[1]:
```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

```
/kaggle/input/datasets-for-churn-telecom/cell2celltrain.csv
/kaggle/input/datasets-for-churn-telecom/cell2cellholdout.csv
```

1.Load The Dataset

[2]:
```
data=pd.read_csv('/kaggle/input/datasets-for-churn-telecom/cell2celltrain.csv')
data.head(10)
```

[2]:

	CustomerID	Churn	MonthlyRevenue	MonthlyMinutes	TotalRecurringCharge	DirectorAssistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	PercChangeRevenues	...	ReferralsMade
0	3000002	Yes	24.00	219.0	22.0	0.25	0.0	0.0	-157.0	-19.0	...	
1	3000010	Yes	16.99	10.0	17.0	0.00	0.0	0.0	-4.0	0.0	...	
2	3000014	No	38.00	8.0	38.0	0.00	0.0	0.0	-2.0	0.0	...	
3	3000022	No	82.28	1312.0	75.0	1.24	0.0	0.0	157.0	8.1	...	
4	3000026	Yes	17.14	0.0	17.0	0.00	0.0	0.0	0.0	-0.2	...	
5	3000030	No	38.05	682.0	52.0	0.25	0.0	0.0	148.0	-3.1	...	
6	3000038	No	31.66	26.0	30.0	0.25	0.0	0.0	60.0	4.0	...	
7	3000042	No	62.13	98.0	66.0	2.48	0.0	0.0	24.0	6.8	...	
8	3000046	No	35.30	24.0	35.0	0.00	0.0	0.0	20.0	-0.3	...	
9	3000050	No	81.00	1056.0	75.0	0.00	0.0	0.0	43.0	2.4	...	

10 rows × 58 columns

2.Removing Null Values from Data

[3]:
```
total_null=data.isnull().sum().sort_values(ascending=False)
percent = ((data.isnull().sum()/data.isnull().count())*100).sort_values(ascending = False)
print("Total records = ", data.shape[0])
missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In Percent'])
missing_data.head(5)
```

```
Total records =  51047
```

[3]:

	Total Missing	In Percent
AgeHH2	909	1.78
AgeHH1	909	1.78
PercChangeMinutes	367	0.72
PercChangeRevenues	367	0.72
MonthlyRevenue	156	0.31

[4]:
```
missing_values = data.isnull()
missing_values_count = missing_values.sum()
# since missing % of data are nearly 1-2% so we can drop complete rows containg NaN vlaues
data_without_missing_values = data.dropna(axis=0)
data_without_missing_values = data.dropna(axis=1)
data.dropna(axis=0,inplace=True)
```

```
data.dropna(axis=1,inplace=True)
duplicate_rows = data.duplicated()
data_without_duplicates = data.drop_duplicates()
```

[5]:
```
total_null=data.isnull().sum().sort_values(ascending=False)
percent = ((data.isnull().sum()/data.isnull().count())*100).sort_values(ascending = False)
print("Total records = ", data.shape[0])
missing_data = pd.concat([total_null,percent.round(2)],axis=1,keys=['Total Missing','In Percent'])
missing_data.head(5)
```

```
Total records =  49752
```

[5]:

	Total Missing	In Percent
CustomerID	0	0.0
HasCreditCard	0	0.0
AgeHH2	0	0.0
ChildrenInHH	0	0.0
HandsetRefurbished	0	0.0

[6]:
```
cleaned_data_summary = pd.DataFrame({
    'Original Data': [data_without_missing_values.shape, missing_values_count.sum(), duplicate_rows.sum()],
    'Cleaned Data': [data.shape, 0, data_without_duplicates.shape]
}, index=['Shape', 'Missing Values', 'Duplicate Rows'])
cleaned_data_summary
```

[6]:

	Original Data	Cleaned Data
Shape	(51047, 44)	(49752, 58)
Missing Values	3515	0
Duplicate Rows	0	(49752, 58)

3.Exploratory Data Analysis

[7]:
```
import seaborn as sns
import matplotlib.pyplot as plt

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

[8]:
```
# Check the dimensions of the dataset
print(data.shape)
```

```
(49752, 58)
```

[9]:
```
# Display the column names
print(data.columns)
```

```
Index(['CustomerID', 'Churn', 'MonthlyRevenue', 'MonthlyMinutes',
       'TotalRecurringCharge', 'DirectorAssistedCalls', 'OverageMinutes',
       'RoamingCalls', 'PercChangeMinutes', 'PercChangeRevenues',
       'DroppedCalls', 'BlockedCalls', 'UnansweredCalls', 'CustomerCareCalls',
       'ThreewayCalls', 'ReceivedCalls', 'OutboundCalls', 'InboundCalls',
       'PeakCallsInOut', 'OffPeakCallsInOut', 'DroppedBlockedCalls',
       'CallForwardingCalls', 'CallWaitingCalls', 'MonthsInService',
       'UniqueSubs', 'ActiveSubs', 'ServiceArea', 'Handsets', 'HandsetModels',
       'CurrentEquipmentDays', 'AgeHH1', 'AgeHH2', 'ChildrenInHH',
       'HandsetRefurbished', 'HandsetWebCapable', 'TruckOwner', 'RVOwner',
       'Homeownership', 'BuysViaMailOrder', 'RespondsToMailOffers',
       'OptOutMailings', 'NonUSTravel', 'OwnsComputer', 'HasCreditCard',
       'RetentionCalls', 'RetentionOffersAccepted', 'NewCellphoneUser',
       'NotNewCellphoneUser', 'ReferralsMadeBySubscriber', 'IncomeGroup',
       'OwnsMotorcycle', 'AdjustmentsToCreditRating', 'HandsetPrice',
       'MadeCallToRetentionTeam', 'CreditRating', 'PrizmCode', 'Occupation',
       'MaritalStatus'],
      dtype='object')
```

[10]:
```
# Calculate summary statistics
data.describe(include="all")
```

[10]:

	CustomerID	Churn	MonthlyRevenue	MonthlyMinutes	TotalRecurringCharge	DirectorAssistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	PercChangeRevenues	...	Referra
count	4.975200e+04	49752	49752.000000	49752.000000	49752.000000	49752.000000	49752.000000	49752.000000	49752.000000	49752.000000	...	

unique	NaN	2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
top	NaN	No	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
freq	NaN	35507	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...
mean	3.200958e+06	NaN	58.717374	525.021466	46.834961	0.891841	39.754040	1.226327	-11.656175	-1.175937	...
std	1.164694e+05	NaN	44.317244	528.510155	23.759492	2.224725	96.091606	9.848360	255.858193	39.392124	...
min	3.000002e+06	NaN	-6.170000	0.000000	-6.000000	0.000000	0.000000	0.000000	-3875.000000	-1107.700000	...
25%	3.100094e+06	NaN	33.610000	158.000000	30.000000	0.000000	0.000000	0.000000	-83.000000	-7.000000	...
50%	3.200144e+06	NaN	48.380000	366.000000	45.000000	0.250000	3.000000	0.000000	-5.000000	-0.300000	...
75%	3.303643e+06	NaN	70.910000	722.000000	60.000000	0.990000	40.000000	0.300000	65.000000	1.600000	...
max	3.399974e+06	NaN	1223.380000	7359.000000	400.000000	159.390000	4321.000000	1112.400000	5192.000000	2483.500000	...

