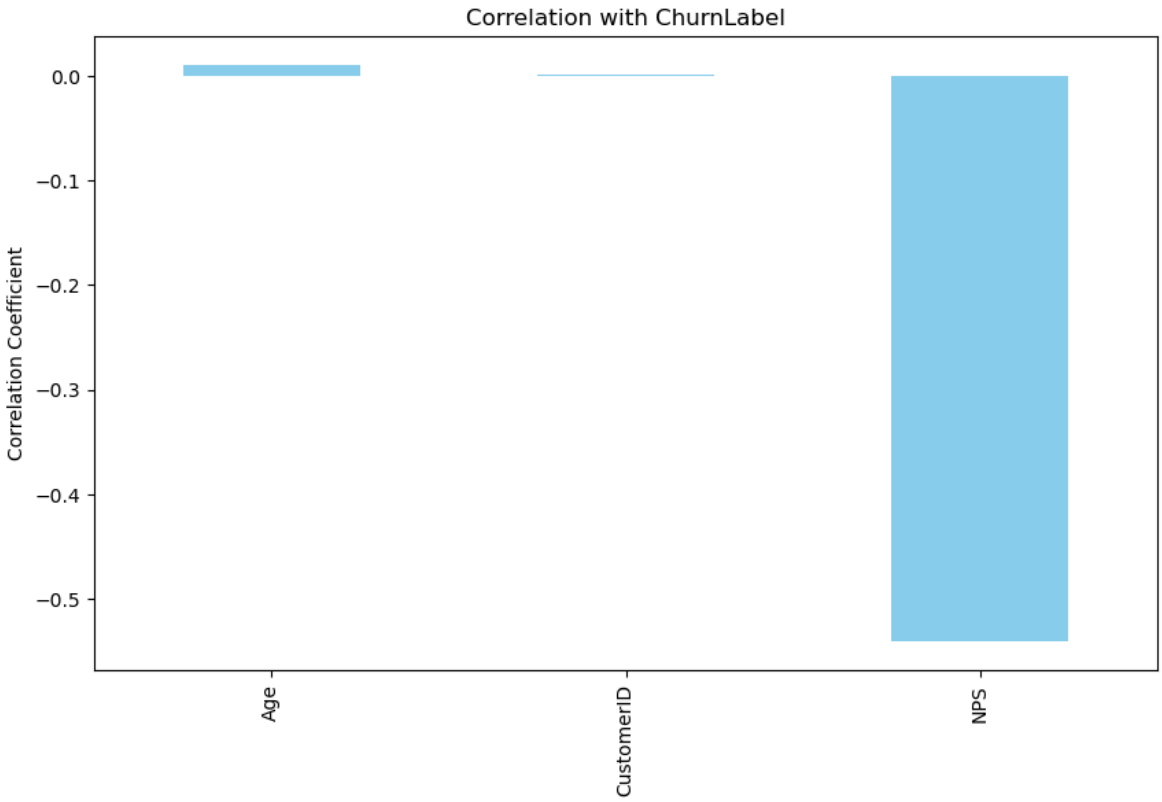


Correlation Analysis: Which columns in the dataset correlates with the 'ChurnLabel' column.

In [8]:
```
#Calculate the correlation with the ChurnLabel
correlation = df.select_dtypes(include=np.number).corr()['ChurnLabel'].sort_valu

#Plot the correlation
plt.figure(figsize=(10, 6))
correlation.drop('ChurnLabel').plot(kind='bar', color='skyblue')
plt.title('Correlation with ChurnLabel')
plt.ylabel('Correlation Coefficient')
plt.show
```

Out[8]: <function matplotlib.pyplot.show(close=None, block=None)>



I am exploring the nested columns and see how they are nested to further explore the dataset

In [9]:
```
#List out the nested columns
nested_columns = [
    'PurchaseHistory',
    'SubscriptionDetails',
    'ServiceInteractions',
    'PaymentHistory',
    'WebsiteUsage',
    'ClickstreamData',
    'EngagementMetrics',
    'Feedback',
    'MarketingCommunication'
]

#Print the fisrt value in each of the columns for a detailed view
w1, w2 = 25, 1000#(width , name)
for col in nested_columns:
    row = [col, df[col][0]]
    print ('\n| {:<{w1}} | {:<{w2}} | '.format(*row, w1=w1, w2=w2))
```

| PurchaseHistory | [{'Product': 'Frozen Cocktail Mixes', 'Frequency': 8, 'Value': 884.43}, {'Product': 'Printer, Copier & Fax Machine Accessories', 'Frequency': 7, 'Value': 397.14}, {'Product': 'Hockey Stick Care', 'Frequency': 10, 'Value': 498.92}, {'Product': 'Guacamole', 'Frequency': 2, 'Value': 718.43}, {'Product': 'Mortisers', 'Frequency': 2, 'Value': 614.08}, {'Product': 'Rulers', 'Frequency': 6, 'Value': 221.68}, {'Product': 'Invitations', 'Frequency': 3, 'Value': 660.04}]
|

| SubscriptionDetails | {'Plan': 'Express', 'Start_Date': '2020-06-08', 'End_Date': '2022-10-27'}
|

| ServiceInteractions | [{'Type': 'Call', 'Date': '2019-09-26'}, {'Type': 'Chat', 'Date': '2021-07-25'}, {'Type': 'Email', 'Date': '2020-04-13'}, {'Type': 'Chat', 'Date': '2020-11-15'}]
|

