Predicting Customer Churn: Identifying Customers that are Susceptible to Churn

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from ast import literal_eval
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.tree import DecisionTreeClassifier
from tqdm import tqdm

import warnings
warnings.filterwarnings("ignore")
```

In [3]:
```
df= pd.read_excel("Dataset.xlsx")
```

In [4]:
```
df.head()
```

Out[4]:

	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	231-!
2	1003	Brandon Patel	36	Female	Lozanostad	jasonbrown@example.org	
3	1004	Tina Martin	62	Female	South Dustin	matthew62@example.net	050.0
4	1005	Christopher Rodriguez	68	Female	West James	shannonstrickland@example.org	

5 rows × 21 columns

In [5]:

```
df.isnull().any()
```

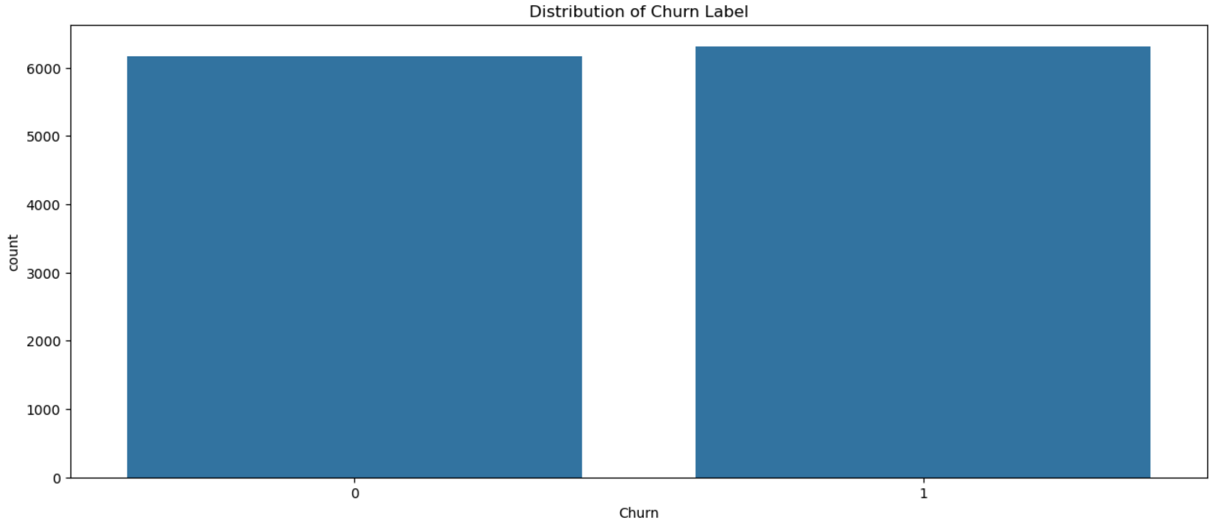
Out[5]:
```
CustomerID               False
Name                     False
Age                      False
Gender                   False
Location                 False
Email                    False
Phone                    False
Address                  False
Segment                  False
PurchaseHistory          False
SubscriptionDetails      False
ServiceInteractions      False
PaymentHistory           False
WebsiteUsage             False
ClickstreamData          False
EngagementMetrics        False
Feedback                 False
MarketingCommunication   False
NPS                      False
ChurnLabel               False
Timestamp                False
dtype: bool
```

In [6]:
```
#Statistical Over view of numerical data
df.describe()
```

Out[6]:

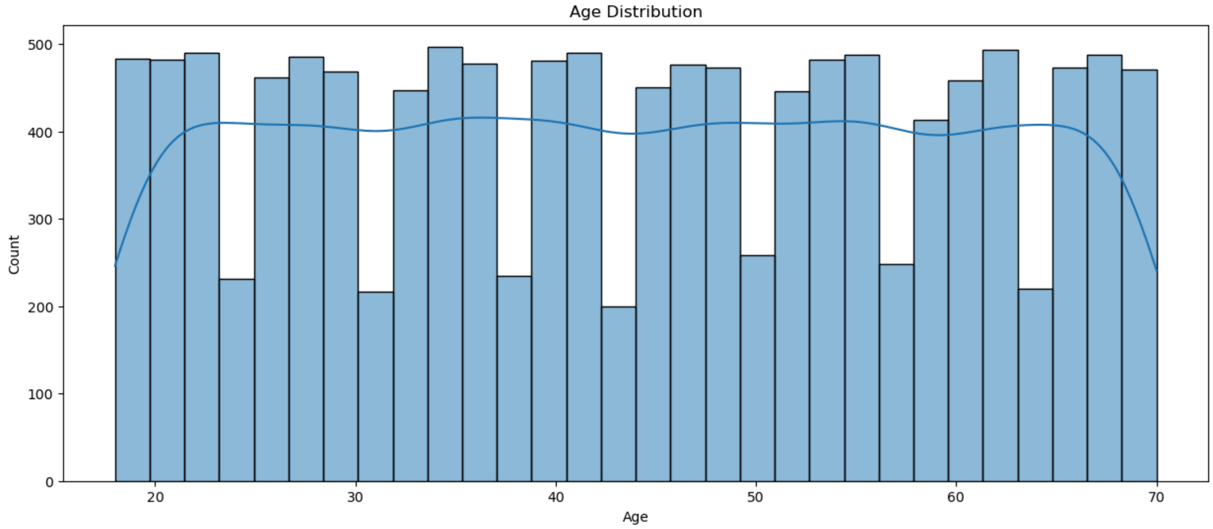
	CustomerID	Age	NPS	ChurnLabel
count	12483.00000	12483.000000	12483.000000	12483.000000
mean	7242.00000	43.930065	2.973884	0.505808
std	3603.67604	15.341521	2.644623	0.499986
min	1001.00000	18.000000	0.000000	0.000000
25%	4121.50000	31.000000	1.000000	0.000000
50%	7242.00000	44.000000	2.000000	1.000000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

In [7]:
```
fig, ax=plt.subplots( figsize=(15, 6))