11 rows × 58 columns

[11]:
```
data.nunique()
```

[11]:
```
CustomerID                  49752
Churn                           2
MonthlyRevenue              12504
MonthlyMinutes               2706
TotalRecurringCharge          211
DirectorAssistedCalls         143
OverageMinutes                739
RoamingCalls                  522
PercChangeMinutes            2236
PercChangeRevenues           2872
DroppedCalls                  283
BlockedCalls                  369
UnansweredCalls               832
CustomerCareCalls             183
ThreewayCalls                  78
ReceivedCalls                6061
OutboundCalls                 778
InboundCalls                  465
PeakCallsInOut               1824
OffPeakCallsInOut            1657
DroppedBlockedCalls           443
CallForwardingCalls            30
CallWaitingCalls              221
MonthsInService                56
UniqueSubs                     15
ActiveSubs                     10
ServiceArea                   743
Handsets                       23
HandsetModels                  13
CurrentEquipmentDays         1421
AgeHH1                         43
AgeHH2                         43
ChildrenInHH                    2
HandsetRefurbished              2
HandsetWebCapable               2
TruckOwner                      2
RVOwner                         2
Homeownership                   2
BuysViaMailOrder                2
RespondsToMailOffers            2
OptOutMailings                  2
NonUSTravel                     2
OwnsComputer                    2
HasCreditCard                   2
RetentionCalls                  5
RetentionOffersAccepted         4
NewCellphoneUser                2
NotNewCellphoneUser             2
ReferralsMadeBySubscriber      12
IncomeGroup                    10
OwnsMotorcycle                  2
AdjustmentsToCreditRating      15
HandsetPrice                   16
MadeCallToRetentionTeam         2
CreditRating                    7
PrizmCode                       4
Occupation                      8
MaritalStatus                   3
dtype: int64
```

[12]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49752 entries, 0 to 51043
Data columns (total 58 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              49752 non-null  int64
 1   Churn                   49752 non-null  object
 2   MonthlyRevenue          49752 non-null  float64
 3   MonthlyMinutes          49752 non-null  float64
 4   TotalRecurringCharge    49752 non-null  float64
 5   DirectorAssistedCalls   49752 non-null  float64
 6   OverageMinutes          49752 non-null  float64
 7   RoamingCalls            49752 non-null  float64
 8   PercChangeMinutes       49752 non-null  float64
 9   PercChangeRevenues      49752 non-null  float64
 10  DroppedCalls            49752 non-null  float64
 11  BlockedCalls            49752 non-null  float64
 12  UnansweredCalls         49752 non-null  float64
 13  CustomerCareCalls       49752 non-null  float64
 14  ThreewayCalls           49752 non-null  float64
 15  ReceivedCalls           49752 non-null  float64
 16  OutboundCalls           49752 non-null  float64
 17  InboundCalls            49752 non-null  float64
```

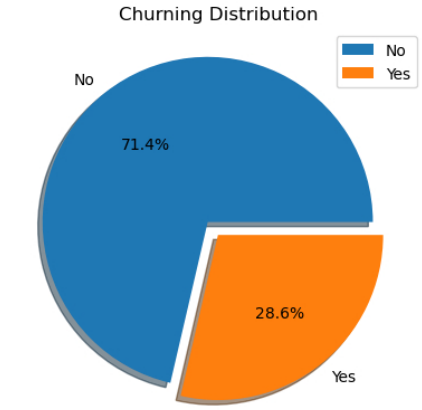
```
 18  PeakCallsInOut             49752 non-null  float64
 19  OffPeakCallsInOut          49752 non-null  float64
 20  DroppedBlockedCalls        49752 non-null  float64
 21  CallForwardingCalls        49752 non-null  float64
 22  CallWaitingCalls           49752 non-null  float64
 23  MonthsInService            49752 non-null  int64
 24  UniqueSubs                 49752 non-null  int64
 25  ActiveSubs                 49752 non-null  int64
 26  ServiceArea                49752 non-null  object
 27  Handsets                   49752 non-null  float64
 28  HandsetModels              49752 non-null  float64
 29  CurrentEquipmentDays       49752 non-null  float64
 30  AgeHH1                     49752 non-null  float64
 31  AgeHH2                     49752 non-null  float64
 32  ChildrenInHH               49752 non-null  object
 33  HandsetRefurbished         49752 non-null  object
 34  HandsetWebCapable          49752 non-null  object
 35  TruckOwner                 49752 non-null  object
 36  RVOwner                    49752 non-null  object
 37  Homeownership              49752 non-null  object
 38  BuysViaMailOrder           49752 non-null  object
 39  RespondsToMailOffers       49752 non-null  object
 40  OptOutMailings             49752 non-null  object
 41  NonUSTravel                49752 non-null  object
 42  OwnsComputer               49752 non-null  object
 43  HasCreditCard              49752 non-null  object
 44  RetentionCalls             49752 non-null  int64
 45  RetentionOffersAccepted    49752 non-null  int64
 46  NewCellphoneUser           49752 non-null  object
 47  NotNewCellphoneUser        49752 non-null  object
 48  ReferralsMadeBySubscriber  49752 non-null  int64
 49  IncomeGroup                49752 non-null  int64
 50  OwnsMotorcycle             49752 non-null  object
 51  AdjustmentsToCreditRating  49752 non-null  int64
 52  HandsetPrice               49752 non-null  object
 53  MadeCallToRetentionTeam    49752 non-null  object
 54  CreditRating               49752 non-null  object
 55  PrizmCode                  49752 non-null  object
 56  Occupation                 49752 non-null  object
 57  MaritalStatus              49752 non-null  object
dtypes: float64(26), int64(9), object(23)
memory usage: 22.4+ MB
```

3a).Churn Distribution

[13]:
```
Churn_types = content_types = data['Churn'].value_counts()
Churn_types
```

[13]:
```
No     35507
Yes    14245
Name: Churn, dtype: int64
```

[14]:
```
label=Churn_types.index
explode=[0.1,0]
plt.pie(x=Churn_types.values, labels=label,shadow=True,explode=explode,autopct='%.1f%%')
plt.title('Churning Distribution')
plt.legend()
plt.show()
```



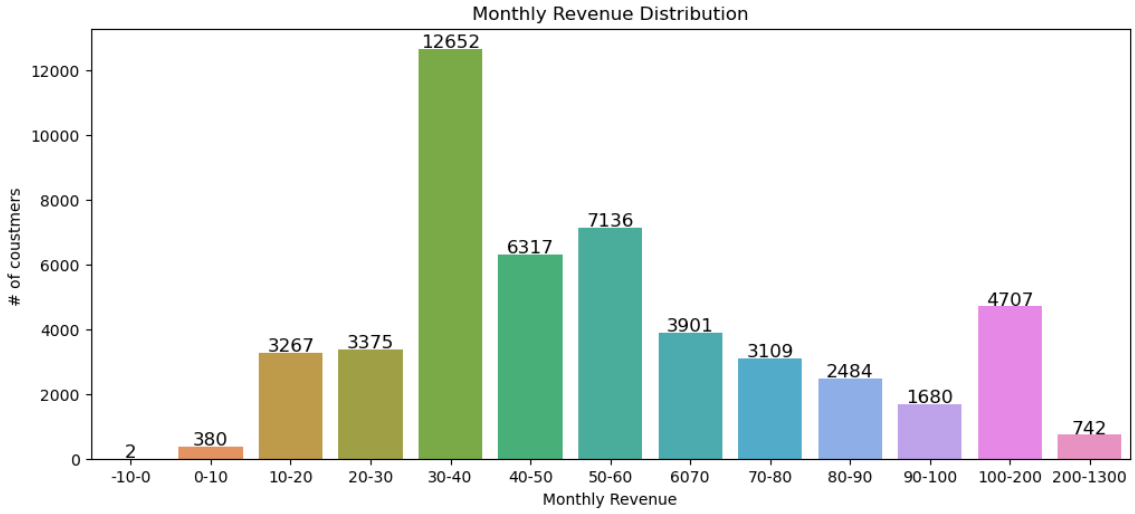
3b).Monthly Revenue Distribution

[15]:
```
monthly_revenue = data['MonthlyRevenue']
monthly_revenue_distribution=np.histogram(monthly_revenue,bins=[-10,0,10,20,30,40,50,60,70,80,90,100,200,1300])
monthly_revenue_distribution_label=['-10-0','0-10','10-20','20-30','30-40','40-50','50-60','6070','70-80','80-90','90-100','100-200','200-
monthly_revenue_distribution
```

[15]:
```
(array([    2,   380,  3267,  3375, 12652,  6317,  7136,  3901,  3109,
```

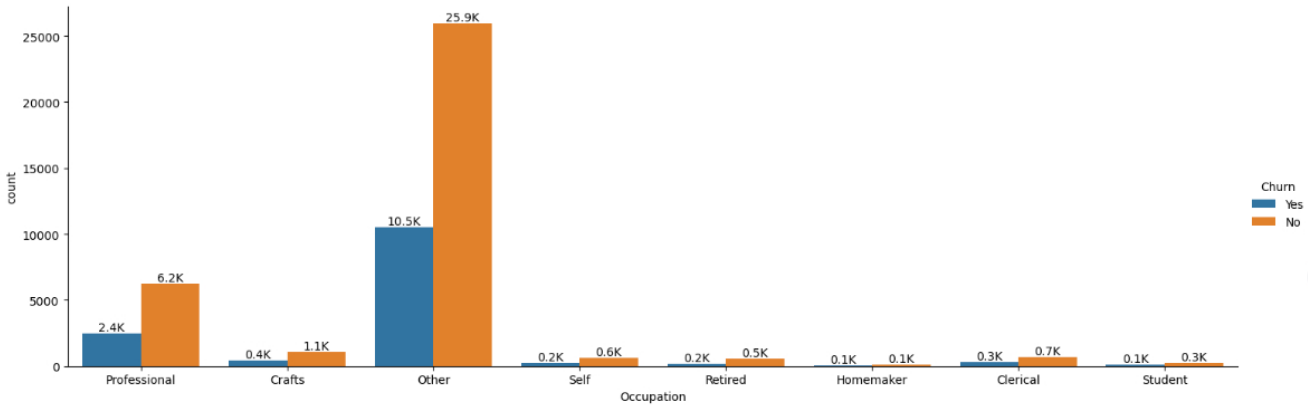
```
        2484,  1680,  4707,   742]),
array([ -10,    0,   10,   20,   30,   40,   50,   60,   70,   80,   90,
        100,  200, 1300]))
```

[16]:
```
plt.figure(figsize=(12,5))
sns.barplot(x=monthly_revenue_distribution_label, y=monthly_revenue_distribution[0])
plt.xlabel('Monthly Revenue')
plt.ylabel('# of coustmers ')
plt.title('Monthly Revenue Distribution')
for i, value in enumerate(monthly_revenue_distribution[0]):
    plt.text(i, value+200, str(value), fontsize=12, color='black', horizontalalignment='center',verticalalignment='center')
plt.show()
```



3c). Plot of Occupation with Churning

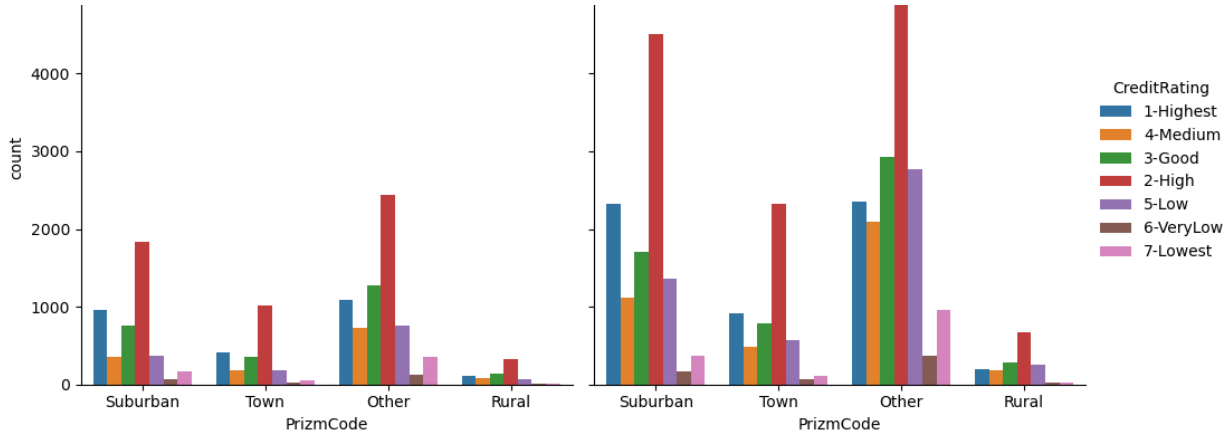
[17]:
```
g=sns.catplot(x="Occupation",hue="Churn", kind="count", data=data,height=5, aspect=3)
# extract the matplotlib axes_subplot objects from the FacetGrid
ax = g.facet_axis(0, 0)  # or ax = g.axes.flat[0]