| PaymentHistory | [{'Method': 'Credit Card', 'Late_Payments': 5}, {'Method': 'PayPal', 'Late_Payments': 11}, {'Method': 'Bank Transfer', 'Late_Payments': 24}]
|

| WebsiteUsage | {'PageViews': 49, 'TimeSpent(minutes)': 15}
|

| ClickstreamData | [{'Action': 'Add to Cart', 'Page': 'register', 'Timestamp': '2020-09-13 17:06:44'}, {'Action': 'Search', 'Page': 'login', 'Timestamp': '2022-03-30 14:51:52'}, {'Action': 'Click', 'Page': 'about', 'Timestamp': '2019-11-10 05:48:48'}, {'Action': 'Add to Cart', 'Page': 'terms', 'Timestamp': '2019-05-15 10:17:44'}, {'Action': 'Add to Cart', 'Page': 'author', 'Timestamp': '2022-07-14 03:40:53'}, {'Action': 'Search', 'Page': 'main', 'Timestamp': '2019-01-13 08:39:42'}, {'Action': 'Add to Cart', 'Page': 'faq', 'Timestamp': '2019-02-19 05:28:25'}, {'Action': 'Add to Cart', 'Page': 'about', 'Timestamp': '2020-11-01 20:59:55'}, {'Action': 'Click', 'Page': 'faq', 'Timestamp': '2021-12-22 16:39:40'}, {'Action': 'Add to Cart', 'Page': 'main', 'Timestamp': '2020-11-11 03:25:36'}, {'Action': 'Click', 'Page': 'privacy', 'Timestamp': '2021-06-13 06:18:41'}, {'Action': 'Add to Cart', 'Page': 'search', 'Timestamp': '2022-03-28 16:25:35'}, {'Action': 'Search', 'Page': 'homepage', 'Timestamp': '2019-09-26 12:27:42'}, {'Action': 'Click', 'Page': 'search', 'Timestamp': '2021-03-31 16:35:39'}, {'Action': 'Search', 'Page': 'main', 'Timestamp': '2021-12-22 10:02:19'}, {'Action': 'Search', 'Page': 'about', 'Timestamp': '2019-08-24 05:11:40'}, {'Action': 'Add to Cart', 'Page': 'index', 'Timestamp': '2021-04-30 00:38:03'}, {'Action': 'Search', 'Page': 'privacy', 'Timestamp': '2021-06-21 16:23:49'}, {'Action': 'Search', 'Page': 'about', 'Timestamp': '2022-04-03 07:25:20'}, {'Action': 'Search', 'Page': 'author', 'Timestamp': '2022-11-07 02:24:31'}, {'Action': 'Search', 'Page': 'about', 'Timestamp': '2019-08-25 17:37:59'}, {'Action': 'Search', 'Page': 'post', 'Timestamp': '2020-12-18 01:36:34'}, {'Action': 'Search', 'Page': 'home', 'Timestamp': '2021-11-24 07:33:26'}, {'Action': 'Search', 'Page': 'login', 'Timestamp': '2020-11-15 07:21:21'}] |

| EngagementMetrics | {'Logins': 19, 'Frequency': 'Weekly'}
|

| Feedback | {'Rating': 1, 'Comment': 'I move baby go small big. Office institution six. Fact until hear technology right company seek.'}
|

| MarketingCommunication | [{'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Open

ed': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}]

Temporal Analysis: I looked at how the Churn rate chances over time to see if there are any reoccuring patterns

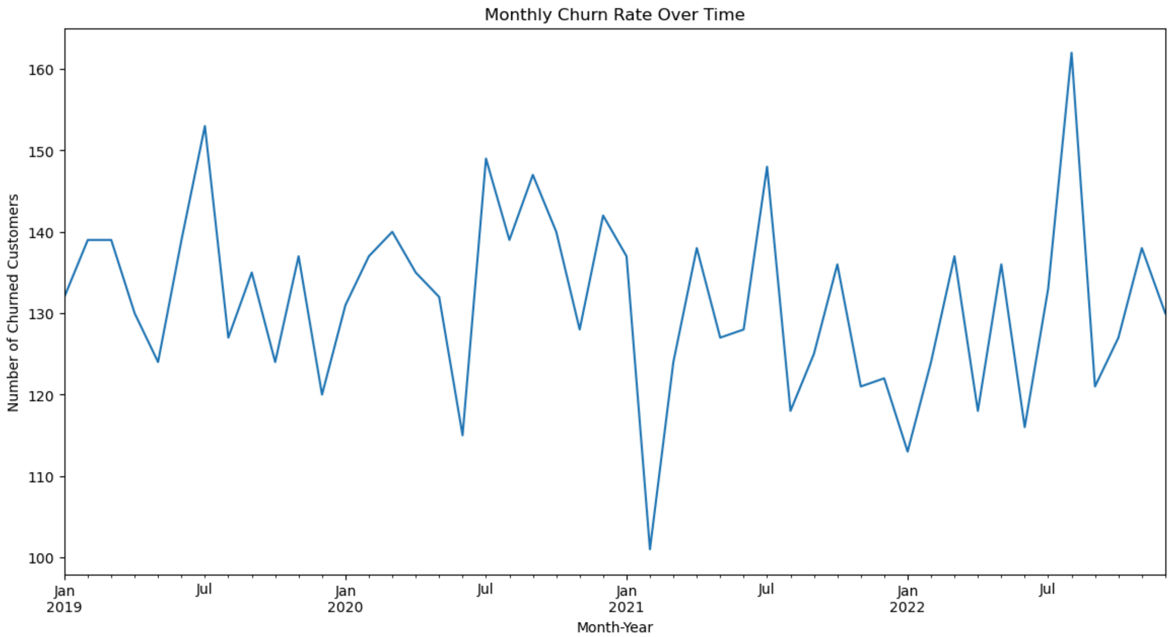
In [10]:

```
#Convert Timestamp to datatime format
df['Timestamp'] = pd.to_datetime(df['Timestamp'])

#Extract the month-year from the Timestamp
df['MonthYear'] = df['Timestamp'].dt.to_period('M')

# Group by MonthYear and calculate the churn rate
monthly_churn_rate = df.groupby('MonthYear')['ChurnLabel'].sum()

# Plot the Churn rate over time
plt.figure(figsize=(14,7))
monthly_churn_rate.plot()
plt.title('Monthly Churn Rate Over Time')
plt.ylabel('Number of Churned Customers')
plt.xlabel('Month-Year')
plt.show()
```