#Distribution of the target variable
sns.countplot(x= "ChurnLabel", data= df, ax=ax)
plt.title("Distribution of Churn Label")
plt.xlabel("Churn")

plt.show();
```

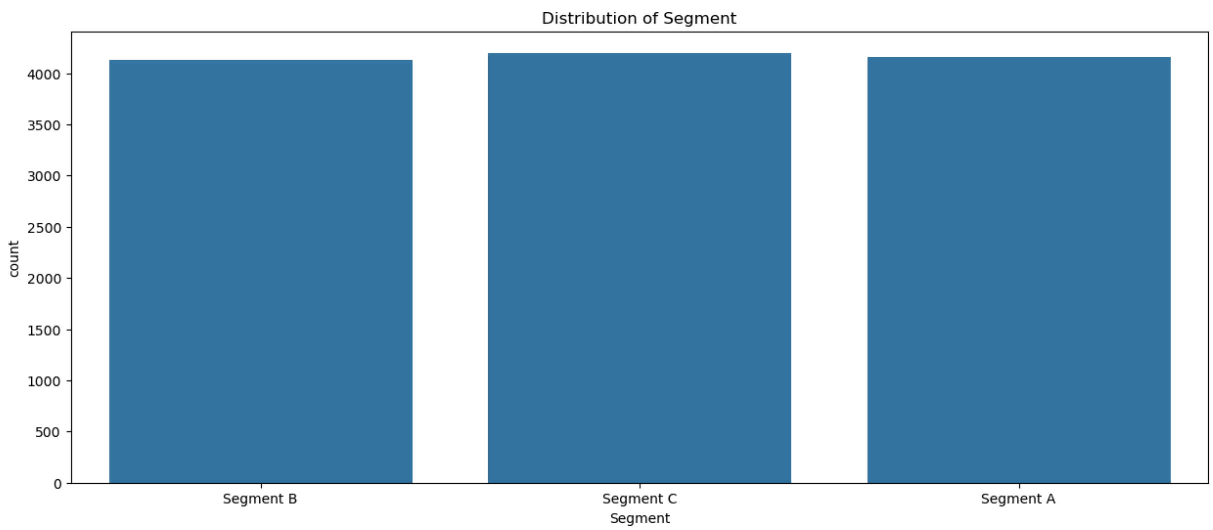


In [8]:
```
# Distribution of Genders
fig, ax=plt.subplots( figsize=(15,6))
sns.countplot(x= "Gender", data=df, ax=ax)
plt.title("Distribution of Gender");
plt.show()
```

In [9]:
```
fig, ax=plt.subplots( figsize=(15,6))
sns.histplot(df["Age"], bins =30, ax=ax, kde=True)
plt.title("Age Distribution");
```



In [10]:
```
fig, ax=plt.subplots( figsize=(15,6))
sns.countplot(x= "Segment", data=df, ax=ax)
plt.title("Distribution of Segment");
plt.show()
```

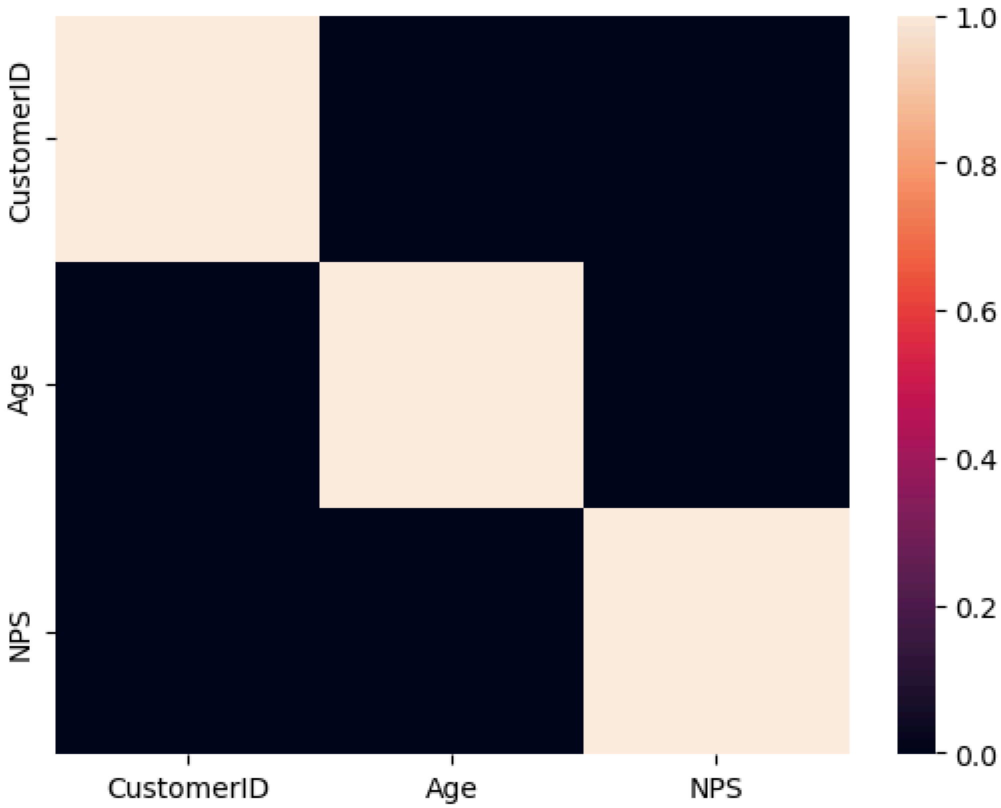


In [11]:
```