# iterate through the axes containers
for c in ax.containers:
    labels = [f'{(v.get_height() / 1000):.1f}K' for v in c]
    ax.bar_label(c, labels=labels, label_type='edge')
```



3d).Plot of PrizmCode and CreditRating with Churning.

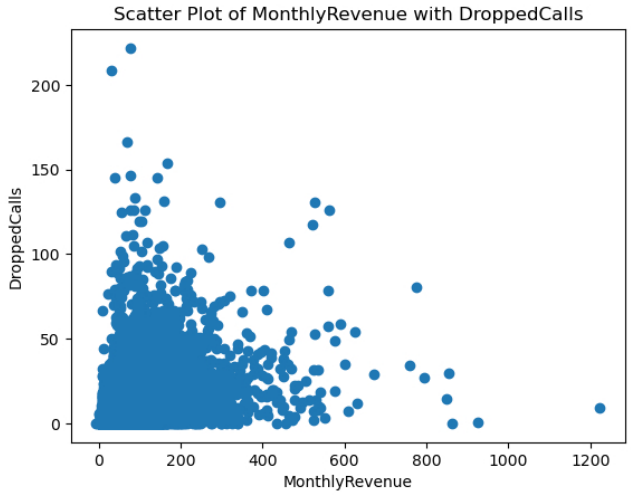
[18]:
```
sns.catplot(x="PrizmCode",hue="CreditRating",kind="count", col='Churn', data=data)
```

[18]:
```
<seaborn.axisgrid.FacetGrid at 0x7c95ad64dde0>
```



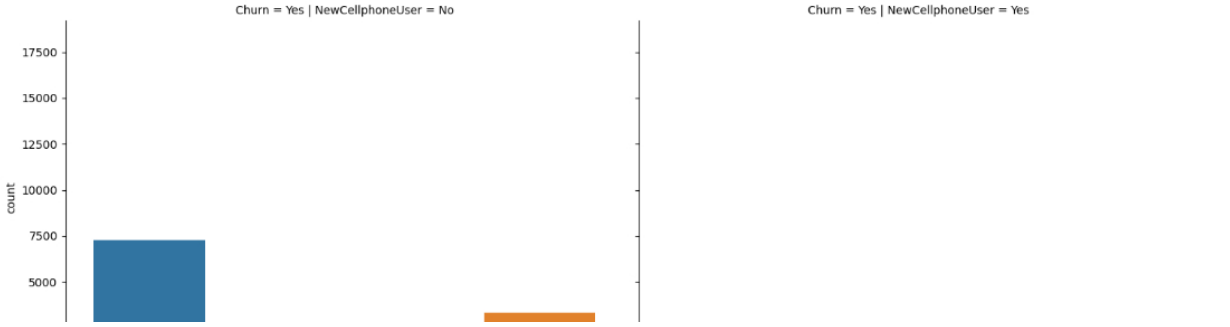
3e). Scatter Plot of MonthlyRevenue with DroppedCalls

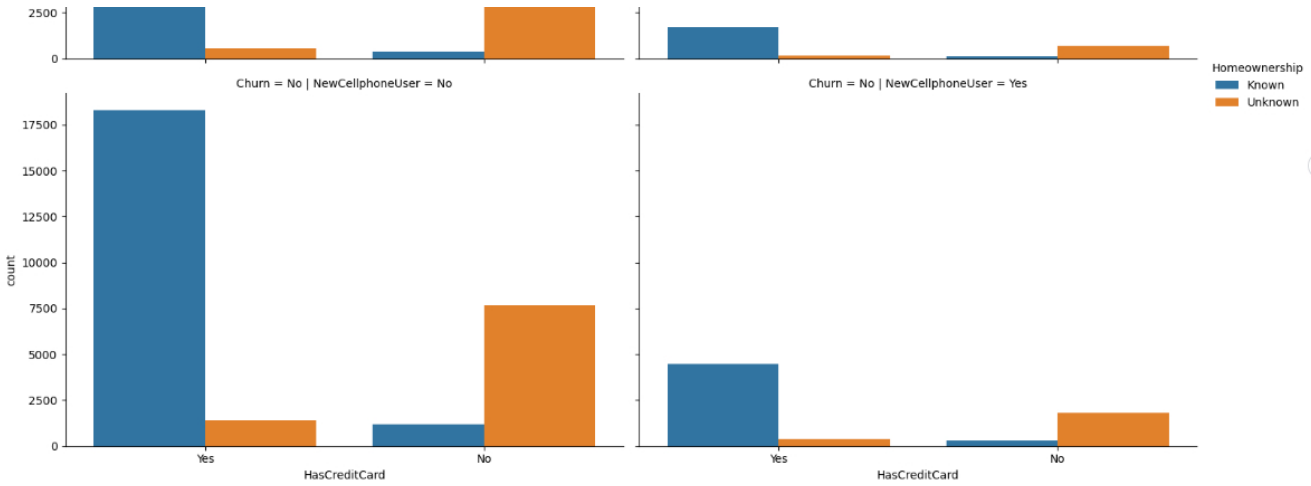
[19]:
```
plt.scatter(data['MonthlyRevenue'], data['DroppedCalls'])
plt.xlabel('MonthlyRevenue')
plt.ylabel('DroppedCalls')
plt.title('Scatter Plot of MonthlyRevenue with DroppedCalls')
plt.show()
```



3f).Catplot of HasCreditCard, NewCellphoneUser and Homeownership with Churning

[20]:
```
sns.catplot(
    data=data,
    x='HasCreditCard',
    kind='count',
    col='NewCellphoneUser',
    row='Churn',
    hue='Homeownership',
    aspect=1.5
)
plt.show()
```

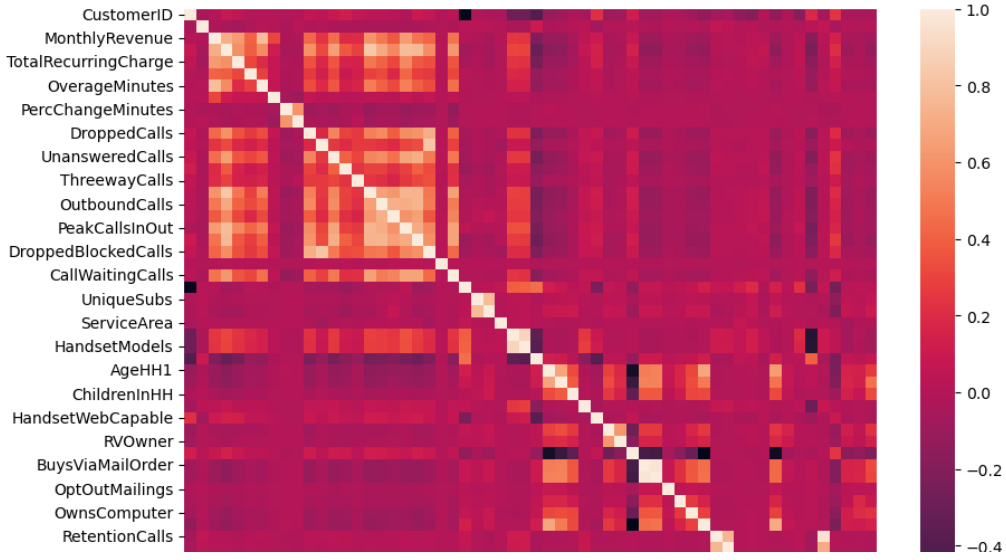


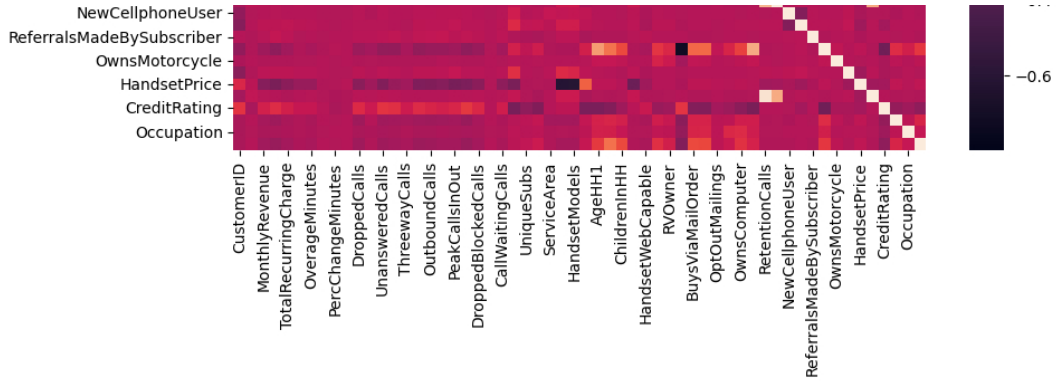


3g).Correlation Diagram

[21]:
```
#Label_encoding For sentimens.
from sklearn import preprocessing
label_encoder = preprocessing.LabelEncoder()
data['Churn']=label_encoder.fit_transform(data['Churn'])
data['ServiceArea']=label_encoder.fit_transform(data['ServiceArea'])
data['ChildrenInHH']=label_encoder.fit_transform(data['ChildrenInHH'])
data['HandsetRefurbished']=label_encoder.fit_transform(data['HandsetRefurbished'])
data['HandsetWebCapable']=label_encoder.fit_transform(data['HandsetWebCapable'])
data['TruckOwner']=label_encoder.fit_transform(data['TruckOwner'])
data['RVOwner']=label_encoder.fit_transform(data['RVOwner'])
data['Homeownership']=label_encoder.fit_transform(data['Homeownership'])
data['BuysViaMailOrder']=label_encoder.fit_transform(data['BuysViaMailOrder'])
data['RespondsToMailOffers']=label_encoder.fit_transform(data['RespondsToMailOffers'])
data['OptOutMailings']=label_encoder.fit_transform(data['OptOutMailings'])
data['NonUSTravel']=label_encoder.fit_transform(data['NonUSTravel'])
data['OwnsComputer']=label_encoder.fit_transform(data['OwnsComputer'])
data['HasCreditCard']=label_encoder.fit_transform(data['HasCreditCard'])
data['NewCellphoneUser']=label_encoder.fit_transform(data['NewCellphoneUser'])
data['NotNewCellphoneUser']=label_encoder.fit_transform(data['NotNewCellphoneUser'])
data['OwnsMotorcycle']=label_encoder.fit_transform(data['OwnsMotorcycle'])
data['HandsetPrice']=label_encoder.fit_transform(data['HandsetPrice'])
data['MadeCallToRetentionTeam']=label_encoder.fit_transform(data['MadeCallToRetentionTeam'])
data['CreditRating']=label_encoder.fit_transform(data['CreditRating'])
data['PrizmCode']=label_encoder.fit_transform(data['PrizmCode'])
data['Occupation']=label_encoder.fit_transform(data['Occupation'])
data['MaritalStatus']=label_encoder.fit_transform(data['MaritalStatus'])
```

[22]:
```
corr = data.corr()
plt.figure(figsize=(10,8))
sns.heatmap(corr)
plt.show()
```





4.Feature Engineering and Selection

[23]:
```
# Calculate correlation matrix
correlation_matrix = data.corr()