I need to see if there are any patterns between the customers feedback (Rating) and the ChurnLabel

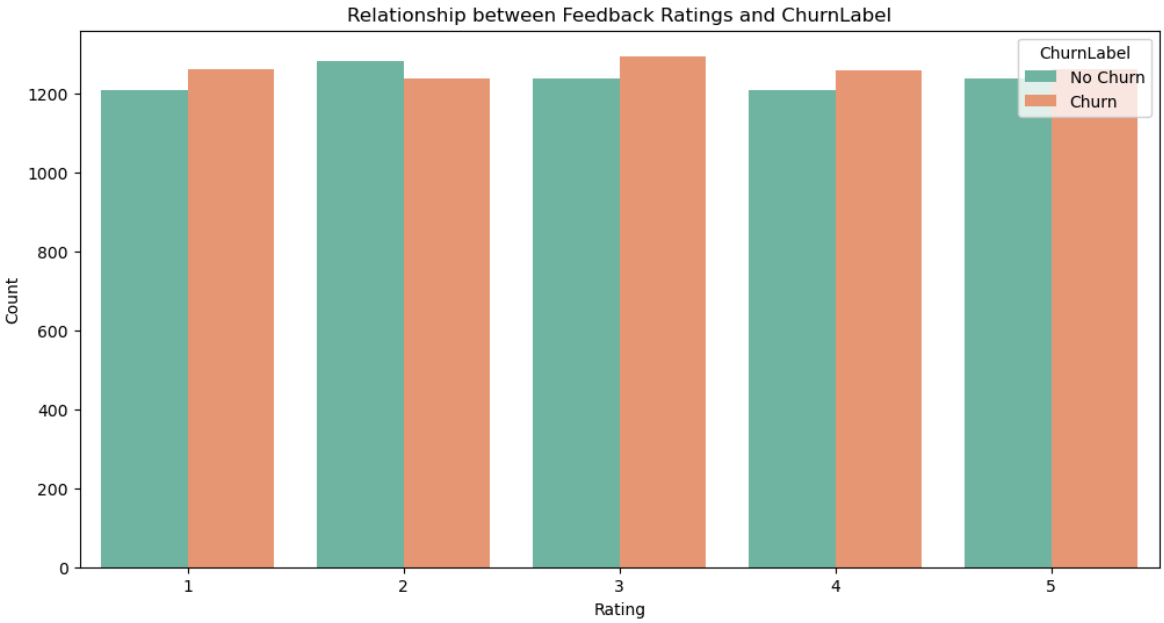
In [11]:

```
# Extract the ratings from the feedback column and create a new column for it
df['FeedbackRating'] = df['Feedback'].apply(lambda x: eval(x)['Rating'])  #eval

# Plot the relationship btwn feedback rating and Churnlabel
plt.figure(figsize=(12, 6))
sns.countplot(x='FeedbackRating', data=df, hue='ChurnLabel', palette='Set2')
```

```
plt.title('Relationship between Feedback Ratings and ChurnLabel')
plt.xlabel('Rating')
plt.ylabel('Count')
plt.legend(title = 'ChurnLabel', loc='upper right', labels=['No Churn', 'Churn'])
plt.show
```

Out[11]: <function matplotlib.pyplot.show(close=None, block=None)>



The visuals shows there isn't much indiction that feedback rating affects the Churn rate based on the positive or negative ratings

Data Preprocessing & Feature Engineering

Here I will be:

- create new fetaure that may have predictive power,
- convert categorical variable to numeric variables, using encoding techniques,
- scaling or normalize numeric variables where necessary,
- split data into train & test subsets
- remove irrelevant features.