#correlation and HeatMap
correlation =df.select_dtypes("number").drop(columns="ChurnLabel").corr()
correlation
```

Out[11]:

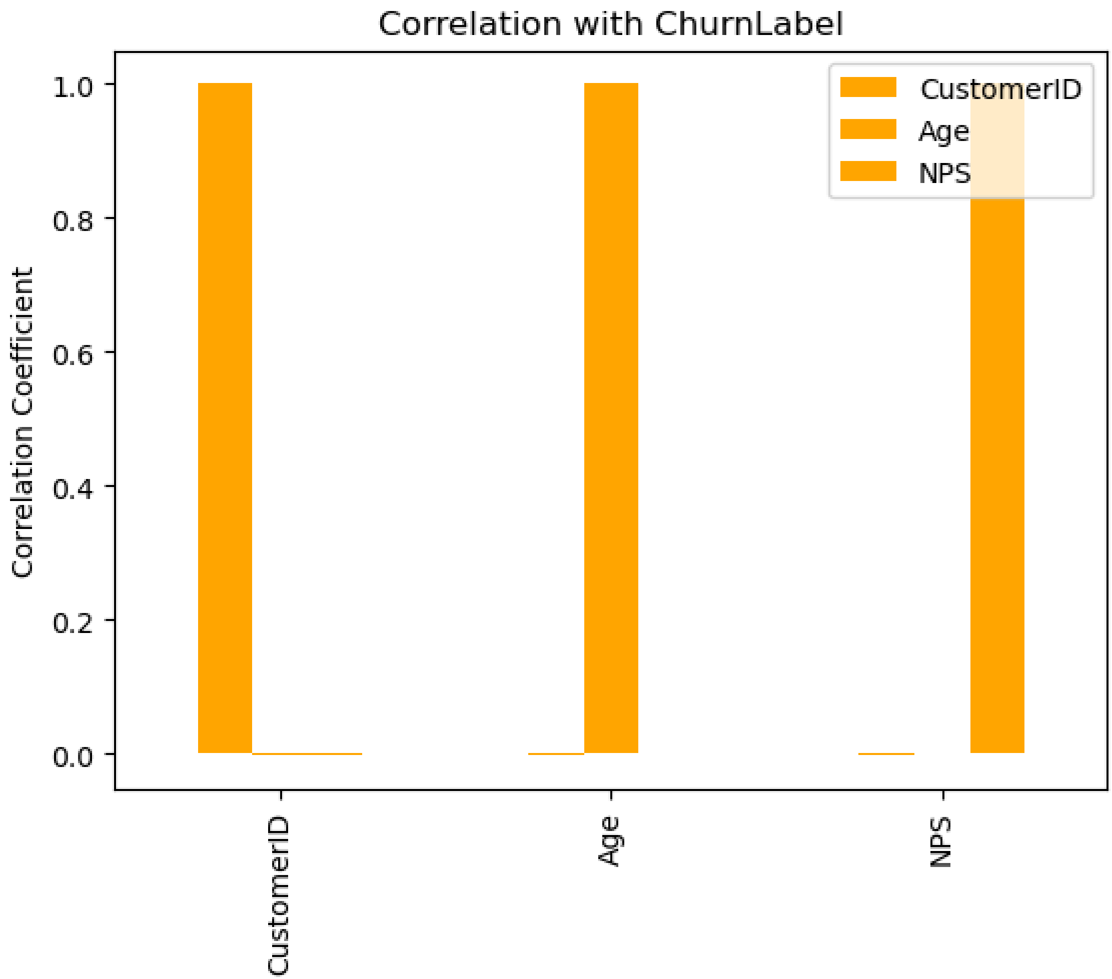
	CustomerID	Age	NPS
CustomerID	1.000000	-0.002670	-0.002513
Age	-0.002670	1.000000	0.000006
NPS	-0.002513	0.000006	1.000000

In [12]:
```
sns.heatmap(correlation);
```

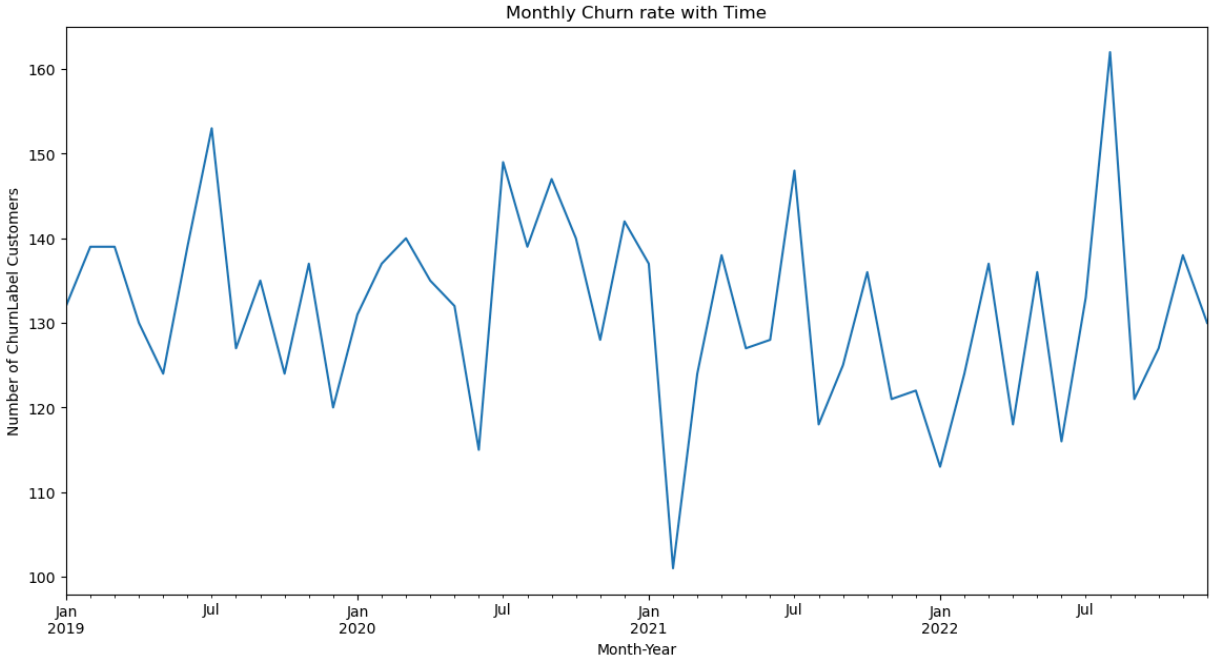


In [13]:
```
#Plot Correlation chart
plt.figure(figsize=(10, 6))
correlation.plot(kind='bar', color="orange")
plt.title("Correlation with ChurnLabel")
plt.ylabel("Correlation Coefficient")
plt.show()
```

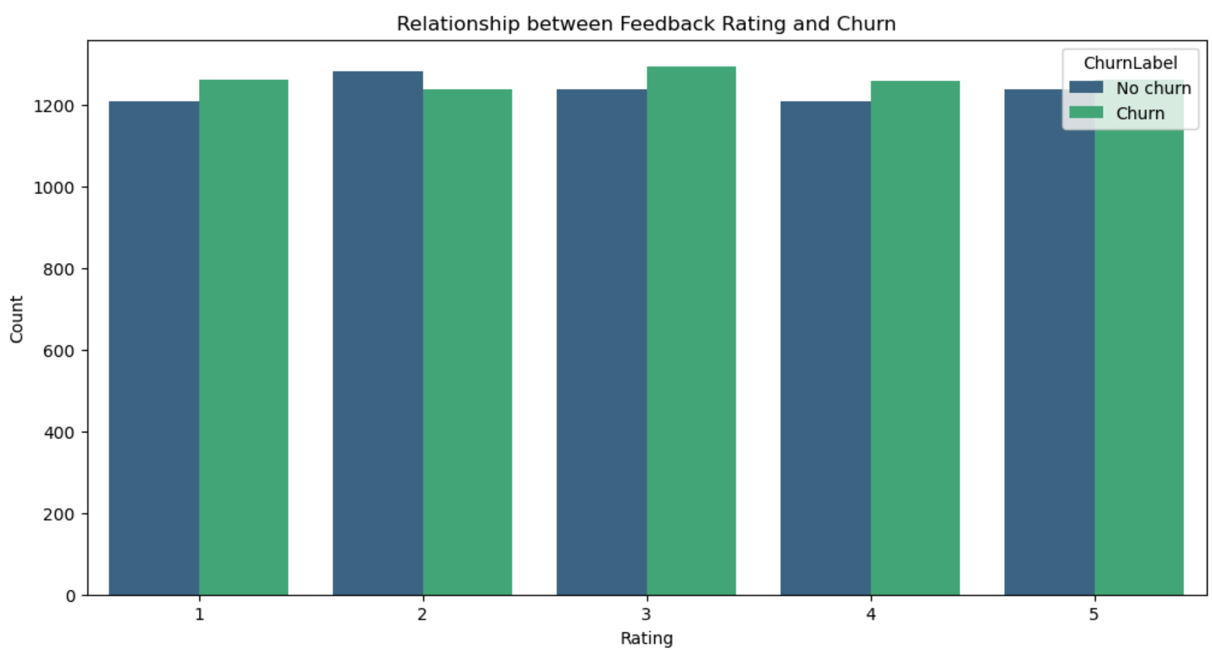
```
<Figure size 1000x600 with 0 Axes>
```



In [14]:
```
# Time zone
df["Timestamp"]= pd.to_datetime(df["Timestamp"])