# Identify features highly correlated with the target variable ('churn')
highly_correlated_features = correlation_matrix['Churn'][abs(correlation_matrix['Churn']) > 0.02]

# Select the highly correlated features
selected_features = highly_correlated_features.index.tolist()
selected_features.remove('Churn')

# Display the selected features
a=selected_features
```

[24]:
```
a
```

[24]:
```
['CustomerID',
 'MonthlyMinutes',
 'TotalRecurringCharge',
 'PercChangeMinutes',
 'UnansweredCalls',
 'CustomerCareCalls',
 'ThreewayCalls',
 'ReceivedCalls',
 'OutboundCalls',
 'InboundCalls',
 'PeakCallsInOut',
 'OffPeakCallsInOut',
 'CallWaitingCalls',
 'UniqueSubs',
 'Handsets',
 'HandsetModels',
 'CurrentEquipmentDays',
 'AgeHH1',
 'HandsetRefurbished',
 'HandsetWebCapable',
 'BuysViaMailOrder',
 'RespondsToMailOffers',
 'RetentionCalls',
 'RetentionOffersAccepted',
 'HandsetPrice',
 'MadeCallToRetentionTeam',
 'CreditRating']
```

[25]:
```
from sklearn.ensemble import RandomForestClassifier

# Prepare the feature matrix and target variable
X = data.drop(columns=['Churn'])
y = data['Churn']

# Train a Random Forest classifier
rf_model = RandomForestClassifier()
rf_model.fit(X, y)

# Get feature importances
feature_importances = pd.Series(rf_model.feature_importances_, index=X.columns).sort_values(ascending=False)

# Select features with importance above a certain threshold
selected_features = feature_importances[feature_importances > 0.02].index.tolist()