I will start by converting the nested values from string formats to ist/ictionaries using the literal_eval function

In [12]:
```
#Listing out the nested columns
nested_columns = [
    'PurchaseHistory',
    'SubscriptionDetails',
    'ServiceInteractions',
    'PaymentHistory',
    'WebsiteUsage',
    'ClickstreamData',
    'EngagementMetrics',
    'Feedback',
    'MarketingCommunication'
]
```

```
#Applying the literal_eval function to lconvert to list/dict
for feature in nested_columns:
    df[feature] = df[feature].apply(literal_eval)
```

More features to be extarced from the dataset

In [13]:
```
# PurchaseHistory extracts all product purchased
df['PurchasProducts'] = df['PurchaseHistory'].apply(lambda x: '|'.join([i['Produ
df['PurchaseFrequency'] = df['PurchaseHistory'].apply(lambda x: sum([i['Frequenc
df['PurchaseValue'] = df['PurchaseHistory'].apply(lambda x: sum([i['Value'] for

#SubscriptionDetails
df['SubscriptionPlan'] = df['SubscriptionDetails'].apply(lambda x: x['Plan'])
df['SubscriptionStartDate'] = df['SubscriptionDetails'].apply(lambda x: x['Start
df['SubscriptionEndDate'] = df['SubscriptionDetails'].apply(lambda x: x['End_Dat
df['SubscriptionDuration'] = (pd.to_datetime(df['SubscriptionEndDate']) - pd.to_

#WebsiteUsage
df['WebsitePageViews'] = df['WebsiteUsage'].apply(lambda x: x['PageViews'])
df['WebsiteTimeSpent'] = df['WebsiteUsage'].apply(lambda x: x['TimeSpent(minutes

#EngagementMetrics
df['EngagementMetricsLogins'] = df['EngagementMetrics'].apply(lambda x: x['Login
df['EngagementMetricsFrequency'] = df['EngagementMetrics'].apply(lambda x: x['Fr

#Feedbacks
df['FeedbackRating'] = df['Feedback'].apply(lambda x: x['Rating'])
df['FeedbackComment'] = df['Feedback'].apply(lambda x: x['Comment'])

#MarketingCommunication
df['MarketingCommunicationNoOfEmails'] = df['MarketingCommunication'].apply(lamb
df['MarketingCommunicationOpenClickDiff'] =df['MarketingCommunication'].apply(
    lambda x: np.mean([
        (pd.to_datetime(i['Email_Clicked']) \
        - pd.to_datetime(i['Email_Opened'])).days for i in x
    ])
)
df['MarketingCommunicationSentOpenDiff'] = df['MarketingCommunication'].apply(
    lambda x: np.mean([
        (pd.to_datetime(i['Email_Opened']) \
         - pd.to_datetime(i['Email_Sent'])).days for i in x
    ])
)
```

Special Extraction from three columns:

- ServivceInteractions,
- PaymentHistory,
- ClickstreamData.

Check for the unique parameters/values in these columns.

In [14]:
```
# Get all unique ServiceInteraction 'Types'
service_interaction_types = df['ServiceInteractions'].apply(lambda x: list(set([
service_interaction_types = service_interaction_types.to_list()
```

```
unique_service_interaction_type = []
for i in service_interaction_types:
    unique_service_interaction_type.extend(i)
unique_service_interaction_type = list(set(unique_service_interaction_type))
print('All unique Service Interaction Types:', unique_service_interaction_type)

#Get all unique PaymentHistory 'Method'
payment_history_methods = df['PaymentHistory'].apply(lambda x: list(set([i['Meth
payment_history_methods = payment_history_methods.to_list()
unique_payment_history_methods = []
for i in payment_history_methods:
    unique_payment_history_methods.extend(i)
unique_payment_history_methods = list(set(unique_payment_history_methods))
print('All unique Payment History Methods:', unique_payment_history_methods)

#Get all unique ClickstreamData 'Action'
clickstream_data_actions = df['ClickstreamData'].apply(lambda x: list(set([i['Ac
clickstream_data_actions = clickstream_data_actions.to_list()
unique_clickstream_data_actions = []
for i in clickstream_data_actions:
    unique_clickstream_data_actions.extend(i)
unique_clickstream_data_actions = list(set(unique_clickstream_data_actions))
print('All unique Clickstream Data Actions:', unique_clickstream_data_actions)
```

```
All unique Service Interaction Types: ['Chat', 'Call', 'Email']
All unique Payment History Methods: ['PayPal', 'Bank Transfer', 'Credit Card']
All unique Clickstream Data Actions: ['Add to Cart', 'Search', 'Click']
```

FROM THESE THREE COLUMNS, I WILL FUTHER ENCODE MORE

In [15]:
```
#ServiceInteractions
for usit in unique_service_interaction_type:
    df[f'ServiceInteractions_{usit}'] = df['ServiceInteractions'].apply(lambda x

# Payment History
df['PaymentHistoryNoOfPayments'] = df['PaymentHistory'].apply(lambda x: sum(i['L
df['PaymentHistoryAvgNoOfLatePayments'] = df['PaymentHistory'].apply(lambda x: n

#ClickStreamData
for ucda in unique_clickstream_data_actions:
    df[f'ClickStreamData_{ucda}'] = df['ClickstreamData'].apply(lambda x: len([i
```

See a subset of the new data

In [16]:
```
df.head()
```

Out[16]:

	CustomerID	Name	Age	Gender	Location	Email	
0	1001	Mark Barrett	31	Male	Andrewfort	allison74@example.net	
1	1002	Jeremy Welch	66	Female	Millerhaven	fmiller@example.com	2
2	1003	Brandon Patel	36	Female	Lozanostad	jasonbrown@example.org	
3	1004	Tina Martin	62	Female	South Dustin	matthew62@example.net	0
4	1005	Christopher Rodriguez	68	Female	West James	shannonstrickland@example.org	