#extract month and year
df["MonthYear"]= df["Timestamp"].dt.to_period("M")

monthly_churn_rate= df.groupby("MonthYear")["ChurnLabel"].sum()

#plot
plt.figure(figsize=(14, 7))
monthly_churn_rate.plot()
plt.title("Monthly Churn rate with Time")
plt.ylabel("Number of ChurnLabel Customers")
plt.xlabel("Month-Year")
plt.show();
```



In [15]:
```
# Customer rating effect on Churn Label
df["feedbackrating"]= df["Feedback"].apply(lambda x: eval(x)["Rating"])

#Plot the relationship
plt.figure(figsize=(12, 6))
sns.countplot(x="feedbackrating", data=df, hue="ChurnLabel", palette="viridis")
plt.title("Relationship between Feedback Rating and Churn")
plt.xlabel("Rating")
plt.ylabel("Count")
plt.legend(title= "ChurnLabel", loc="upper right", labels = ["No churn", "Churn"])
plt.show();
```



From the visualization there doesnt seems to be any indication that the "Feedback" rating affect "Churn Label"

In [17]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12483 entries, 0 to 12482
Data columns (total 23 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              12483 non-null  int64
 1   Name                    12483 non-null  object
 2   Age                     12483 non-null  int64
 3   Gender                  12483 non-null  object
 4   Location                12483 non-null  object
 5   Email                   12483 non-null  object
 6   Phone                   12483 non-null  object
 7   Address                 12483 non-null  object
 8   Segment                 12483 non-null  object
 9   PurchaseHistory         12483 non-null  object
 10  SubscriptionDetails     12483 non-null  object
 11  ServiceInteractions     12483 non-null  object
 12  PaymentHistory          12483 non-null  object
 13  WebsiteUsage            12483 non-null  object
 14  ClickstreamData         12483 non-null  object
 15  EngagementMetrics       12483 non-null  object
 16  Feedback                12483 non-null  object
 17  MarketingCommunication  12483 non-null  object
 18  NPS                     12483 non-null  int64
 19  ChurnLabel              12483 non-null  int64
 20  Timestamp               12483 non-null  datetime64[ns]
 21  MonthYear               12483 non-null  period[M]
 22  feedbackrating          12483 non-null  int64
dtypes: datetime64[ns](1), int64(5), object(16), period[M](1)
memory usage: 2.2+ MB
```

In [18]:
```
#create nested columns
nested_columns=[
    "PurchaseHistory",
    "SubscriptionDetails",
    "ServiceInteractions",
    "PaymentHistory",
    "WebsiteUsage",
    "ClickstreamData",
    "EngagementMetrics",
    "Feedback",
    "MarketingCommunication"
]
w1,  w2 = 25, 100
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, 'Va
lue': 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': 'Mortise
rs', 'Frequency': 2, 'Value': 614.08}, {'Product': 'Rulers', 'Frequency': 6, 'Valu
e': 221.68}, {'Product': 'Invitations', 'Frequency': 3, 'Value': 660.04}]|

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

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

|PaymentHistory |[{'Method': 'Credit Card', 'Late_Payments': 5}, {'Metho
d': 'PayPal', 'Late_Payments': 11}, {'Method': 'Bank Transfer', 'Late_Payments': 2
4}]|

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

|ClickstreamData |[{'Action': 'Add to Cart', 'Page': 'register', 'Timestam
p': '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 10:17:4
4'}, {'Action': 'Add to Cart', 'Page': 'author', 'Timestamp': '2022-07-14 03:40:5
3'}, {'Action': 'Search', 'Page': 'main', 'Timestamp': '2019-01-13 08:39:42'}, {'Act
ion': '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': 'Cli
ck', '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': 'pr
ivacy', 'Timestamp': '2021-06-13 06:18:41'}, {'Action': 'Add to Cart', 'Page': 'sear
ch', 'Timestamp': '2022-03-28 16:25:35'}, {'Action': 'Search', 'Page': 'homepage',
'Timestamp': '2019-09-26 12:27:42'}, {'Action': 'Click', 'Page': 'search', 'Timestam
p': '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 00:38:0
3'}, {'Action': 'Search', 'Page': 'privacy', 'Timestamp': '2021-06-21 16:23:49'},
{'Action': 'Search', 'Page': 'about', 'Timestamp': '2022-04-03 07:25:20'}, {'Actio
n': 'Search', 'Page': 'author', 'Timestamp': '2022-11-07 02:24:31'}, {'Action': 'Sea
rch', 'Page': 'about', 'Timestamp': '2019-08-25 17:37:59'}, {'Action': 'Search', 'Pa
ge': 'post', 'Timestamp': '2020-12-18 01:36:34'}, {'Action': 'Search', 'Page': 'hom
e', 'Timestamp': '2021-11-24 07:33:26'}, {'Action': 'Search', 'Page': 'login', 'Time
stamp': '2020-11-15 07:21:21'}]|

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

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

|MarketingCommunication |[{'Email_Sent': '2019-10-17', 'Email_Opened': '2022-01-1
2', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opened': '20
22-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Email_Opene
d': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-17', 'Emai
l_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '2019-10-1

```
7', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sent': '20
19-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Email_Sen
t': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-27'}, {'Em
ail_Sent': '2019-10-17', 'Email_Opened': '2022-01-12', 'Email_Clicked': '2022-11-2
7'}]|
```

convert nested_colums to list from str

In [20]:
```
nested_columns=[
    "PurchaseHistory",
    "SubscriptionDetails",
    "ServiceInteractions",
    "PaymentHistory",
    "WebsiteUsage",
    "ClickstreamData",
    "EngagementMetrics",
    "Feedback",
    "MarketingCommunication"
]
for feature in nested_columns:
    df[feature]=df[feature].apply(literal_eval)
```

In [21]:
```
#extraction of features
#PurchaseHistory
df["PurchaseProduct"]=df["PurchaseHistory"].apply(lambda x: '|'.join([i["Product"]f
df["PurchaseFrequency"]= df["PurchaseHistory"].apply(lambda x:sum(i["Frequency"] fo
df["PurchaseValue"]=df["PurchaseHistory"].apply(lambda x:sum(i["Value"] for i in x)