# Display the selected features
b=selected_features
```

[25]:
```
b
```

[26]:
```
['CurrentEquipmentDays',
 'PercChangeMinutes',
 'CustomerID',
 'MonthlyMinutes',
 'MonthlyRevenue',
 'PercChangeRevenues',
 'ServiceArea',
 'MonthsInService',
 'PeakCallsInOut',
 'OffPeakCallsInOut',
 'ReceivedCalls',
 'UnansweredCalls',
 'OutboundCalls',
 'DroppedBlockedCalls',
 'TotalRecurringCharge',
 'DroppedCalls',
 'InboundCalls',
 'AgeHH1',
 'OverageMinutes',
 'BlockedCalls']
```

[27]:
```
for i in range(len(a)):
    for j in range(len(b)):
        if(a[i]==b[j]):
            print(a[i])
            break;
```

```
CustomerID
MonthlyMinutes
TotalRecurringCharge
PercChangeMinutes
UnansweredCalls
ReceivedCalls
OutboundCalls
InboundCalls
PeakCallsInOut
OffPeakCallsInOut
CurrentEquipmentDays
AgeHH1
```

[28]:
```
X=data[['Churn','MonthlyMinutes','TotalRecurringCharge','PercChangeMinutes','UnansweredCalls','ReceivedCalls','OutboundCalls',
        'InboundCalls','PeakCallsInOut','OffPeakCallsInOut','CurrentEquipmentDays','AgeHH1']].copy()
# X=data
# X=X.drop(['CustomerID'],axis=1)
X
```

[28]:

	Churn	MonthlyMinutes	TotalRecurringCharge	PercChangeMinutes	UnansweredCalls	ReceivedCalls	OutboundCalls	InboundCalls	PeakCallsInOut	OffPeakCallsInOut	CurrentEquipmentDay
0	1	219.0	22.0	-157.0	6.3	97.2	0.0	0.0	58.0	24.0	361.
1	1	10.0	17.0	-4.0	2.7	0.0	0.0	0.0	5.0	1.0	1504.
2	0	8.0	38.0	-2.0	0.0	0.4	0.3	0.0	1.3	3.7	1812.
3	0	1312.0	75.0	157.0	76.0	200.3	370.3	147.0	555.7	303.7	458.
4	1	0.0	17.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	852.
...	...	...	...	...	...	...	...	...	...	...	.
51035	0	76.0	30.0	0.0	13.0	11.2	5.7	0.0	29.3	17.0	883.
51037	0	63.0	17.0	-38.0	8.7	14.0	0.0	0.0	16.0	14.7	883.
51040	0	724.0	70.0	-40.0	23.3	70.2	25.7	0.3	262.0	12.7	882.
51041	1	384.0	30.0	0.0	27.0	21.7	1.3	0.0	99.3	54.3	489.
51043	0	1745.0	85.0	122.0	41.3	681.5	89.7	33.3	318.7	248.3	464.

49752 rows × 12 columns

[29]:
```
X=data
```

[30]:
```
X.nunique()
```

[30]:
```
CustomerID                49752
Churn                         2
MonthlyRevenue            12504
MonthlyMinutes             2706
TotalRecurringCharge        211
DirectorAssistedCalls       143
OverageMinutes              739
RoamingCalls                522
PercChangeMinutes          2236
PercChangeRevenues         2872
DroppedCalls                283
BlockedCalls                369
UnansweredCalls             832
CustomerCareCalls           183
ThreewayCalls                78
ReceivedCalls              6061
OutboundCalls               778
InboundCalls                465
```

```
PeakCallsInOut               1624
OffPeakCallsInOut            1657
DroppedBlockedCalls           443
CallForwardingCalls            30
CallWaitingCalls              221
MonthsInService                56
UniqueSubs                     15
ActiveSubs                     10
ServiceArea                   743
Handsets                       23
HandsetModels                  13
CurrentEquipmentDays         1421
AgeHH1                         43
AgeHH2                         43
ChildrenInHH                    2
HandsetRefurbished              2
HandsetWebCapable               2
TruckOwner                      2
RVOwner                         2
Homeownership                   2
BuysViaMailOrder                2
RespondsToMailOffers            2
OptOutMailings                  2
NonUSTravel                     2
OwnsComputer                    2
HasCreditCard                   2
RetentionCalls                  5
RetentionOffersAccepted         4
NewCellphoneUser                2
NotNewCellphoneUser             2
ReferralsMadeBySubscriber      12
IncomeGroup                    10
OwnsMotorcycle                  2
AdjustmentsToCreditRating      15
HandsetPrice                   16
MadeCallToRetentionTeam         2
CreditRating                    7
PrizmCode                       4
Occupation                      8
MaritalStatus                   3
dtype: int64
```

[31]:

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 49752 entries, 0 to 51043
Data columns (total 58 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   CustomerID                 49752 non-null  int64
 1   Churn                      49752 non-null  int64
 2   MonthlyRevenue             49752 non-null  float64
 3   MonthlyMinutes             49752 non-null  float64
 4   TotalRecurringCharge       49752 non-null  float64
 5   DirectorAssistedCalls      49752 non-null  float64
 6   OverageMinutes             49752 non-null  float64
 7   RoamingCalls               49752 non-null  float64
 8   PercChangeMinutes          49752 non-null  float64
 9   PercChangeRevenues         49752 non-null  float64
 10  DroppedCalls               49752 non-null  float64
 11  BlockedCalls               49752 non-null  float64
 12  UnansweredCalls            49752 non-null  float64
 13  CustomerCareCalls          49752 non-null  float64
 14  ThreewayCalls              49752 non-null  float64
 15  ReceivedCalls              49752 non-null  float64
 16  OutboundCalls              49752 non-null  float64
 17  InboundCalls               49752 non-null  float64
 18  PeakCallsInOut             49752 non-null  float64
 19  OffPeakCallsInOut          49752 non-null  float64
 20  DroppedBlockedCalls        49752 non-null  float64
 21  CallForwardingCalls        49752 non-null  float64
 22  CallWaitingCalls           49752 non-null  float64
 23  MonthsInService            49752 non-null  int64
 24  UniqueSubs                 49752 non-null  int64
 25  ActiveSubs                 49752 non-null  int64
 26  ServiceArea                49752 non-null  int64
 27  Handsets                   49752 non-null  float64
 28  HandsetModels              49752 non-null  float64
 29  CurrentEquipmentDays       49752 non-null  float64
 30  AgeHH1                     49752 non-null  float64
 31  AgeHH2                     49752 non-null  float64
 32  ChildrenInHH               49752 non-null  int64
 33  HandsetRefurbished         49752 non-null  int64
 34  HandsetWebCapable          49752 non-null  int64
 35  TruckOwner                 49752 non-null  int64
 36  RVOwner                    49752 non-null  int64
 37  Homeownership              49752 non-null  int64
 38  BuysViaMailOrder           49752 non-null  int64
 39  RespondsToMailOffers       49752 non-null  int64
 40  OptOutMailings             49752 non-null  int64
 41  NonUSTravel                49752 non-null  int64
 42  OwnsComputer               49752 non-null  int64
 43  HasCreditCard              49752 non-null  int64
 44  RetentionCalls             49752 non-null  int64
 45  RetentionOffersAccepted    49752 non-null  int64
 46  NewCellphoneUser           49752 non-null  int64
 47  NotNewCellphoneUser        49752 non-null  int64
 48  ReferralsMadeBySubscriber  49752 non-null  int64
 49  IncomeGroup                49752 non-null  int64
 50  OwnsMotorcycle             49752 non-null  int64
 51  AdjustmentsToCreditRating  49752 non-null  int64
 52  HandsetPrice               49752 non-null  int64
 53  MadeCallToRetentionTeam    49752 non-null  int64
 54  CreditRating               49752 non-null  int64
 55  PrizmCode                  49752 non-null  int64
 56  Occupation                 49752 non-null  int64
 57  MaritalStatus              49752 non-null  int64
dtypes: float64(26), int64(32)
memory usage: 22.4 MB
```

5 Model Training and Evaluation

Reduction of Data:- Reducing DATA with equal ratio of Churners, so that Churn Distribution should be same for model prediction

[32]:
```
x0=X[X['Churn']==0]
x1=X[X['Churn']==1]
x0.shape, x1.shape
```

[32]: ((35507, 58), (14245, 58))

[33]:
```
x0=x0[:int(x0.shape[0]/1)]
x1=x1[:int(x1.shape[0]/1)]
x0.shape, x1.shape
```

[33]: ((35507, 58), (14245, 58))

[34]:
```
X=pd.concat([x0,x1],axis=0)
X
```

[34]:

	CustomerID	Churn	MonthlyRevenue	MonthlyMinutes	TotalRecurringCharge	DirectorAssistedCalls	OverageMinutes	RoamingCalls	PercChangeMinutes	PercChangeRevenues	...	Referrals
2	3000014	0	38.00	8.0	38.0	0.00	0.0	0.0	-2.0	0.0	...	
3	3000022	0	82.28	1312.0	75.0	1.24	0.0	0.0	157.0	8.1	...	
5	3000030	0	38.05	682.0	52.0	0.25	0.0	0.0	148.0	-3.1	...	
6	3000038	0	31.66	26.0	30.0	0.25	0.0	0.0	60.0	4.0	...	
7	3000042	0	62.13	98.0	66.0	2.48	0.0	0.0	24.0	6.8	...	
...	...	...	...	...	...	...	...	...	...	...	...	...
51019	3399754	1	57.67	535.0	45.0	0.00	64.0	0.0	313.0	44.5	...	
51029	3399846	1	30.24	63.0	30.0	0.25	0.0	0.0	13.0	0.7	...	
51031	3399874	1	33.63	28.0	30.0	0.00	0.0	0.0	0.0	0.0	...	
51033	3399882	1	38.34	320.0	30.0	0.99	21.0	0.0	-66.0	-8.3	...	
51041	3399946	1	117.49	384.0	30.0	0.00	250.0	0.0	0.0	0.0	...	

49752 rows × 58 columns

[35]:
```
# Split the dataset into features (X) and target variable (y)
from sklearn.model_selection import train_test_split
y = X['Churn']
X =X.drop('Churn', axis=1)# Assuming 'churn' is the target variable column


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[36]:
```
# Print the shapes of the datasets
print("Training set - Features:", X_train.shape)
print("Training set - Target variable:", y_train.shape)
print("Testing set - Features:", X_test.shape)
print("Testing set - Target variable:", y_test.shape)
```

```
Training set - Features: (39801, 57)
Training set - Target variable: (39801,)
Testing set - Features: (9951, 57)
Testing set - Target variable: (9951,)
```

Logistic Regression

[37]:
```
from sklearn.linear_model import LogisticRegression

# Create an instance of the LogisticRegression model
model = LogisticRegression()

# Train the model on the training data
model.fit(X_train, y_train)
```

[37]: LogisticRegression

```
LogisticRegression()
```

+ Code + Markdown

[38]:
```
# Make predictions on the testing data
y_pred = model.predict(X_test)
```

[39]:
```
y_pred.shape
```

[39]:
```
(9951,)
```

[40]:
```
y_test.shape
```

[40]:
```
(9951,)
```

[41]:
```
# Evaluate the model's performance using appropriate evaluation metrics
# Example: Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7176163199678425
```

[42]:
```
import pickle
pickle.dump(model,open('model2.pkl','wb'))
```

[43]:
```
coefficients = model.coef_
```

[44]:
```
features=X_train.columns
```

Random Forest

[45]:
```
from sklearn.ensemble import RandomForestClassifier

# Create an instance of the RandomForestClassifier model
model = RandomForestClassifier()

# Train the model on the training data
model.fit(X_train, y_train)
```

[45]:
```
▾ RandomForestClassifier
RandomForestClassifier()
```

[46]:
```
# Make predictions on the testing data
y_pred = model.predict(X_test)
# Evaluate the model's performance using appropriate evaluation metrics
# Example: Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7221384785448699
```

+ Code + Markdown

[ ]:

[47]:
```
import pickle
pickle.dump(model,open('model3.pkl','wb'))
```

[48]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
import eli5
from eli5.sklearn import PermutationImportance
```

[49]:
```
X = data.drop('Churn', axis=1)
y = data['Churn']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
clf = RandomForestClassifier()
clf.fit(X_train, y_train)
perm = PermutationImportance(clf, random_state=42).fit(X_test, y_test)
eli5.show_weights(perm, feature_names=X_test.columns.tolist())
instance = X_test.iloc[0]  # Choose a specific instance from the test set


eli5.show_prediction(clf, instance, feature_names=X_test.columns.tolist())
```

[49]: **y=0** (probability **0.830**) top features

Contribution[?]	Feature
+0.715	<BIAS>
+0.030	PercChangeMinutes
+0.025	PercChangeRevenues
+0.022	MonthlyMinutes
+0.019	DroppedBlockedCalls
+0.017	TotalRecurringCharge
+0.011	UnansweredCalls
+0.011	MonthlyRevenue
+0.009	ServiceArea
+0.009	UniqueSubs
+0.008	BlockedCalls
+0.007	HandsetModels
+0.006	ReceivedCalls
+0.005	IncomeGroup
+0.005	HandsetWebCapable
+0.005	DirectorAssistedCalls
+0.005	OverageMinutes
+0.004	CustomerID
+0.004	HandsetRefurbished
+0.003	OffPeakCallsInOut
+0.003	CallWaitingCalls
+0.003	MonthsInService
+0.003	NotNewCellphoneUser
+0.003	PeakCallsInOut
+0.003	OutboundCalls
+0.002	PrizmCode
+0.002	MadeCallToRetentionTeam
+0.001	RetentionCalls
+0.001	Occupation
+0.001	ActiveSubs
+0.001	OwnsComputer
+0.001	RVOwner
+0.000	OptOutMailings
+0.000	ReferralsMadeBySubscriber
+0.000	DroppedCalls
+0.000	NonUSTravel
+0.000	RetentionOffersAccepted
+0.000	ChildrenInHH
+0.000	Handsets
+0.000	AdjustmentsToCreditRating
+0.000	ThreewayCalls
-0.000	CallForwardingCalls
-0.000	OwnsMotorcycle
-0.000	MaritalStatus
-0.001	Homeownership
-0.001	HasCreditCard
-0.002	TruckOwner
-0.003	AgeHH2
-0.003	NewCellphoneUser
-0.003	RoamingCalls
-0.003	BuysViaMailOrder
-0.004	RespondsToMailOffers
-0.004	CreditRating
-0.009	InboundCalls
-0.010	CustomerCareCalls
-0.011	AgeHH1
-0.021	HandsetPrice
-0.043	CurrentEquipmentDays

XGBoost

[50]:
```
from xgboost import XGBClassifier
```

```
# Create an instance of the XGBClassifier model
model = XGBClassifier()

# Train the model on the training data
model.fit(X_train, y_train)
```

[50]:

```
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_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

[51]:

```
# Make predictions on the testing data
y_pred = model.predict(X_test)

# Evaluate the model's performance using appropriate evaluation metrics
# Example: Calculate accuracy
from sklearn.metrics import accuracy_score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy}")
```

```
Accuracy: 0.7115867751984725
```

with open('churn_model.pkl', 'rb') as file: mod = pickle.load(file)

[52]:

```
importances = model.feature_importances_
```

[53]:

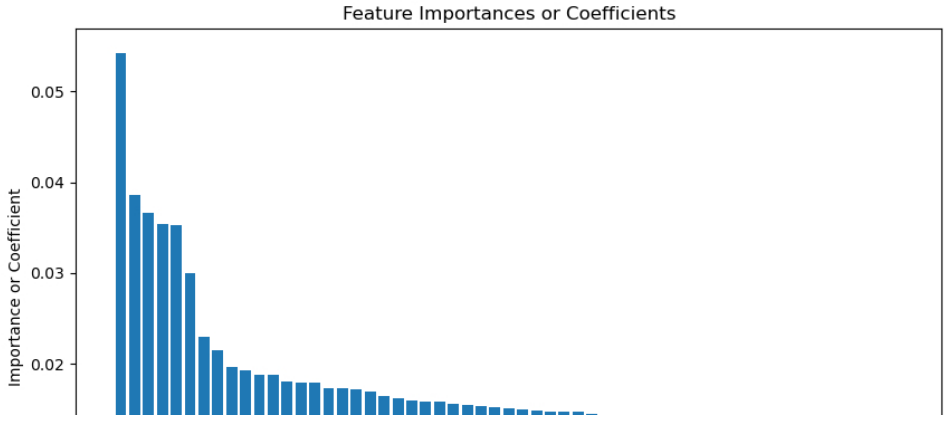
```
features=X_train.columns
```

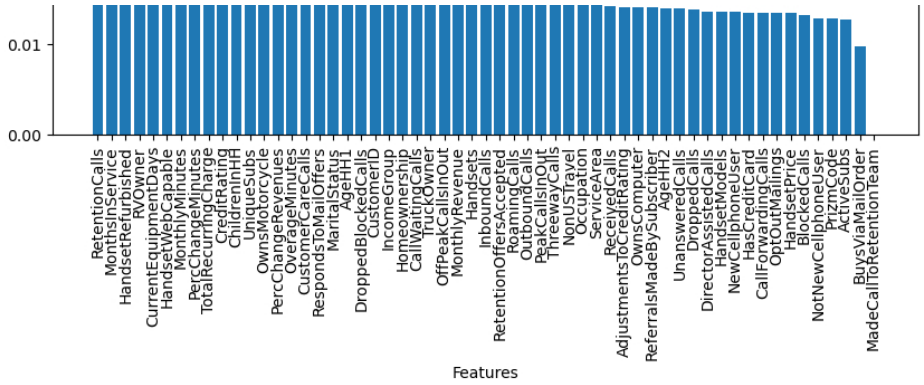
[54]:

```
# Map feature importances or coefficients to feature names
feature_importances = dict(zip(features, importances))
```

[55]:

```
# Sort feature importances or coefficients in descending order
sorted_importances = sorted(feature_importances.items(), key=lambda x: x[1], reverse=True)

# Extract feature names and their corresponding importances or coefficients
sorted_features, sorted_values = zip(*sorted_importances)

# Plot the feature importances or coefficients as a bar chart
plt.figure(figsize=(10, 6))
plt.bar(sorted_features, sorted_values)
plt.xlabel('Features')
plt.ylabel('Importance or Coefficient')
plt.title('Feature Importances or Coefficients')
plt.xticks(rotation=90)
plt.show()
```





[56]:
```
import eli5
from eli5.sklearn import PermutationImportance
```

[57]:
```
# Create a DataFrame to store the sorted feature importances or coefficients
df_importances = pd.DataFrame({'Features': sorted_features, 'Importance or Coefficient': sorted_values})

# Display the sorted feature importances or coefficients as a table
print(df_importances)
```

```
                   Features  Importance or Coefficient
0              RetentionCalls                   0.054246
1            MonthsInService                   0.038539
2          HandsetRefurbished                  0.036602
3                     RVOwner                  0.035320
4        CurrentEquipmentDays                  0.035268
5           HandsetWebCapable                  0.030014
6              MonthlyMinutes                  0.022956
7           PercChangeMinutes                  0.021497
8        TotalRecurringCharge                  0.019634
9                CreditRating                  0.019274
10               ChildrenInHH                  0.018782
11                 UniqueSubs                  0.018775
12              OwnsMotorcycle                 0.017943
13         PercChangeRevenues                  0.017843
14             OverageMinutes                  0.017838
15          CustomerCareCalls                  0.017318
16       RespondsToMailOffers                  0.017200
17              MaritalStatus                  0.017089
18                      AgeHH1                 0.016937
19        DroppedBlockedCalls                  0.016432
20                 CustomerID                  0.016101
21               IncomeGroup                   0.015931
22             Homeownership                   0.015802
23           CallWaitingCalls                  0.015744
24                 TruckOwner                  0.015536
25          OffPeakCallsInOut                  0.015452
26             MonthlyRevenue                  0.015272
27                    Handsets                 0.015213
28               InboundCalls                  0.015028
29    RetentionOffersAccepted                  0.014931
30               RoamingCalls                  0.014841
31              OutboundCalls                  0.014634
32             PeakCallsInOut                  0.014627
33              ThreewayCalls                  0.014623
34                NonUSTravel                  0.014452
35                 Occupation                  0.014300
36                ServiceArea                  0.014279
37              ReceivedCalls                  0.014224
38  AdjustmentsToCreditRating                  0.014101
39               OwnsComputer                  0.014093
40  ReferralsMadeBySubscriber                  0.014021
41                      AgeHH2                 0.013984
42            UnansweredCalls                  0.013958
43               DroppedCalls                  0.013800
44      DirectorAssistedCalls                  0.013567
45               HandsetModels                 0.013502
46            NewCellphoneUser                  0.013499
47               HasCreditCard                 0.013452
48         CallForwardingCalls                  0.013419
49              OptOutMailings                  0.013408
50               HandsetPrice                  0.013400
51               BlockedCalls                  0.013228
52         NotNewCellphoneUser                  0.012794
53                   PrizmCode                  0.012784
54                 ActiveSubs                   0.012722
55           BuysViaMailOrder                   0.009773
56    MadeCallToRetentionTeam                   0.000000
```

Hyperparameter Tuning

[58]:
```
from sklearn.linear_model import LogisticRegression

# Initialize the logistic regression model
model = LogisticRegression()
```

[59]:
```
# Define the hyperparameters for logistic regression
hyperparameters = {
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs'],
    'max_iter': [100, 200, 300]
}
```

[60]:
```
from sklearn.model_selection import GridSearchCV

# Perform grid search cross-validation
grid_search = GridSearchCV(model, hyperparameters, cv=5)
grid_search.fit(X_train, y_train)
```

[60]:
GridSearchCV
estimator: LogisticRegression
LogisticRegression

[61]:
```
# Get the best hyperparameters
best_params = grid_search.best_params_
print("Best Hyperparameters:", best_params)
```

```
Best Hyperparameters: {'C': 0.01, 'max_iter': 100, 'solver': 'liblinear'}
```

[62]:
```
# Train the model with the best hyperparameters
model = LogisticRegression(**best_params)
model.fit(X_train, y_train)
```

[62]:
LogisticRegression
LogisticRegression(C=0.01, solver='liblinear')

[63]:
```
from sklearn.metrics import classification_report

# Generate predictions on the testing set
y_pred = model.predict(X_test)

# Evaluate the model's performance
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.71      1.00      0.83      7034
           1       0.00      0.00      0.00      2917

    accuracy                           0.71      9951
   macro avg       0.35      0.50      0.41      9951
weighted avg       0.50      0.71      0.59      9951
```

Evaluation Metrics

[64]:
```
from sklearn.linear_model import LogisticRegression

# Create a Logistic Regression classifier
classifier = LogisticRegression()

# Train the classifier on the training data
classifier.fit(X_train, y_train)
```

[64]:
LogisticRegression
LogisticRegression()

[65]:
```
from sklearn.model_selection import GridSearchCV

# Define the hyperparameters to tune
parameters = {'C': [0.1, 1, 10]}
```

```
# Create a GridSearchCV object
grid_search = GridSearchCV(classifier, parameters)

# Perform grid search on the training data
grid_search.fit(X_train, y_train)

# Get the best hyperparameters
best_params = grid_search.best_params_
```

[66]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

# Predict the target variable for the testing data
y_pred = classifier.predict(X_test)

# Calculate evaluation metrics
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred)
recall = recall_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
roc_auc
```

[66]: 0.5

[67]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

accuracy = {'TF-IDF':[]}

lr = LinearRegression()
lr.fit(X_train, y_train)
lr_preds = lr.predict(X_test)
lr_acc = accuracy_score(y_test, lr_preds.round())
accuracy['TF-IDF'].append(lr_acc)


lgr = LogisticRegression()
lgr.fit(X_train, y_train)
lgr_preds = lgr.predict(X_test)
lgr_acc = accuracy_score(y_test, lgr_preds)
accuracy['TF-IDF'].append(lgr_acc)

dt = DecisionTreeClassifier()
dt.fit(X_train, y_train)
dt_preds = dt.predict(X_test)
dt_acc = accuracy_score(y_test, dt_preds)
accuracy['TF-IDF'].append(dt_acc)

rf = RandomForestClassifier()
rf.fit(X_train, y_train)
rf_preds = rf.predict(X_test)
rf_acc = accuracy_score(y_test, rf_preds)
accuracy['TF-IDF'].append(rf_acc)

knn = KNeighborsClassifier()
knn.fit(X_train, y_train)
knn_preds = knn.predict(X_test)
knn_acc = accuracy_score(y_test, knn_preds)
accuracy['TF-IDF'].append(knn_acc)

print(f"Linear Regression Accuracy: {lr_acc}")
print(f"Logistic Regression Accuracy: {lgr_acc}")
print(f"Decision Tree Accuracy: {dt_acc}")
print(f"Random Forest Accuracy: {rf_acc}")
print(f"KNN Accuracy: {knn_acc}")
```

```
Linear Regression Accuracy: 0.7051552607778113
Logistic Regression Accuracy: 0.7068636317957994
Decision Tree Accuracy: 0.6124007637423374
Random Forest Accuracy: 0.7174153351421968
KNN Accuracy: 0.6480755702944427
```

[ ]:

[68]:
```
from sklearn.model_selection import GridSearchCV
```

```
tfidf = TfidfVectorizer()
lr = LogisticRegression()
hyperparameters = {
#     'C': [0.01, 0.1, 1, 10, 100],
#     'penalty': ['l1', 'l2']
    'C': [0.01, 0.1, 1, 10],
    'solver': ['liblinear', 'lbfgs'],
    'max_iter': [100, 200, 300]
}
grid_search = GridSearchCV(lr, hyperparameters, cv=5)
grid_search.fit(X_train, y_train)
best_params = grid_search.best_params_

lr = LogisticRegression(**best_params)
lr.fit(X_train, y_train)

a = lr.score(X_test, y_test)
print("Accuracy:", a)
accuracy['TF-IDF'].append(a)
```

```
Accuracy: 0.7068636317957994
```

hyperparameters = { 'max_depth': [5, 10, 15, 20, 25], 'min_samples_split': [2, 5, 10, 15, 20], 'min_samples_leaf': [1, 2, 5, 10, 15] } dt = DecisionTreeClassifier() grid_search = GridSearchCV(dt, hyperparameters, cv=5) grid_search.fit(X_train,y_train) best_params = grid_search.best_params_

dt = DecisionTreeClassifier(**best_params) dt.fit(X_train, y_train) a = dt.score(X_test, y_test) print("Accuracy:", a) accuracy['TF-IDF'].append(a)

param_grid = { 'n_estimators': [50, 100, 200], 'max_depth': [5, 10, 15], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4] } rf = RandomForestClassifier() grid_search = GridSearchCV(rf, param_grid, cv=5) grid_search.fit(X_train, y_train)

best_params = grid_search.best_params_ rf = RandomForestClassifier(**best_params) rf.fit(X_train, y_train) a = rf.score(X_test,y_test) print("Accuracy:", a) accuracy['TF-IDF'].append(a)

hyperparameters = { 'n_neighbors': [3, 5, 7, 9, 11], 'weights': ['uniform', 'distance'], 'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'], 'p': [1, 2] }

knn = KNeighborsClassifier()

grid_search = GridSearchCV(knn, hyperparameters, cv=5) grid_search.fit(X_train,y_train)

best_params = grid_search.best_params_

knn = KNeighborsClassifier(**best_params) knn.fit(X_train, y_train) a = knn.score(X_test,y_test) print("Accuracy:", a) accuracy['TF-IDF'].append(a)

[69]:
```
accuracy['TF-IDF']
```

[69]:
```
[0.7051552607778113,
 0.7068636317957994,
 0.6124007637423374,
 0.7174153351421968,
 0.6480755702944427,
 0.7068636317957994]
```

models = ['Linear Regression', 'Logistic Regression', 'Decision Tree', 'Random Forest', 'KNN', 'Logistic Regression after CV', 'Decision Tree after CV', 'Random Forest after CV', 'KNN after CV'] accuracies = accuracy['TF-IDF'] fig, ax = plt.subplots(figsize=(9, 5)) ax.barh(models, accuracies, color='skyblue') ax.set_title('Model Accuracies', fontsize=16) ax.set_xlabel('Accuracy', fontsize=14) ax.set_ylabel('Models', fontsize=14) ax.set_xlim(0, 1) ax.tick_params(axis='both', labelsize=12) for i, acc in enumerate(accuracies): ax.text(acc+0.01, i, f'{acc:.2f}', va='center', fontsize=12)

plt.show()

Neural network

[70]:
```
2+3
```

[70]:
```
5
```

Model Comparison and Selection

[71]:
```
X = data.drop('Churn', axis=1)  # Replace 'churn' with the actual column name for the target variable
y = data['Churn']
```

[72]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

[73]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
```

[74]:
```
logreg = LogisticRegression()
rf = RandomForestClassifier()
xgb_model = xgb.XGBClassifier()
```

[75]:
```
logreg.fit(X_train,

 y_train)
rf.fit(X_train, y_train)
xgb_model.fit(X_train, y_train)
```

[75]:
```
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_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

[76]:
```
logreg_preds = logreg.predict(X_test)
rf_preds = rf.predict(X_test)
xgb_preds = xgb_model.predict(X_test)
```

[77]:
```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score

logreg_acc = accuracy_score(y_test, logreg_preds)
logreg_prec = precision_score(y_test, logreg_preds)
logreg_rec = recall_score(y_test, logreg_preds)
logreg_f1 = f1_score(y_test, logreg_preds)
logreg_roc_auc = roc_auc_score(y_test, logreg_preds)

rf_acc = accuracy_score(y_test, rf_preds)
rf_prec = precision_score(y_test, rf_preds)
rf_rec = recall_score(y_test, rf_preds)
rf_f1 = f1_score(y_test, rf_preds)
rf_roc_auc = roc_auc_score(y_test, rf_preds)

xgb_acc = accuracy_score(y_test, xgb_preds)
xgb_prec = precision_score(y_test, xgb_preds)
xgb_rec = recall_score(y_test, xgb_preds)
xgb_f1 = f1_score(y_test, xgb_preds)
xgb_roc_auc = roc_auc_score(y_test, xgb_preds)
```

[78]:
```
print("Logistic Regression Performance:")
print("Accuracy:", logreg_acc)
print("Precision:", logreg_prec)
print("Recall:", logreg_rec)
print("F1-score:", logreg_f1)
print("ROC-AUC:", logreg_roc_auc)

print("\nRandom Forest Performance:")
print("Accuracy:", rf_acc)
print("Precision:", rf_prec)
print("Recall:", rf_rec)
print("F1-score:", rf_f1)
print("ROC-AUC:", rf_roc_auc)
```

```
print("\nXGBoost Performance:")
print("Accuracy:", xgb_acc)
print("Precision:", xgb_prec)
print("Recall:", xgb_rec)
print("F1-score:", xgb_f1)
print("ROC-AUC:", xgb_roc_auc)
```

```
Logistic Regression Performance:
Accuracy: 0.7068636317957994
Precision: 0.0
Recall: 0.0
F1-score: 0.0
ROC-AUC: 0.5

Random Forest Performance:
Accuracy: 0.7147020399959803
Precision: 0.6065573770491803
Recall: 0.07610558793280768
F1-score: 0.1352421565641182
ROC-AUC: 0.5278167973784027

XGBoost Performance:
Accuracy: 0.7115867751984725
Precision: 0.52297165200391
Recall: 0.1834076105587933
F1-score: 0.2715736040609137
ROC-AUC: 0.557015150175615
```

Ensemble Method Implementation

Voting Ensemble

[79]:
```
from sklearn.ensemble import VotingClassifier

# Initialize individual classifiers
classifier1 = LogisticRegression()
classifier2 = RandomForestClassifier()
classifier3 = XGBClassifier()

# Create a voting ensemble
voting_classifier = VotingClassifier(
    estimators=[('lr', classifier1), ('rf', classifier2), ('xgb', classifier3)],
    voting='hard'  # or 'soft' for soft voting
)

# Train the voting ensemble
voting_classifier.fit(X_train, y_train)

# Evaluate the ensemble's performance
accuracy = voting_classifier.score(X_test, y_test)
accuracy
```

[79]:
```
0.7159079489498543
```

Bagging Ensemble

[80]:
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier

# Initialize a base classifier
base_classifier = DecisionTreeClassifier()

# Create a bagging ensemble
bagging_classifier = BaggingClassifier(
    base_estimator=base_classifier,
    n_estimators=10  # Number of base classifiers
)

# Train the bagging ensemble
bagging_classifier.fit(X_train, y_train)

# Evaluate the ensemble's performance
accuracy = bagging_classifier.score(X_test, y_test)
accuracy
```

[80]:
```
0.7044518138880514
```

Boosting Ensemble

[81]:

```
import xgboost as xgb

# Define the boosting parameters
boosting_params = {
    'objective': 'binary:logistic',
    'n_estimators': 100,  # Number of boosting rounds
    'learning_rate': 0.1,
    'max_depth': 3
}

# Create a boosting ensemble
boosting_classifier = xgb.XGBClassifier(**boosting_params)

# Train the boosting ensemble
boosting_classifier.fit(X_train, y_train)
```

+ Code + Markdown