5 rows × 46 columns

See all columns

In [17]:
```
df.columns
```

Out[17]:
```
Index(['CustomerID', 'Name', 'Age', 'Gender', 'Location', 'Email', 'Phone',
       'Address', 'Segment', 'PurchaseHistory', 'SubscriptionDetails',
       'ServiceInteractions', 'PaymentHistory', 'WebsiteUsage',
       'ClickstreamData', 'EngagementMetrics', 'Feedback',
       'MarketingCommunication', 'NPS', 'ChurnLabel', 'Timestamp', 'MonthYear',
       'FeedbackRating', 'PurchasProducts', 'PurchaseFrequency',
       'PurchaseValue', 'SubscriptionPlan', 'SubscriptionStartDate',
       'SubscriptionEndDate', 'SubscriptionDuration', 'WebsitePageViews',
       'WebsiteTimeSpent', 'EngagementMetricsLogins',
       'EngagementMetricsFrequency', 'FeedbackComment',
       'MarketingCommunicationNoOfEmails',
       'MarketingCommunicationOpenClickDiff',
       'MarketingCommunicationSentOpenDiff', 'ServiceInteractions_Chat',
       'ServiceInteractions_Call', 'ServiceInteractions_Email',
       'PaymentHistoryNoOfPayments', 'PaymentHistoryAvgNoOfLatePayments',
       'ClickStreamData_Add to Cart', 'ClickStreamData_Search',
       'ClickStreamData_Click'],
      dtype='object')
```

Pick out some Columns

In [18]:
```
df_ = df[[
    'Age',
```

```
    'Gender',
    'NPS',
    'ChurnLabel',
    'PurchaseFrequency',
    'PurchaseValue',
    'SubscriptionPlan',
    'WebsitePageViews',
    'WebsiteTimeSpent',
    'EngagementMetricsLogins',
    'EngagementMetricsFrequency',
    'FeedbackRating',
    'MarketingCommunicationNoOfEmails',
    'MarketingCommunicationOpenClickDiff',
    'MarketingCommunicationSentOpenDiff',
    'ServiceInteractions_Email',
    'ServiceInteractions_Chat',
    'ServiceInteractions_Call',
    'PaymentHistoryNoOfPayments',
    'ClickStreamData_Search',
    'ClickStreamData_Click',
    'ClickStreamData_Add to Cart',
    'SubscriptionDuration'
]]

df_.head()
```

Out[18]:

	Age	Gender	NPS	ChurnLabel	PurchaseFrequency	PurchaseValue	SubscriptionPlan
0	31	Male	3	1	38	3994.72	Express
1	66	Female	6	0	4	2844.35	Pro
2	36	Female	3	0	14	1866.52	Essential
3	62	Female	1	1	28	1378.64	Smart
4	68	Female	3	0	39	2425.05	Basic

5 rows × 23 columns

Let's see the name of the columne we have now

In [19]:
```
df_.columns
```

Out[19]:
```
Index(['Age', 'Gender', 'NPS', 'ChurnLabel', 'PurchaseFrequency',
       'PurchaseValue', 'SubscriptionPlan', 'WebsitePageViews',
       'WebsiteTimeSpent', 'EngagementMetricsLogins',
       'EngagementMetricsFrequency', 'FeedbackRating',
       'MarketingCommunicationNoOfEmails',
       'MarketingCommunicationOpenClickDiff',
       'MarketingCommunicationSentOpenDiff', 'ServiceInteractions_Email',
       'ServiceInteractions_Chat', 'ServiceInteractions_Call',
       'PaymentHistoryNoOfPayments', 'ClickStreamData_Search',
       'ClickStreamData_Click', 'ClickStreamData_Add to Cart',
       'SubscriptionDuration'],
      dtype='object')
```

In [20]:
```
df_.loc[0] #this shows if all column values are in numeric numbers suitbale for
```

Out[20]:
```
Age                                    31
Gender                               Male
NPS                                     3
ChurnLabel                              1
PurchaseFrequency                      38
PurchaseValue                     3994.72
SubscriptionPlan                  Express
WebsitePageViews                       49
WebsiteTimeSpent                       15
EngagementMetricsLogins                19
EngagementMetricsFrequency         Weekly
FeedbackRating                          1
MarketingCommunicationNoOfEmails        8
MarketingCommunicationOpenClickDiff 319.0
MarketingCommunicationSentOpenDiff  818.0
ServiceInteractions_Email               1
ServiceInteractions_Chat                2
ServiceInteractions_Call                1
PaymentHistoryNoOfPayments             40
ClickStreamData_Search                 12
ClickStreamData_Click                   4
ClickStreamData_Add to Cart             8
SubscriptionDuration                  871
Name: 0, dtype: object
```

We need to encode all varibles in strings to numenical value to be able to use our ML prediction metrics. Also to check if we have exccess varibles to be encoded.

Now Checking for unique values in each column

In [21]:
```
print('Total datset length', len(df_))

df_[['Gender', 'SubscriptionPlan', 'EngagementMetricsFrequency']].nunique()
```

```
Total datset length 12483
```

Out[21]:
```
Gender                      2
SubscriptionPlan           20
EngagementMetricsFrequency  3
dtype: int64
```

Encoding the string parameters.