#SubscriptionDetails
df["SubscriptionPlan"]=df["SubscriptionDetails"].apply(lambda x: x["Plan"])
df["SubscriptionStartDate"]=df["SubscriptionDetails"].apply(lambda x: x["Start_Date
df["SubscriptionEndDate"]=df["SubscriptionDetails"].apply(lambda x: x["End_Date"])
df["SubscriptionDuration"]=(pd.to_datetime(df["SubscriptionEndDate"]) - pd.to_datet

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

#Engagemntmetrics
df["EngagementMetricsLogins"]=df["EngagementMetrics"].apply(lambda x: x["Logins"])
df["EngagementMetricsFrequency"]= df["EngagementMetrics"].apply(lambda x: x["Freque

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

#marketing Communication
df["MarketingCommunicationNoofEmails"] = df["MarketingCommunication"].apply(lambda

df["marketingCommunicationOpenClickDiff"] = df["MarketingCommunication"].apply(
    lambda x: np.mean([
        (pd.to_datetime(i["Email_Clicked"]) - pd.to_datetime(i["Email_Opened"])).da
        for i in x if i.get("Email_Clicked") and i.get("Email_Opened")
    ]) if x else np.nan
```

```
)

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 if i.get("Email_Opened") and i.get("Email_Sent")
    ]) if x else np.nan
)
```

In [22]:
```
# Extraction from three columns

ServiceInteractionTypes= df["ServiceInteractions"].apply(lambda x: list(set([i["Typ
ServiceInteractionTypes = ServiceInteractionTypes.to_list()
unique_service_interaction_types= []
for i in ServiceInteractionTypes:
    unique_service_interaction_types.extend(i)
unique_service_interaction_types= list(set(unique_service_interaction_types))
print ("All Unique Service Interaction Types:", unique_service_interaction_types)

#Get all payment method
payment_history_method =df["PaymentHistory"].apply(lambda x: list(set([i["Method"]f
payment_history_method= payment_history_method.to_list()
unique_payment_history_method =[]
for i in payment_history_method:
    unique_payment_history_method.extend(i)
unique_payment_history_method= list(set(unique_payment_history_method))
print("All unique Payment History Method:", unique_payment_history_method)

# Unique ClickStreamData "Action"
clickstream_data_actions = df["ClickstreamData"].apply(lambda x: list(set([i["Actio
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 Action:", unique_clickstream_data_actions)
```

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

In [23]:
```
#ServiceInteractions
for usit in unique_service_interaction_types:
    df[f"ServiceInteractions_{usit}"]= df["ServiceInteractions"].apply(lambda x: le

#PaymentHistory
df["PaymentHistoryOfNoLatePayment"]= df["PaymentHistory"].apply(lambda x: sum(i["La
df["PaymentHistoryAvgNoOfLatePayment"]= df["PaymentHistory"].apply(lambda x: np.mea

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

In [24]:
```
df.columns
```

```
Out[24]: Index(['CustomerID', 'Name', 'Age', 'Gender', 'Location', 'Email', 'Phone',
               'Address', 'Segment', 'PurchaseHistory', 'SubscriptionDetails',
               'ServiceInteractions', 'PaymentHistory', 'WebsiteUsage',
               'ClickstreamData', 'EngagementMetrics', 'Feedback',
               'MarketingCommunication', 'NPS', 'ChurnLabel', 'Timestamp', 'MonthYear',
               'feedbackrating', 'PurchaseProduct', 'PurchaseFrequency',
               'PurchaseValue', 'SubscriptionPlan', 'SubscriptionStartDate',
               'SubscriptionEndDate', 'SubscriptionDuration', 'WebsitePageViews',
               'WebsiteTimeSpent', 'EngagementMetricsLogins',
               'EngagementMetricsFrequency', 'FeedbackRating', 'FeedbackComment',
               'MarketingCommunicationNoofEmails',
               'marketingCommunicationOpenClickDiff',
               'MarketingCommunicationSentOpenDiff', 'ServiceInteractions_Call',
               'ServiceInteractions_Chat', 'ServiceInteractions_Email',
               'PaymentHistoryOfNoLatePayment', 'PaymentHistoryAvgNoOfLatePayment',
               'ClickstreamData_Add to Cart', 'ClickstreamData_Search',
               'ClickstreamData_Click'],
              dtype='object')
```

```
In [25]: df_= df[[
             "Age",
             "Gender",
             "NPS",
             "ChurnLabel",
             "PurchaseFrequency",
             "PurchaseValue",
             "SubscriptionPlan",
             "WebsitePageViews",
             "WebsiteTimeSpent",
             "EngagementMetricsLogins",
             "EngagementMetricsFrequency",
             "FeedbackRating",
             "MarketingCommunicationNoofEmails",
             "marketingCommunicationOpenClickDiff",
             "ServiceInteractions_Call",
             "ServiceInteractions_Email",
             "ServiceInteractions_Chat",
             "PaymentHistoryOfNoLatePayment",
             "ClickstreamData_Click",
             "ClickstreamData_Add to Cart",
             "ClickstreamData_Search",
             "SubscriptionDuration"

         ]]
         df_.head()
```

Out[25]:

	Age	Gender	NPS	ChurnLabel	PurchaseFrequency	PurchaseValue	SubscriptionPlan	W
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 × 22 columns

In [26]:
```
df_.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 12483 entries, 0 to 12482
Data columns (total 22 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Age                                12483 non-null  int64  
 1   Gender                             12483 non-null  object 
 2   NPS                                12483 non-null  int64  
 3   ChurnLabel                         12483 non-null  int64  
 4   PurchaseFrequency                  12483 non-null  int64  
 5   PurchaseValue                      12483 non-null  float64
 6   SubscriptionPlan                   12483 non-null  object 
 7   WebsitePageViews                   12483 non-null  int64  
 8   WebsiteTimeSpent                   12483 non-null  int64  
 9   EngagementMetricsLogins            12483 non-null  int64  
 10  EngagementMetricsFrequency         12483 non-null  object 
 11  FeedbackRating                     12483 non-null  int64  
 12  MarketingCommunicationNoofEmails   12483 non-null  int64  
 13  marketingCommunicationOpenClickDiff 12483 non-null  float64
 14  ServiceInteractions_Call           12483 non-null  int64  
 15  ServiceInteractions_Email          12483 non-null  int64  
 16  ServiceInteractions_Chat           12483 non-null  int64  
 17  PaymentHistoryOfNoLatePayment      12483 non-null  int64  
 18  ClickstreamData_Click              12483 non-null  int64  
 19  ClickstreamData_Add to Cart        12483 non-null  int64  
 20  ClickstreamData_Search             12483 non-null  int64  
 21  SubscriptionDuration               12483 non-null  int64  
dtypes: float64(2), int64(17), object(3)
memory usage: 2.1+ MB
```

In [27]:
```
#Number of Unique Values

print("Total Dataset Length:", len(df_))
df_[["Gender", "SubscriptionPlan", "EngagementMetricsFrequency"]].nunique()
```

```
Total Dataset Length: 12483
```

Out[27]:
```
Gender                        2
SubscriptionPlan             20
EngagementMetricsFrequency    3
dtype: int64
```

In [28]:
```
#Encoding string into Parameters

#gender
gender_map= {"male": 0, "female": 1}

#subcriptionPlan encoding
unique_subscription_plans = df_["SubscriptionPlan"].unique()
subscription_plan_map= {unique_subscription_plans[i]: i for i in range(len(unique_s

#EngagementMetrics Frequency encoding
unique_engagement_frequency = df_["EngagementMetricsFrequency"].unique()
engagement_frequency_map = {unique_engagement_frequency[i]: i for i in range(len(un

#Encode
df_.loc[:, "Gender"]= df_.loc[:, "Gender"].map(gender_map)
df_.loc[:, "SubscriptionPlan"]= df_.loc[:, "SubscriptionPlan"].map(subscription_pla
df_.loc[:, "EngagementMetricsFrequency"]= df_.loc[:, "EngagementMetricsFrequency"].
```

In [29]:
```
df_.loc[0]
```

Out[29]:
```
Age                                        31
Gender                                    NaN
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
ServiceInteractions_Call                    1
ServiceInteractions_Email                   1
ServiceInteractions_Chat                    2
PaymentHistoryOfNoLatePayment              40
ClickstreamData_Click                       4
ClickstreamData_Add to Cart                 8
ClickstreamData_Search                     12
SubscriptionDuration                      871
Name: 0, dtype: object
```

In [30]:
```
df_.head()
```

Out[30]:

	Age	Gender	NPS	ChurnLabel	PurchaseFrequency	PurchaseValue	SubscriptionPlan	W
0	31	NaN	3	1	38	3994.72	0	
1	66	NaN	6	0	4	2844.35	1	
2	36	NaN	3	0	14	1866.52	2	
3	62	NaN	1	1	28	1378.64	3	
4	68	NaN	3	0	39	2425.05	4	

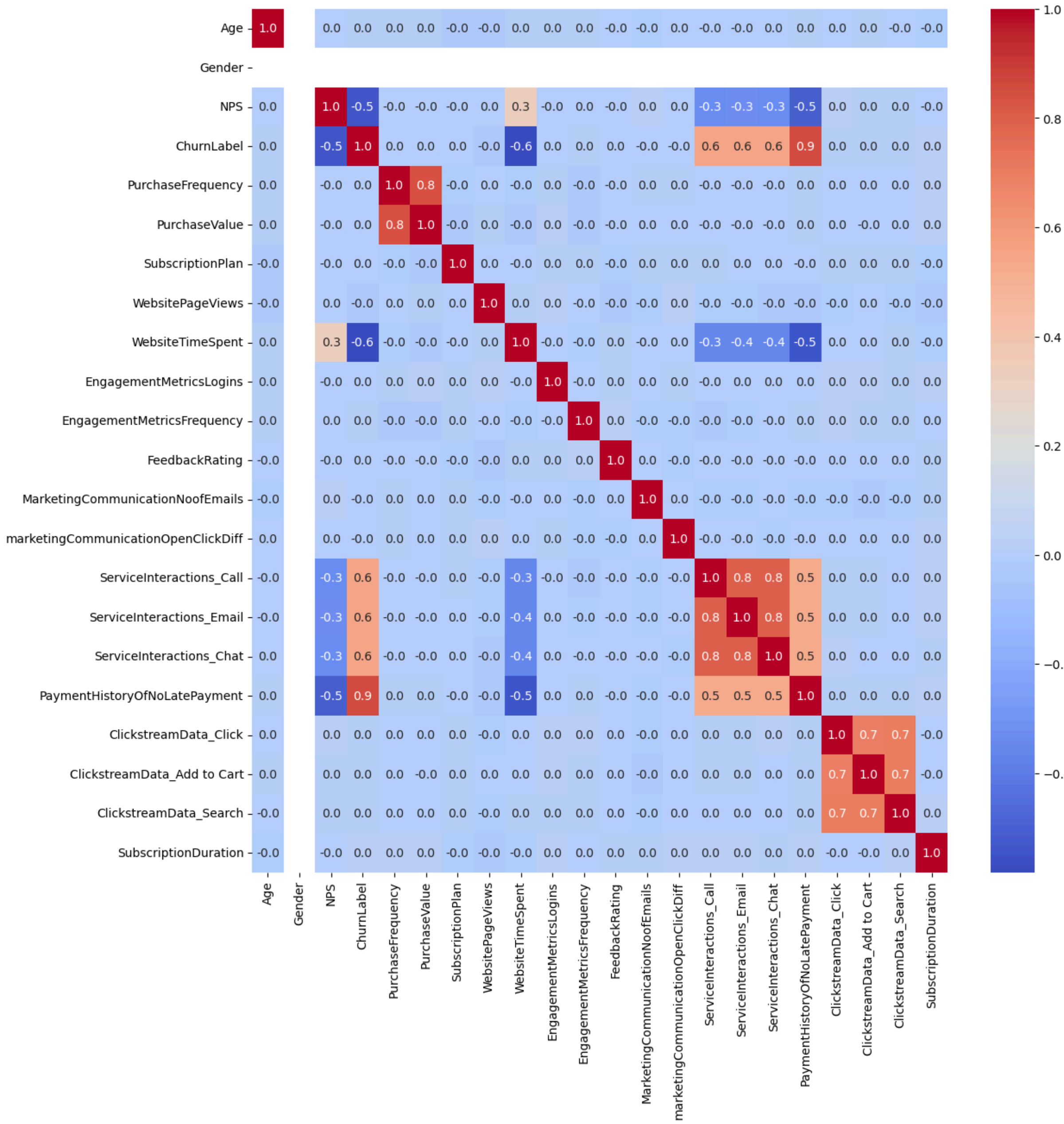
5 rows × 22 columns

In [31]:

```
# fill all nan to 0 because logistic regression will no evaluate nan value
df_["Gender"].fillna(0, inplace=True)
```

In [32]:

```
#Plot correlation matrix

df_corr = df_.corr()
fig, ax = plt.subplots(figsize=(13, 13))
sns.heatmap(df_corr, annot=True, fmt=".1f", ax=ax, cmap="coolwarm")  # Optional: Aa
plt.show()
```



In [33]:
```
#Randomised split into train and test set

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, random_st
X_test, X_val, y_test, y_val= train_test_split( X_other, y_other, train_size=0.3, r
```

In [34]:
```
x.head()
```

Out[34]:

	Age	Gender	NPS	PurchaseFrequency	PurchaseValue	SubscriptionPlan	WebsitePageVi
0	31	0	3	38	3994.72	0	
1	66	0	6	4	2844.35	1	
2	36	0	3	14	1866.52	2	
3	62	0	1	28	1378.64	3	
4	68	0	3	39	2425.05	4	