In [22]:
```
#Gender encoding
gender_map = {'Male': 0, 'Female': 1}

#Subcription encoding
unique_subscription_plans = df_['SubscriptionPlan'].unique()
subscription_plan_map = {unique_subscription_plans[i]: i for i in range(len(uniq

#EngagementMetricsFrequency
unique_engagement_frequency = df_['EngagementMetricsFrequency'].unique()
engagement_frequency_map = {unique_engagement_frequency[i]: i for i in range(len

#Encode
df_.loc[:, 'Gender'] = df_.loc[:, 'Gender'].map(gender_map)
```

```
df_.loc[:, 'SubscriptionPlan'] = df_.loc[:, 'SubscriptionPlan'].map(subscription
df_.loc[:, 'EngagementMetricsFrequency'] = df_.loc[:, 'EngagementMetricsFrequenc
```

In [23]:
```
df_.loc[0]
```

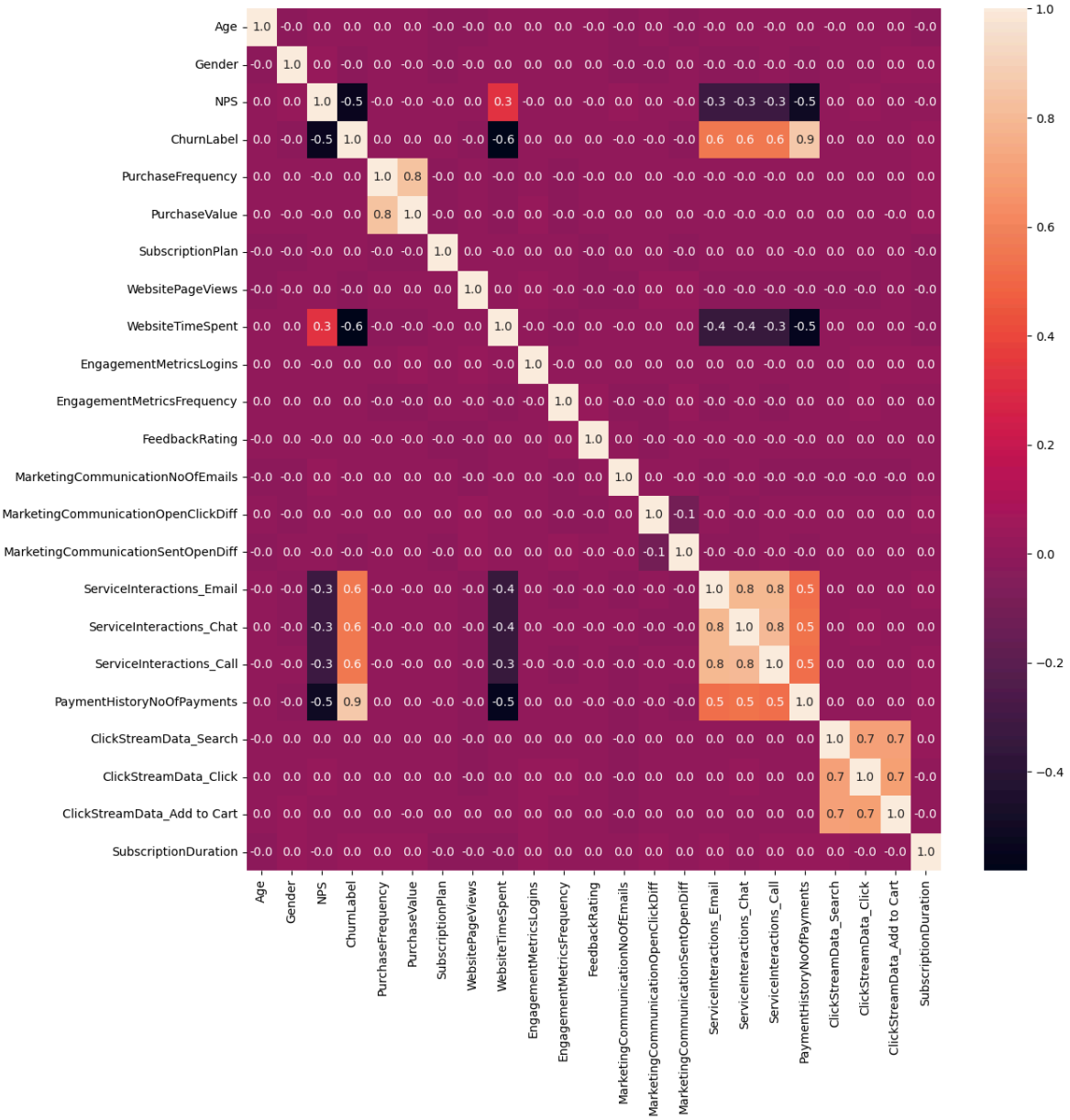
Out[23]:

Age	31
Gender	0
NPS	3
ChurnLabel	1
PurchaseFrequency	38
PurchaseValue	3994.72
SubscriptionPlan	0
WebsitePageViews	49
WebsiteTimeSpent	15
EngagementMetricsLogins	19
EngagementMetricsFrequency	0
FeedbackRating	1
MarketingCommunicationNoOfEmails	8
MarketingCommunicationOpenClickDiff	319.0
MarketingCommunicationSentOpenDiff	818.0
ServiceInteractions_Email	1
ServiceInteractions_Chat	2
ServiceInteractions_Call	1
PaymentHistoryNoOfPayments	40
ClickStreamData_Search	12
ClickStreamData_Click	4
ClickStreamData_Add to Cart	8
SubscriptionDuration	871

Name: 0, dtype: object

Plot correlation Matrix

In [24]:
```
df_corr = df_.corr()
fig, ax = plt.subplots(figsize=(13, 13))
sns.heatmap(df_corr, annot=True, fmt='.1f', ax=ax)
plt.show()
```



Split data into train, test and validation sets.

In [25]:
```
X = df_.drop(columns=['ChurnLabel'])
y = df_['ChurnLabel']

X_train, X_other, y_train, y_other = train_test_split(X, y, train_size=0.8, rand
X_test, X_val, y_test, y_val = train_test_split(X_other, y_other, test_size=0.3,
```

In [29]:
```
ss= StandardScaler()
X_train = ss.fit_transform(X_train)
X_val = ss.transform(X_val)
X_test = ss.transform(X_test)
```

Modelling

Two different models for modelling:

- LogisticRegression,
- DecisionTreeClassifier.

Metrics:

- Accuracy score,
- Precision score,
- Recall score,
- F1 score.

In [34]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_sc

def evaluate(X, y, model, subset=''):
    y_pred = model.predict(X)

    print(f'{subset} Accuracy Score: {accuracy_score(y, y_pred)}')
    print(f'{subset} Precision Score: {precision_score(y, y_pred)}')
    print(f'{subset} Recall Score: {recall_score(y, y_pred)}')
    print(f'{subset} F1 Score: {f1_score(y, y_pred)}')
```

Modelling with Logoistic Regression

In [38]:
```
#Build the model
lr = LogisticRegression()
lr.fit(X_train, y_train)



# Evaluate the model on train and validation subsets
evaluate(X_train, y_train, lr, 'Train')
evaluate(X_val, y_val, lr, 'Validation')
```

```
Train Accuracy Score: 0.9709593430803124
Train Precision Score: 0.9767071471232331
Train Recall Score: 0.965938176806458
Train F1 Score: 0.9712928133042962
Validation Accuracy Score: 0.968
Validation Precision Score: 0.9643835616438357
Validation Recall Score: 0.9696969696969697
Validation F1 Score: 0.967032967032967
```

Modelling with Decision Tree

In [46]:
```
    #Build the model
    dt = DecisionTreeClassifier(max_depth=5)
    dt.fit(X_train, y_train)
/

    # Evaluate on the train and validation subsets
    evaluate(X_train, y_train, dt, subset='Train')
    evaluate(X_val, y_val, dt, subset='Validation')
```

```
Train Accuracy Score: 0.9768676146605247
Train Precision Score: 0.9775413711583925
Train Recall Score: 0.9769639692852924
Train F1 Score: 0.9772525849335303
Validation Accuracy Score: 0.968
Validation Precision Score: 0.9643835616438357
Validation Recall Score: 0.9696969696969697
Validation F1 Score: 0.967032967032967
```

Evaluation on the Test Set

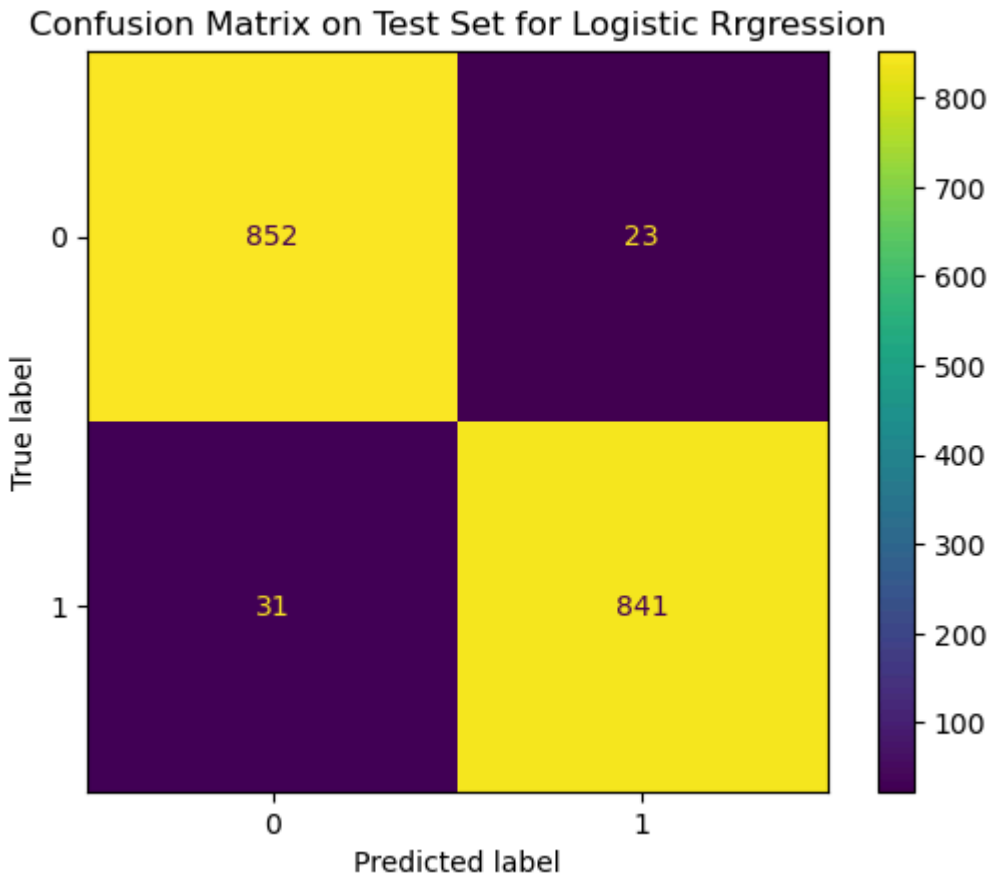
In [47]:
```
evaluate(X_test, y_test, lr, 'LogisticsRegression Test')
evaluate(X_test, y_test, dt, 'DecisionTreeClassifier Test')
```