5 rows × 21 columns

In [35]:

```
#StandardScaling

ss= StandardScaler()
X_train= ss.fit_transform(X_train)
X_val= ss.fit_transform(X_val)
X_test= ss.fit_transform(X_test)
```

In [36]:

```
list(X_train)[:10]
```

```
Out[36]: [array([ 1.35582596e-01,  0.00000000e+00, -3.67740766e-01,  1.72337871e+00,
                  8.07522462e-02, -1.30009557e+00, -1.43549676e+00, -3.80486275e-01,
                 -1.44949680e+00,  2.44037067e-04, -1.43185490e+00, -1.56884724e+00,
                  9.96170020e-01, -4.78823620e-01, -4.85701783e-01, -3.13234130e-01,
                  6.27411978e-01,  8.19615418e-01,  1.76498664e+00,  1.39321679e+00,
                 -2.10389876e-01]),
          array([ 0.98362217,  0.        ,  0.00944281,  0.33258674,  0.1750178 ,
                 -1.30009557,  1.64338021, -0.45319284, -1.21914406,  1.21872111,
                  0.70150707, -1.22148224, -0.58588634, -0.47882362,  0.53661635,
                  0.02744615,  0.30915167, -0.10681923, -0.28620648, -0.10224586,
                 -0.95604117]),
          array([-1.43002893,  0.        , -0.74492434,  2.0710767 ,  0.78241604,
                 -0.25968572,  0.60555651, -0.59860597, -1.10396769,  1.21872111,
                 -0.72073424, -0.17938727, -0.73655838, -0.9954668 , -0.8264745 ,
                 -0.9945947 ,  1.0729764 ,  0.44904156,  1.39204244, -0.10224586,
                 -1.10517143]),
          array([ 1.50549268e+00,  0.00000000e+00, -3.67740766e-01,  6.80284733e-01,
                  3.00929702e-01, -9.53292288e-01,  1.67797433e+00, -4.53192839e-01,
                  1.66026517e+00,  2.44037067e-04, -1.43185490e+00,  1.21007270e+00,
                  1.91016335e-01,  1.93217791e+00,  3.09241169e+00,  1.21982715e+00,
                  2.13673582e-01, -1.03325388e+00, -8.45622786e-01,  2.71619806e-01,
                 -4.46512786e-01]),
          array([ 0.1355826 ,  0.        , -0.36774077, -0.24690991, -0.67911866,
                  1.47433071,  0.60555651,  1.6552975 ,  1.42991243, -1.21823304,
                 -1.4318549 , -0.52675226,  0.67128345, -0.9954668 , -0.99686085,
                 -0.65391441, -0.93206352, -0.84796695,  0.27320983, -0.10224586,
                 -0.32223757]),
          array([-1.36479512,  0.        , -0.36774077, -0.7684569 , -0.99072307,
                 -0.08628407, -1.15874377, -0.9257855 ,  0.73885422, -1.21823304,
                 -0.72073424, -0.52675226,  0.23339285,  1.24332033,  1.04777542,
                  0.70880672,  0.7547161 ,  1.19018928,  1.57851454,  1.39321679,
                  0.44205501]),
          array([-0.71245698,  0.        , -0.36774077,  0.50643574,  1.18505088,
                  0.08711757,  0.01745642,  1.29176468, -1.21914406,  1.21872111,
                 -0.72073424, -0.52675226, -0.32691878,  0.21003396, -0.8264745 ,
                  0.36812644, -0.83658542, -1.03325388, -0.09973438, -0.47611152,
                 -0.50865039]),
          array([ 9.18388358e-01,  0.00000000e+00,  2.27254426e+00, -1.29000388e+00,
                 -1.28107257e+00,  9.54125783e-01,  5.70962389e-01,  6.37405613e-01,
                  1.08438332e+00,  2.44037067e-04,  1.41262772e+00,  1.67977723e-01,
                 -8.26019898e-01, -3.06609225e-01,  1.95843641e-01, -3.13234130e-01,
                 -8.68411454e-01,  1.00490235e+00,  1.57851454e+00,  1.39321679e+00,
                 -1.00575126e+00]),
          array([ 1.37502505e+00,  0.00000000e+00,  9.44280932e-03, -4.78708570e-01,
                 -4.80277936e-01, -1.47349722e+00,  1.64338021e+00, -1.26013303e-01,
                  1.31473606e+00,  2.44037067e-04, -9.61358788e-03,  5.15342715e-01,
                 -7.13015872e-01, -3.06609225e-01,  5.36616353e-01, -1.42893988e-01,
                  1.77314908e+00,  7.84676976e-02,  2.13793085e+00,  8.32418301e-01,
                 -1.01817878e+00]),
          array([ 0.72268692,  0.        , -1.12210792, -0.24690991, -0.11635133,
                 -1.64689886, -0.67442605, -1.10755191,  1.19955969, -1.21823304,
                 -0.00961359, -1.56884724,  3.56701161, -0.30660923,  0.02545728,
                 -0.65391441,  1.70949701, -0.47739309,  0.64615403, -0.10224586,
                  0.4109862 ])]
```

Modeling and Evaluation

LogisticsRegression DecisionTreeClassifier

Metrics *Acuuracy Score* Precision Score *F1 score* Recall Score

In [38]:
```
#Evaluate

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

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

In [39]:
```
#Build Model with Linear Regression
lr= LogisticRegression()
lr.fit(X_train, y_train)

#Evaluate the model
evaluate(X_train, y_train, lr, "Train")
evaluate(X_val, y_val, lr, "Validation")
```

```
Train Accuracy Score: 0.9712597636691368
Train Precision Score: 0.9665288442606812
Train Recall Score: 0.9767210505372065
Train F1 Score: 0.9715982187036121
Validation Accuracy Score: 0.9719679633867276
Validation Precision Score: 0.9677047289504037
Validation Recall Score: 0.9755813953488373
Validation F1 Score: 0.971627099015634
```

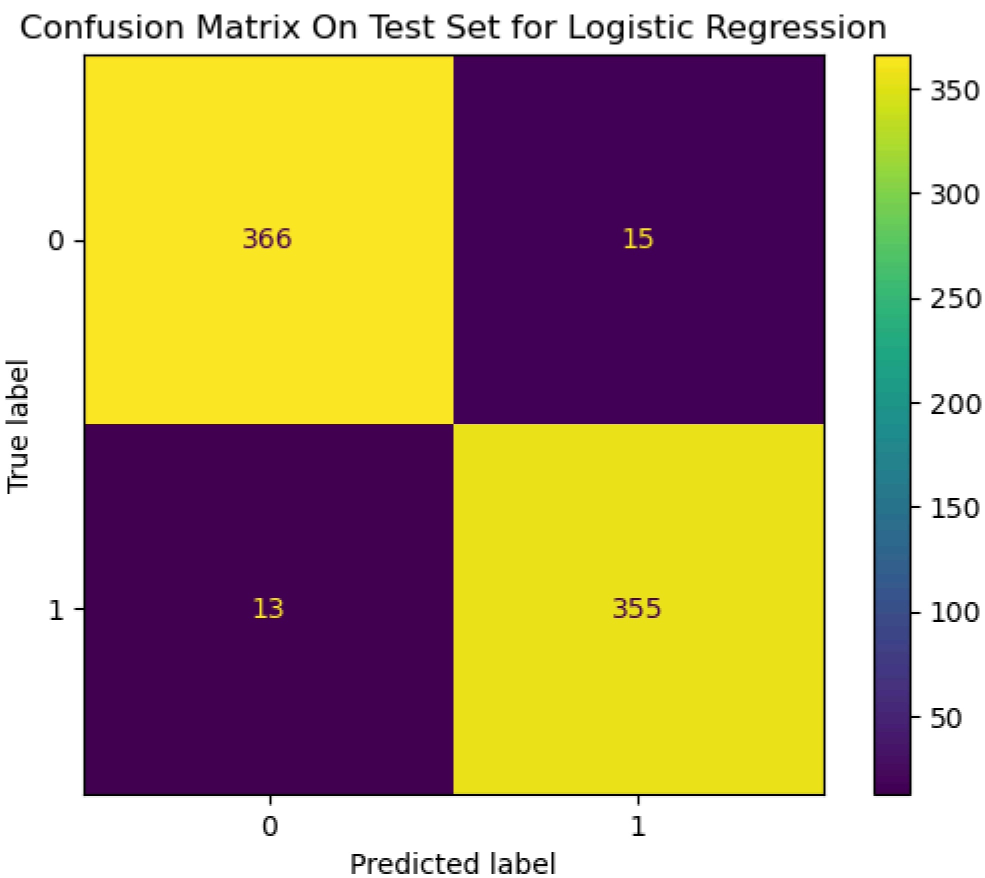
In [40]:
```
# Build Model with DecisionTreeClassifier
dt= DecisionTreeClassifier(max_depth=5)
dt.fit(X_train, y_train)

#Evaluate on train and validate subsets
evaluate(X_train, y_train, dt, "Train")
evaluate(X_val, y_val, dt, "Validation")
```

```
Train Accuracy Score: 0.9766673342679751
Train Precision Score: 0.9769639692852924
Train Recall Score: 0.9771563607719574
Train F1 Score: 0.9770601555577434
Validation Accuracy Score: 0.9708237986270023
Validation Precision Score: 0.972318339100346
Validation Recall Score: 0.9689655172413794
Validation F1 Score: 0.9706390328151986
```

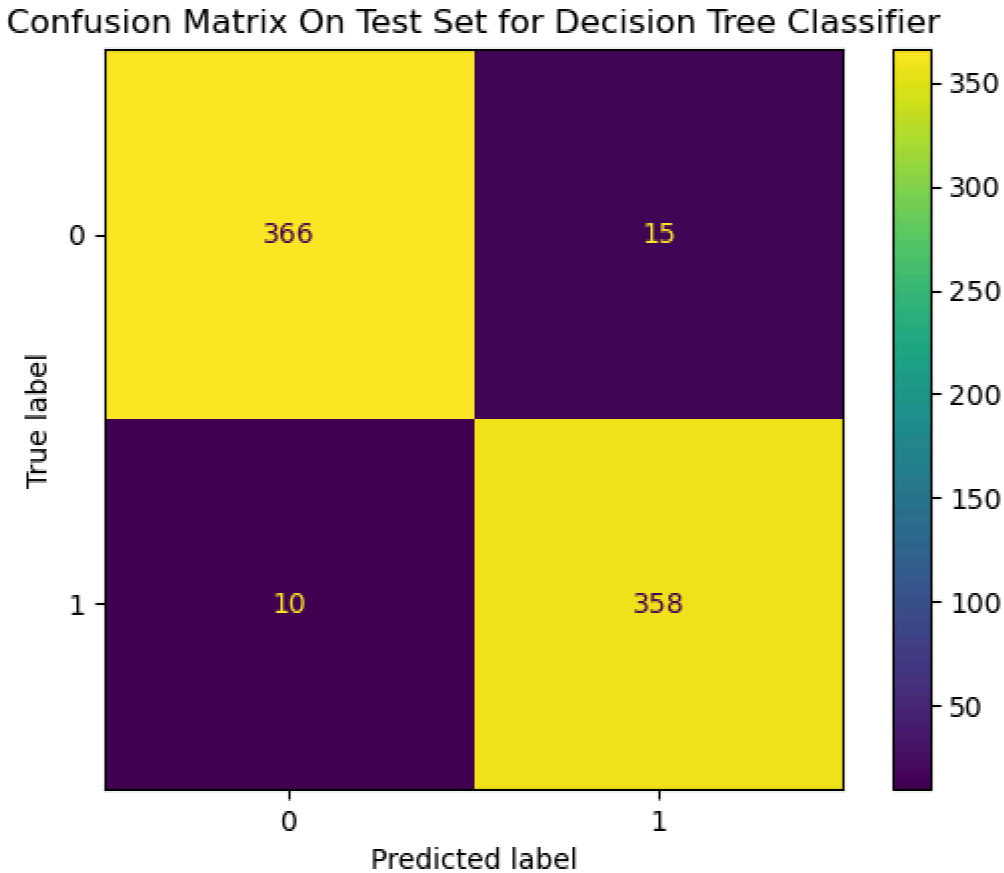
In [41]:
```
# Evaluate on Test Set
evaluate(X_test, y_test, lr, "LogisticRegression Test")
evaluate(X_test, y_test, dt, "DescisionTreesClassifier Test")
```

```
LogisticRegression Test Accuracy Score: 0.9626168224299065
LogisticRegression Test Precision Score: 0.9646739130434783
LogisticRegression Test Recall Score: 0.9594594594594594
LogisticRegression Test F1 Score: 0.962059620596206
DescisionTreesClassifier Test Accuracy Score: 0.9666221628838452
DescisionTreesClassifier Test Precision Score: 0.9728260869565217
DescisionTreesClassifier Test Recall Score: 0.9597855227882037
DescisionTreesClassifier Test F1 Score: 0.9662618083670715
```

In [42]:
```
#Plot the confusion matrx
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_matr
display.plot()
plt.title("Confusion Matrix On Test Set for Logistic Regression")
plt.show()
```



In [81]:
```
dt_y_pred= dt.predict(X_test)
decision_tree_classifier_confision_matixr= confusion_matrix(y_test, dt_y_pred)

display= ConfusionMatrixDisplay(confusion_matrix=decision_tree_classifier_confision
display.plot()
plt.title("Confusion Matrix On Test Set for Decision Tree Classifier")
plt.show()
```



CONCLUSION

The most important features: -the number of service interaction the customer has had through calls, email and chat - the number of times customers had made Late Payment - the time spent on the company website - the net promotion score