```
LogisticsRegression Test Accuracy Score: 0.9690898683457355
LogisticsRegression Test Precision Score: 0.9733796296296297
LogisticsRegression Test Recall Score: 0.9644495412844036
LogisticsRegression Test F1 Score: 0.9688940092165899
DecisionTreeClassifier Test Accuracy Score: 0.9725243274184316
DecisionTreeClassifier Test Precision Score: 0.9735632183908046
DecisionTreeClassifier Test Recall Score: 0.9713302752293578
DecisionTreeClassifier Test F1 Score: 0.9724454649827784
```

Plotting Confusion Matrix

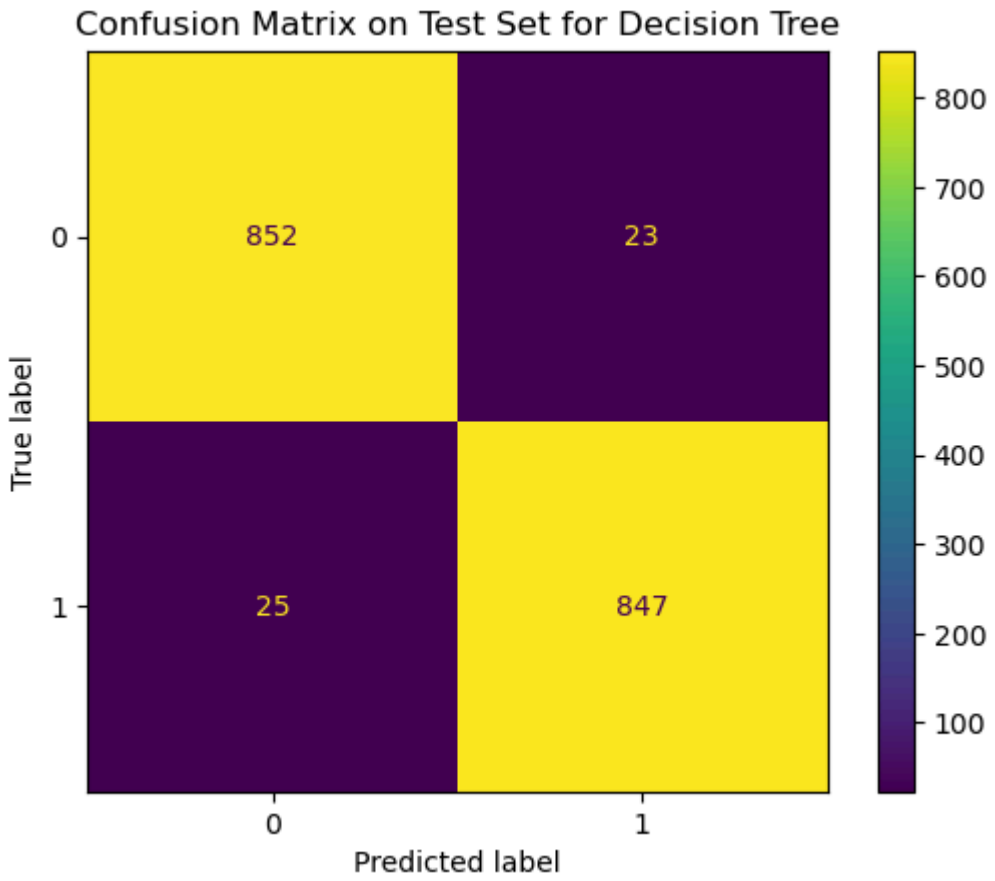
In [49]:
```
lr_y_pred = lr.predict(X_test)
logistic_regression_confusion_matrix = confusion_matrix(y_test, lr_y_pred)

display = ConfusionMatrixDisplay(confusion_matrix=logistic_regression_confusion_
display.plot()
plt.title('Confusion Matrix on Test Set for Logistic Rrgression')
plt.show()
```



In [51]:
```
dt_y_pred = dt.predict(X_test)
decision_tree_confusion_matrix = confusion_matrix(y_test, dt_y_pred)

display = ConfusionMatrixDisplay(confusion_matrix=decision_tree_confusion_matrix
display.plot()
plt.title('Confusion Matrix on Test Set for Decision Tree')
plt.show()
```



Conclusion

The most important Features the affects and correlates Churn Label are:

- The number of service interaction the customer had through Call Email, Chat,
- The number of times the customers has made last payments,
- The time customer spents on the company websites,
- The NET Promoter Score (NPS).

In [ ]: