Customer Churn Predictive Analysis

Technology used:

- Python (pandas, seaborne, and scikit-learn)

The Purpose:

Why do we care about customer churn?

- Customer churn is an important matrix to look at because it costs more to acquire new customers than it does to retain existing customers.
- An increase in customer retention of just 5% can create at least a 25% increase in profit because returning customers will likely spend more on your company's products and services.
- It's cheaper to retain a customer than to acquire a new one
- By predicting customer churn, Telco companies can find ways to retain a customer. Either by giving out extra data, or a discount

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

1. Data processing:

I started off with reading the data, looking at the spread of it, the dimension. Then I moved on to checking duplicate rows and columns, as well as NA values. There 934 rows with missing value on the `Age` column, and I decided to drop the row containing missing values because it is only a small proportion of the dataset and it should not have a significant effect by removing them. I plot a pie chart and found the data has imbalanced class. Which will affect how I build and select the models later on.

In [3]:
```
dataframe = pd.read_excel('telus.xlsx')
df = dataframe
```

```
In [4]: df.head() #df dim is 86682 * 34
```

Out[4]:

	Cust_id	Age	LocID	GenID	RaceID	PkgID	CusCare_fla
0	6908fecdf3e8f8113a2350ca53ae229c075b5674	30.0	238	1	2	15	N
1	d1199c10898d9c6ef00ecb16507edbeaaec6e41a	29.0	265	1	2	6	N
2	2688962865d0325c5627b98caa792d8dbe57348e	29.0	240	1	1	8	N
3	a4e38c645638fb9776b3e68cfc9c3e22cf61843f	35.0	238	1	2	12	N
4	43f5e843b004f6590edb3da6f13843ee229fbede	38.0	234	2	1	13	N

5 rows × 34 columns

```
In [5]: df.describe()
```

Out[5]:

	Age	LocID	GenID	RaceID	PkgID	Num_calls_Cuscare
count	85748.000000	86682.000000	86682.000000	86682.000000	86682.000000	86682.(
mean	35.932873	238.686555	1.471747	1.620775	117.015032	0.(
std	11.962735	8.619000	0.499204	0.806007	102.087051	0.(
min	2.000000	231.000000	1.000000	1.000000	1.000000	0.(
25%	26.000000	233.000000	1.000000	1.000000	10.000000	0.(
50%	34.000000	237.000000	1.000000	1.000000	89.000000	0.(
75%	44.000000	242.000000	2.000000	2.000000	227.000000	0.(
max	96.000000	272.000000	2.000000	4.000000	300.000000	0.(

8 rows × 32 columns

Checking duplicates

```
In [4]: duplicated_rows = df.loc[df.Cust_id.duplicated()]
        duplicated_rows.count().sum()
```

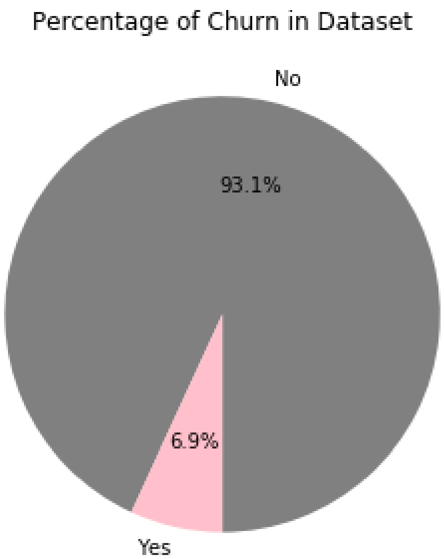
Out[4]: 0

Using a Pie chart to see the porpotion of customer churn

We see only 6.9% of customer churn. An imbalanced dataset should be treated. I will discuss this after feature selection.

In [6]:
```
# Data to plot
sizes = df['churnid'].value_counts(sort = 1)
colors = ["grey","pink"]
plt.figure(figsize=(5,5))

# Plot
plt.pie(sizes, labels=["No", "Yes"], colors=colors,
        autopct='%2.1f%%', shadow=False, startangle=270,)
plt.title('Percentage of Churn in Dataset')
plt.show()
```



Checking NA values

In [7]:
```python
#Checking NA values:
df.isnull()
#Counting # of NA values in each column:
df.isnull().sum()
```

Out[7]:
```
Cust_id                               0
Age                                 934
LocID                                 0
GenID                                 0
RaceID                                0
PkgID                                 0
CusCare_flag                          0
Num_calls_Cuscare                     0
BillCycle_ID                          0
Mean_Mthly_Paid                       0
Total_Bills                           0
Num_of_payments                       0
Timely_full_payments                  0
Delayed_Partial_Payments              0
Total_Paid_last6mth                   0
Mean_Mthly_Paid_last6mth              0
Total_Bills_last6mth                  0
Num_Payments_last6mth                 0
Timely_Full_Payments_last6mth         0
Delayed_Partial_Payments_last6mth     0
ttl_Data                              0
ttl_int_SMS                           0
ttl_int_Min                           0
avg_Data                              0
avg_int_SMS                           0
avg_int_Min                           0
ttl_Data_last6mth                     0
ttl_int_SMS_last6mth                  0
ttl_int_Min_last6mth                  0
avg_Data_last6mth                     0
avg_int_SMS_last6mth                  0
avg_int_Min_last6mth                  0
avg_online_days                       0
churnid                               0
dtype: int64
```

We see column Age has 934 missing values. Since it is only 934/86682 (1.08%) of the dataset. I have decided to drop the rows that contain missing Age.

In [8]:
```python
df = df.dropna();
#drop some unnecessary columns first
df.isnull().sum().sum()
#making sure we have 0 NAN values
```

Out[8]: 0

2. Feature selection / Feature importance:

Checking multicollinearity using VIF, the VIF score determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable. The Ideal VIF score is less than 10.

I also use random forest to evaluate feature importance. Here we see `avg_online_days` has almost 0.8 predictive strength, we see a huge difference between the top two features. I wanted to look at other predictors too so I removed average online days and did another random forest search. Now we have a rough idea of what variables we want to use for our model. We need to check for multicollinearity using VIF score above.

Checking multicollinearity using VIF

VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable.

- VIF starts at 1 and has no upper limit
- VIF = 1, no correlation between the independent variable and the other variables
- VIF exceeding 10 indicates high multicollinearity between this independent variable and the others

$$VIF = \frac{1}{(1 - R^2)}$$

In [9]:
```
from statsmodels.stats.outliers_influence import variance_inflation_fact
or

def calc_vif(X):

    # Calculating VIF
    vif = pd.DataFrame()
    vif["variables"] = X.columns
    vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(
X.shape[1])]

    return(vif)
```

Getting the VIF score, here we see there's quite a few columns that are suspected to multicollinearity, let's work on them. Focus on the variables that have VIF value = inf. They are:

- Total_Bills_last6mth, Num_Payments_last6mth, Timely_Full_Payments_last6mth, Delayed_Partial_Payments_last6mth
- Total_Bills, Num_of_payments, Timely_full_payments, Delayed_Partial_Payments

In [10]:
```
df_vif = df.drop(columns = ["CusCare_flag", "Cust_id"])
X = df_vif.iloc[:,:-1]
calc_vif(X)
```

```
C:\Users\ChunLin\Anaconda3\lib\site-packages\statsmodels\regression\lin
ear_model.py:1638: RuntimeWarning: invalid value encountered in double_
scalars
  return 1 - self.ssr/self.uncentered_tss
C:\Users\ChunLin\Anaconda3\lib\site-packages\statsmodels\stats\outliers
_influence.py:185: RuntimeWarning: divide by zero encountered in double
_scalars
  vif = 1. / (1. - r_squared_i)
```

Out[10]:

	variables	VIF
0	Age	10.588382
1	LocID	165.195268
2	GenID	9.813539
3	RaceID	5.394203
4	PkgID	4.953958
5	Num_calls_Cuscare	NaN
6	BillCycle_ID	4.882028
7	Mean_Mthly_Paid	45.917125
8	Total_Bills	inf
9	Num_of_payments	inf
10	Timely_full_payments	inf
11	Delayed_Partial_Payments	inf
12	Total_Paid_last6mth	131.400675
13	Mean_Mthly_Paid_last6mth	143.989726
14	Total_Bills_last6mth	inf
15	Num_Payments_last6mth	inf
16	Timely_Full_Payments_last6mth	inf
17	Delayed_Partial_Payments_last6mth	inf
18	ttl_Data	13.874809
19	ttl_int_SMS	9.373152
20	ttl_int_Min	6.798250
21	avg_Data	25.538535
22	avg_int_SMS	23.163095
23	avg_int_Min	13.265017
24	ttl_Data_last6mth	7.283044
25	ttl_int_SMS_last6mth	2019.157026
26	ttl_int_Min_last6mth	97.105572
27	avg_Data_last6mth	26.078185
28	avg_int_SMS_last6mth	2046.747496
29	avg_int_Min_last6mth	108.034105
30	avg_online_days	127.184015

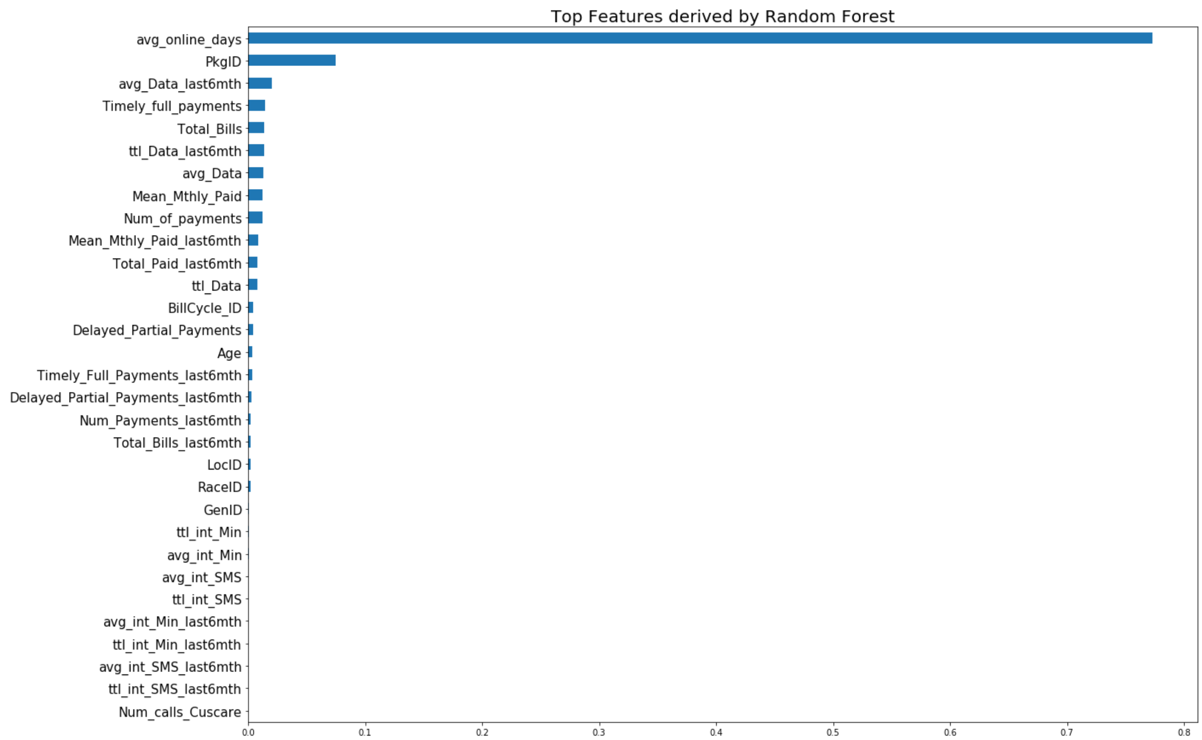
Feature Selection with Random Forest

In [11]:
```
from sklearn.ensemble import RandomForestClassifier
```

`avg_online_days` is a very strong predictor, which may be good or bad. We need to look into it more

In [12]:
```
X, y = df_vif.drop('churnid',axis=1), df_vif[['churnid']]
rfc = RandomForestClassifier(random_state=0, n_estimators=100)
model = rfc.fit(X, y.values.ravel())
(pd.Series(model.feature_importances_, index=X.columns)
   .nlargest(47)
   .plot(kind='barh', figsize=[20,15])
   .invert_yaxis())
plt.yticks(size=15)
plt.title('Top Features derived by Random Forest', size=20)
```

Out[12]:
```
Text(0.5, 1.0, 'Top Features derived by Random Forest')
```



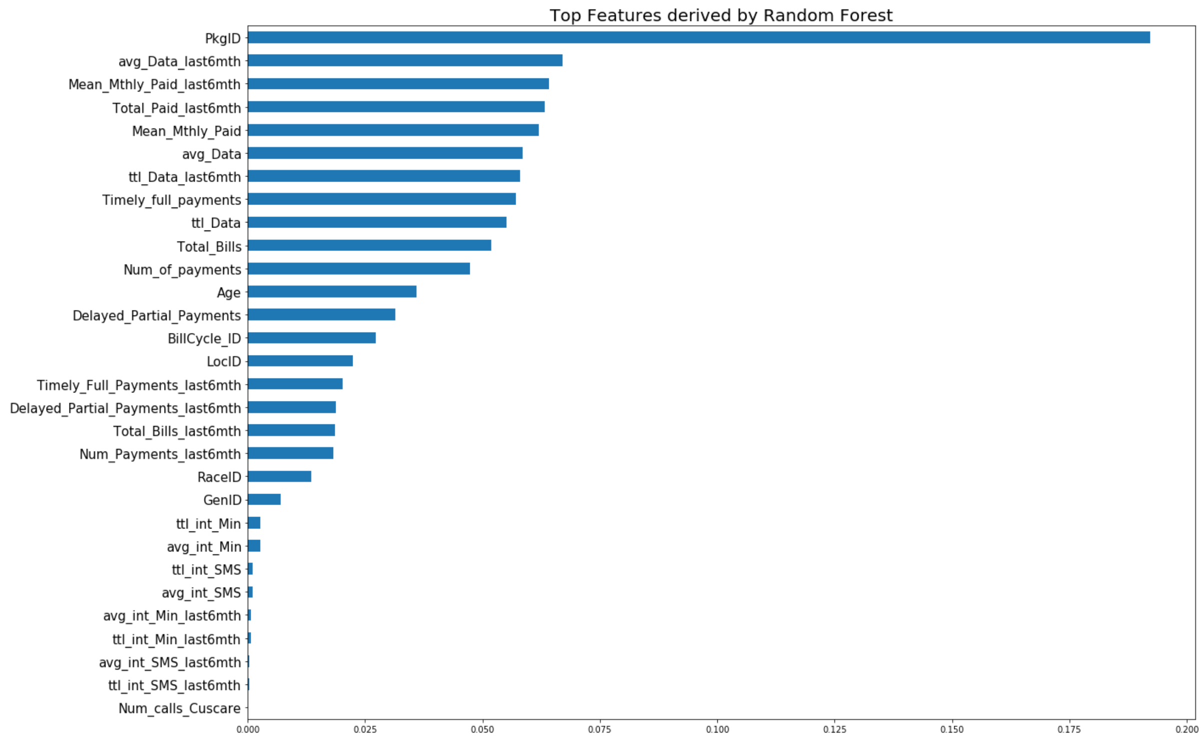
removed `avg_online_days`

In [13]:

```
X2, y2 = df_vif.drop(['churnid', 'avg_online_days'],axis=1), df_vif[['ch
urnid']]

rfc2 = RandomForestClassifier(random_state=0, n_estimators=100)
model2 = rfc2.fit(X2, y2.values.ravel())
(pd.Series(model2.feature_importances_, index=X2.columns)
   .nlargest(47)
   .plot(kind='barh', figsize=[20,15])
   .invert_yaxis())
plt.yticks(size=15)
plt.title('Top Features derived by Random Forest', size=20)
```

Out[13]: `Text(0.5, 1.0, 'Top Features derived by Random Forest')`



Now we have a rough idea of what variables we want to use for our model. We need to check for multicollinearity using VIF score above.

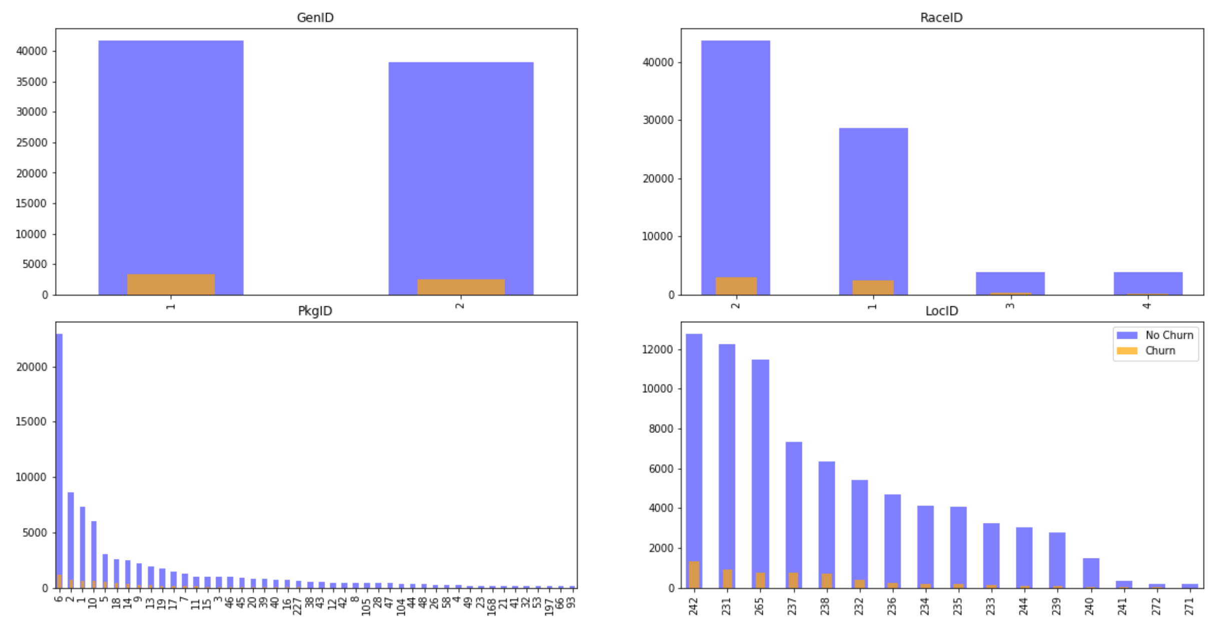
Visualize some of the variables:

I plotted some bar graphs to show categorical variables with respect to churn id. Here we see distribution of Gender, Race Pkgid and location are pretty similar across churn and non-churn customer, as well as how imbalanced the two classes is.

categorical variables

In [15]:

```
categorical_features = ['GenID', 'RaceID', 'PkgID', 'LocID']
ROWS, COLS = 2, 2
fig, ax = plt.subplots(ROWS, COLS, figsize=(20, 10) )
row, col = 0, 0
for i, categorical_feature in enumerate(categorical_features):
    if col == COLS - 1: row += 1
    col = i % COLS
    df[df.churnid==0][categorical_feature].value_counts().plot(kind = 'bar', width=.5, ax=ax[row, col], color='blue', alpha=0.5).set_title(categorical_feature)
    df[df.churnid==1][categorical_feature].value_counts().plot(kind = 'bar', width=.3, ax=ax[row, col], color='orange', alpha=0.7).set_title(categorical_feature)
    plt.legend(['No Churn', 'Churn'])
    fig.subplots_adjust(hspace=0.1)
```



- From the above plots, we can tell that GenID (gender) customer are equally likely to churn because the ratio of churn and non-churn are the same.
- LocID (location id) the churn and non-churn rate vary from 1% to 10%, which means area code has a correlation with churn rate in my opinion therefore I will be including LocID for our initial model building
- RaceID (race) the churn and non-churn rate are insignificant between id 1&2 and 3&4
- PkgID (device plan) plan #6 has the highest churn rate which may be useful to include in the model

We can use Groupby to summarize the counts in each category on churnid

I used groupby to summarize the counts in each category on churn id. We do see a difference in 3%-9% in Race and Location, but difference in Gender is rather insignificant.

In [26]:

```
print(df.groupby('LocID')['churnid'].value_counts())
```

In [27]:
```
print(df.groupby('GenID')['churnid'].value_counts())
#Churn rate between Gen 1 & 2 doesn't differ a lot (only a 1.15% differe
nce)
#therefore I will likely be removing GenID as a variable
```

In [28]:
```
print(df.groupby('RaceID')['churnid'].value_counts())
#the percentage churn goes as low as 3% and as high as 9%
```

In [29]:
```
print(df.groupby('CusCare_flag')['churnid'].value_counts())
#only 'No' response is recorded, not useful to include in the model
```

In [30]:
```
print(df.groupby('Num_calls_Cuscare')['churnid'].value_counts())
#only 0 calls are recorded, not useful to include in the model, drop the
column
```

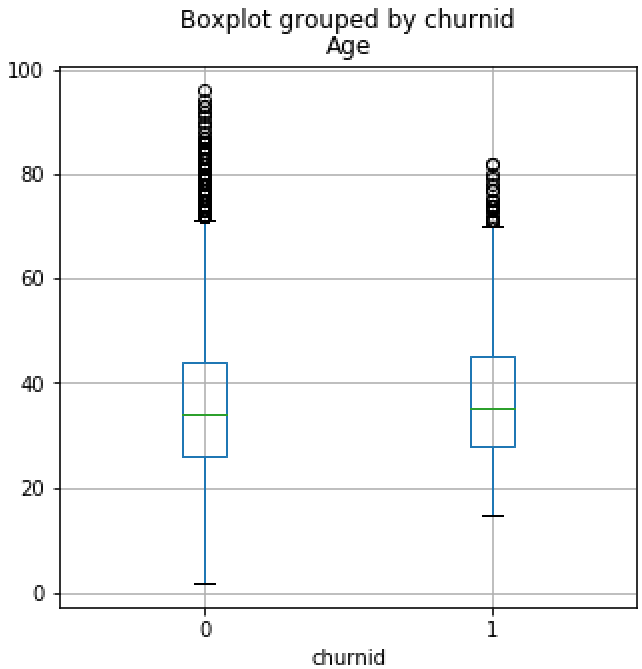
summarize all columns churnid = 0 #starting from row 5945 and all columns churnid = 1

In [454]:
```
# churn1 = df[0:5945] #5945 rows × 31 columns
# churn0 = df[5945:] #79803 rows × 31 columns
# churn1.describe()
# churn0.describe()
```

Exmine `Age`

questionable churn status, having a lot of outliers on churnid = 0, mostly seniors over 65 years old. possibly passed away (meaning no action of 'churn'), account simply gets terminated? More clarifications need, but let's include that to our model.

```
In [16]: boxplot = df.boxplot(column=['Age'], by='churnid', figsize=(5,5))
```



Examine `Timely_full_payments`, `Num_of_payments`, `Delayed_Partial_Payments`

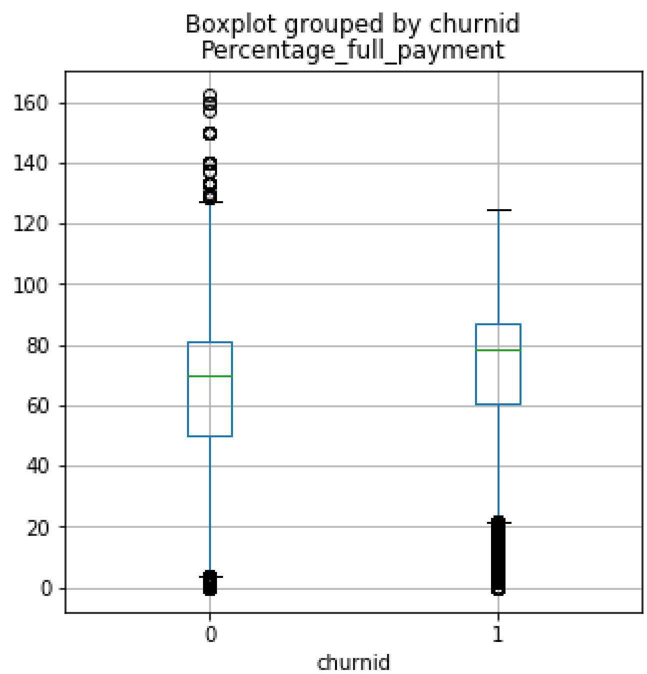
I examined `timely_full_payments`, `num_of_payments`, `delayed_partial_payments` and decided to combine them to a single predictor As I noticed the addition of timely full payments and delayed partical payments is just number of payments. Instead of using all three variables, I combined the three variables into one new variable called `percentage_full_payment`

I did the same thing with the last6months and created a new variable called `percentage_full_payment_last6mth` I checked all the variables that I suspected multicollinearity and confirm them with scatter plots and correlation matrix. I was able to find a lot of variables that are highly correlated suck as mean monthly paid and total paid last 6 month and mean monthly paid last 6 months. I ended up keep the variable that have a higher importance score from random forest search

In [17]:
```
full_payment_percentage = (df.Num_of_payments - df.Delayed_Partial_Payme
nts) / df.Num_of_payments
df['Percentage_full_payment'] = full_payment_percentage*100
boxplot = df.boxplot(column=['Percentage_full_payment'], by='churnid',fi
gsize=(5,5))
```

```
C:\Users\ChunLin\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```



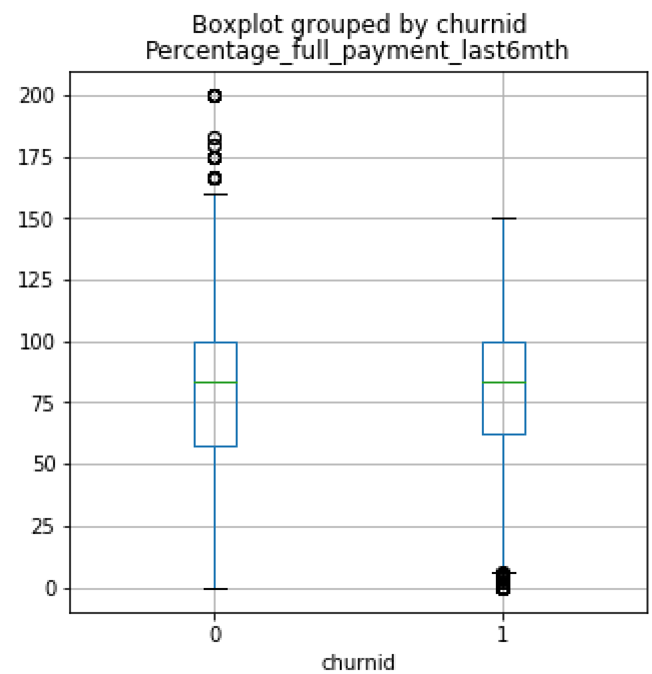
Examine `Timely_full_payments_last6mth` , `Num_of_payments_last6mth` , `Total_Bills_last6mth` , `Delayed_Partial_Payments_last6mth` ¶

Similar to before, I created another variable named `Percentage_full_payment_last6mth` from `Timely_full_payments_last6mth` and `Num_of_payments_last6mth` , and added that new column to the dataframe. Now instead of using all three columns `Num_of_payments_last6mth` , Timely_full_payments_last6mth, and `Delayed_Partial_Payments_last6mth` , we only care about the percentage of timely full payments made by the customer in the last 6 months.

In [18]:
```
full_payment_percentage_last6mth = (df.Num_Payments_last6mth-df.Delayed_
Partial_Payments_last6mth) / df.Num_Payments_last6mth
df['Percentage_full_payment_last6mth'] = full_payment_percentage_last6mt
h *100
boxplot = df.boxplot(column=['Percentage_full_payment_last6mth'], by='ch
urnid',figsize=(5,5))
```

```
C:\Users\ChunLin\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: S
ettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```



Treating Multicollinearity with VIF and Pairwise Correlation

1. Drop redundant variables or the one with high VIF — this may again lead to loss of information
2. Come up with interaction terms or polynomial terms and drop the redundant features
3. Use Principal component analysis (also a dimensionality reduction technique) which is a statistical procedure to convert a set of possibly correlated predictors into a set of linearly uncorrelated variables.

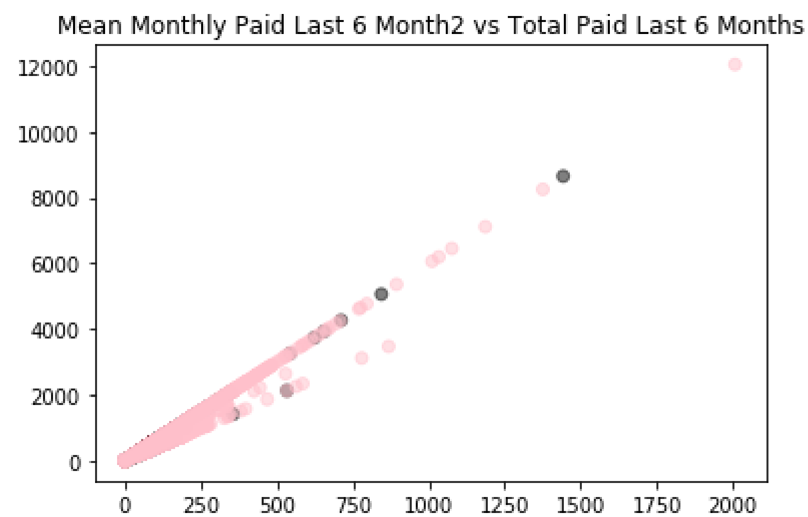
Examine `Mean_Mthly_Paid_last6mth` , `Total_Paid_last6mth` and `Mean_Mthly_Paid`

`Mean_Mthly_Paid_last6mth` and `Total_Paid_last6mth` has a very high positive correlation. Including both variables in the model will cause multicolinearity. Hence I will be removing `Total_Paid_last6mth`

In [19]:
```
def pltcolor(lst):
    cols=[]
    for l in lst:
        if l==1:
            cols.append('black')
        else:
            cols.append('pink')
    return cols
# Create the colors list using the function above
cols=pltcolor(df.churnid)
```

In [20]:
```
plt.scatter(df.Mean_Mthly_Paid_last6mth, df.Total_Paid_last6mth, c = col
s, alpha = 0.5)
plt.title("Mean Monthly Paid Last 6 Month2 vs Total Paid Last 6 Months")
```

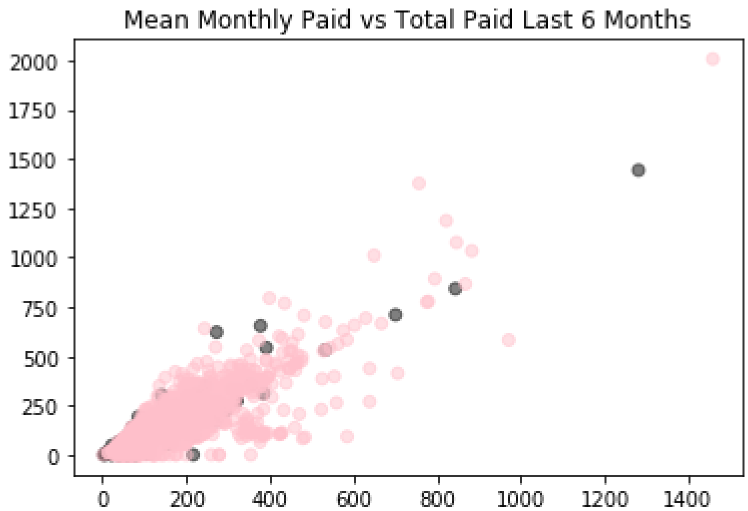
Out[20]:
```
Text(0.5, 1.0, 'Mean Monthly Paid Last 6 Month2 vs Total Paid Last 6 Mo
nths')
```



`Mean_Mthly_Paid` and `Total_Paid_last6mth` has a less positive correlation. However, It is highly correlated with `Total_Paid_last6mth` and `Mean_Mthly_Paid_last6mth` with correlation 0.852621 and 0.884625 respectively. I will likely keep `Mean_mthly_Paid_last6mth` because it's a stronger feature.

In [21]:
```python
plt.scatter(df.Mean_Mthly_Paid, df.Mean_Mthly_Paid_last6mth, c = cols, a
lpha = 0.5)
plt.title("Mean Monthly Paid vs Total Paid Last 6 Months")
```

Out[21]: `Text(0.5, 1.0, 'Mean Monthly Paid vs Total Paid Last 6 Months')`



Examine `ttl_Data`, `avg_Data` , `ttl_int_SMS`, `avg_int_SMS` , `ttl_int_Min`, `avg_int_Min`

- `ttl_int_SMS` and `avg_int_SM`
- `ttl_int_Min` and `avg_int_Min`

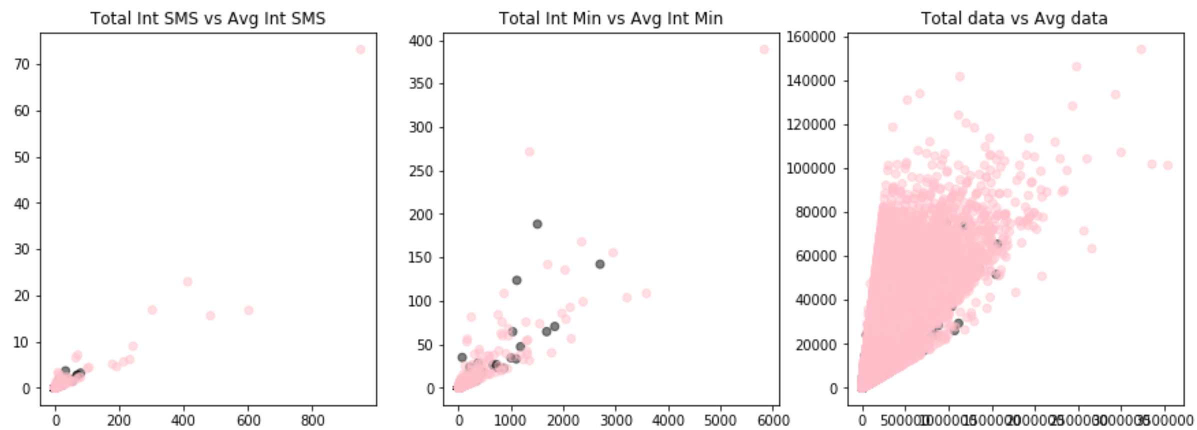
Both pairs appear to be postive correlated. The correlation matrix above shows `ttl_Data` and `avg_Data` has a correlation of 0.773245. `ttl_Data` and `ttl_Data_last6mth` has a correlation of 0.848406.

- `ttl_Data` and `avg_Data`

I'm surprised that the `avg_Data` doesn't have a strong positive correlation with `ttl_Data`

```
In [22]: fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,5))
ax1.scatter(df.ttl_int_SMS, df.avg_int_SMS, c = cols, alpha = 0.5)
ax2.scatter(df.ttl_int_Min, df.avg_int_Min, c = cols, alpha = 0.5)
ax3.scatter(df.ttl_Data, df.avg_Data, c = cols, alpha = 0.5)
ax1.set_title('Total Int SMS vs Avg Int SMS')
ax2.set_title('Total Int Min vs Avg Int Min')
ax3.set_title('Total data vs Avg data')
```

```
Out[22]: Text(0.5, 1.0, 'Total data vs Avg data')
```



Exmine `avg_Data`, `avg_DAta_last6mth`, `ttl_Data_last6mth`

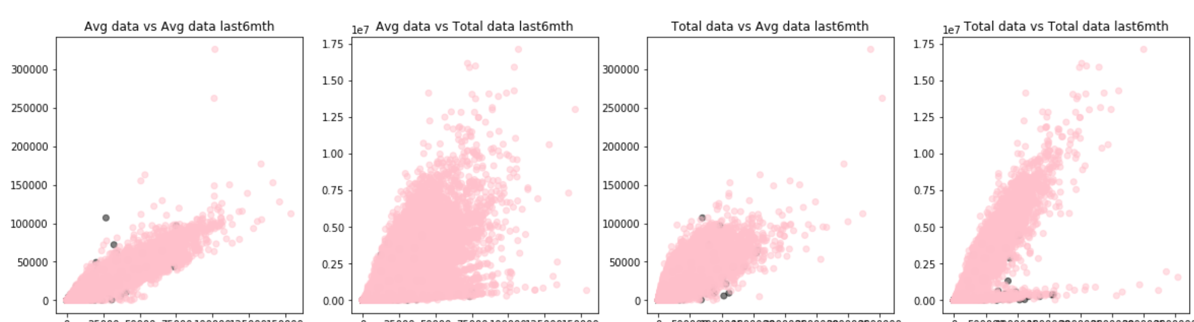
```
In [23]: fig, (ax1, ax2, ax3, ax4) = plt.subplots(1, 4, figsize=(20,5))
ax1.scatter(df.avg_Data, df.avg_Data_last6mth, c = cols, alpha = 0.5)
ax2.scatter(df.avg_Data, df.ttl_Data_last6mth, c = cols, alpha = 0.5)

ax3.scatter(df.ttl_Data, df.avg_Data_last6mth, c = cols, alpha = 0.5)
ax4.scatter(df.ttl_Data, df.ttl_Data_last6mth, c = cols, alpha = 0.5)

ax1.set_title('Avg data vs Avg data last6mth')
ax2.set_title('Avg data vs Total data last6mth')

ax3.set_title('Total data vs Avg data last6mth')
ax4.set_title('Total data vs Total data last6mth')
```

```
Out[23]: Text(0.5, 1.0, 'Total data vs Total data last6mth')
```



Examine `ttl_Data_last6mth`, `ttl_int_SMS_last6mth`, `ttl_int_Min_last6mth`, `avg_Data_last6mth`, `avg_int_SMS_last6mth`, `avg_int_Min_last6mth`

- `ttl_int_SMS_last6mth` and `avg_int_SMS_last6mth`,
- `ttl_int_Min_last6mth` and `avg_int_Min_last6mth`

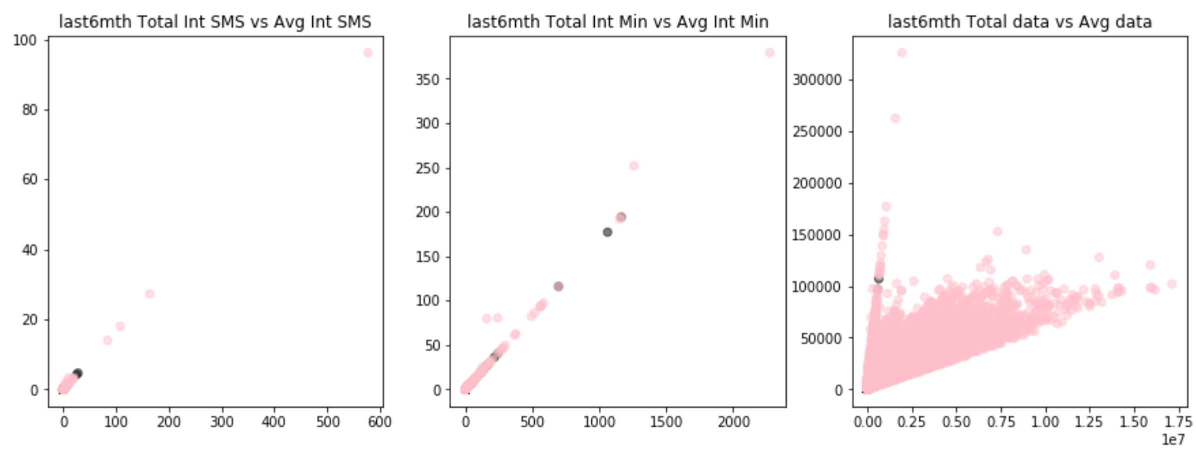
Both pairs appear to be highly postive correlated.

- `ttl_Data_last6mth` and `avg_Data_last6mth`

I'm surprised that the `avg_Data_last6mth` doesn't have a strong positive correlation with `ttl_Data_last6mth`.

In [24]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1, 3, figsize=(15,5))
ax1.scatter(df.ttl_int_SMS_last6mth, df.avg_int_SMS_last6mth, c = cols, alpha = 0.5)
ax2.scatter(df.ttl_int_Min_last6mth, df.avg_int_Min_last6mth, c = cols, alpha = 0.5)
ax3.scatter(df.ttl_Data_last6mth, df.avg_Data_last6mth, c = cols, alpha = 0.5)
ax1.set_title('last6mth Total Int SMS vs Avg Int SMS')
ax2.set_title('last6mth Total Int Min vs Avg Int Min')
ax3.set_title('last6mth Total data vs Avg data')
```

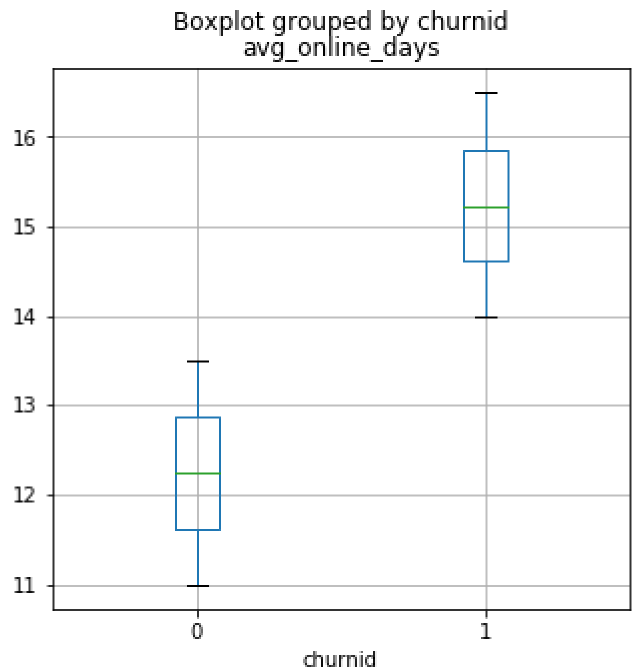
Out[24]:
```
Text(0.5, 1.0, 'last6mth Total data vs Avg data')
```



Examine `avg_online_days`

From the box plot we see there's a significant different in average online days between the churn and non-churn customers.

In [25]:
```
boxplot = df.boxplot(column=['avg_online_days'], by='churnid',figsize=(5
,5))
```



Removing columns

To start off. I will be dropping the 9 columns that are the worst features derived by random forest

In [26]:
```
df_reduced = df.drop(columns = ['BillCycle_ID', 'CusCare_flag', 'Cust_i
d',
                                'Num_calls_Cuscare', 'ttl_int_SMS_last6m
th', 'avg_int_SMS_last6mth',
                                'ttl_int_Min_last6mth', 'avg_int_Min_las
t6mth', 'avg_int_SMS',
                                'ttl_int_SMS', 'avg_int_Min', 'ttl_int_M
in'])
```

Then I'll be dropping columns that are causing multicollinearity

In [27]:
```
df_reduced = df_reduced.drop(columns = ['Timely_full_payments', 'Num_of_
payments', 'Delayed_Partial_Payments',
                                        'Timely_Full_Payments_last6mth',
'Delayed_Partial_Payments_last6mth',
                                        'Num_Payments_last6mth', 'avg_Dat
a', 'ttl_Data', 'GenID', 'RaceID',
                                        'Total_Paid_last6mth', 'Percentag
e_full_payment_last6mth',
                                        'Mean_Mthly_Paid_last6mth'])
```

Check for missing values, see which rows have missing value.

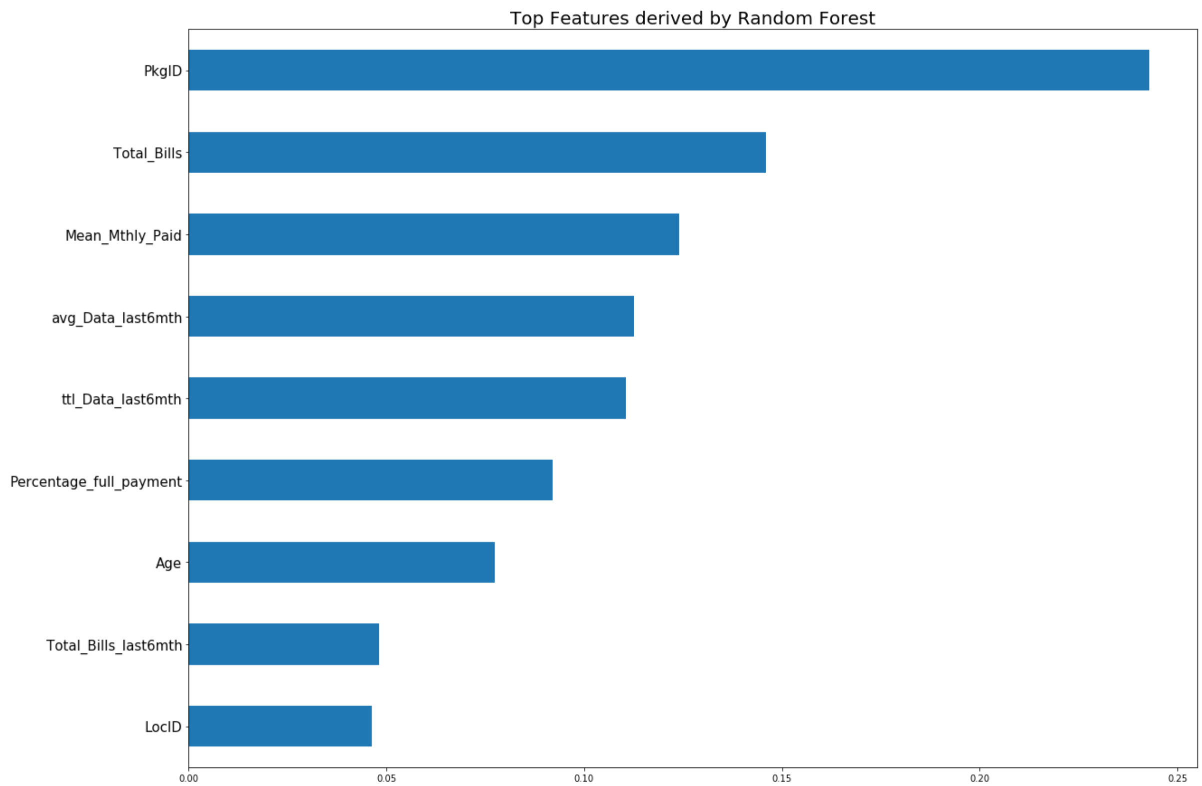
column Percentage_full_payment_last6mth has missing values. After looking at it the null values come from both denominator and neumerators = 0.

In [28]:
```
df_reduced = df_reduced.fillna(0)
```

Take a look at the Feature importance chart again

In [29]:
```
X, y = df_reduced.drop(['churnid', 'avg_online_days'],axis=1), df_reduce
d[['churnid']]
rfc = RandomForestClassifier(random_state=0, n_estimators=100)
model = rfc.fit(X, y.values.ravel())
(pd.Series(model.feature_importances_, index=X.columns)
   .nlargest(47)
   .plot(kind='barh', figsize=[20,15])
   .invert_yaxis())
plt.yticks(size=15)
plt.title('Top Features derived by Random Forest', size=20)
```

Out[29]:
```
Text(0.5, 1.0, 'Top Features derived by Random Forest')
```



Reordering columns: move churnid column to last column

In [30]:
```
col_name="churnid"
first_col = df_reduced.pop(col_name)
```

In [31]:
```
df_reduced.insert(10, col_name, first_col)
```

Checking VIF one more time

- LocID has high VIF, but looking at the correlation matrix, the variable does not have high correlation with any other variables. I will keep LocID for now.
- Mean_mthly_Paid and Mean_Mthly_Paid_last6mth are highly positive correlated. I will create two dataframes one keeps Mean_Mthly_Paid and one keeps Mean_Mthly_Paid_last6mth. Same thing goes with Percentage_full_payment and Percentage_full_payment_last6mth.
- Similar situation with prefix ttl *and avg* variables. I will be keeping one set with ttl *variables and one with avg* variables.

In [32]:
```
X_new = df_reduced.iloc[:,:-1]
calc_vif(X_new)
```

Out[32]:

	variables	VIF
0	Age	10.331538
1	LocID	159.692274
2	PkgID	4.396574
3	Mean_Mthly_Paid	11.514922
4	Total_Bills	5.921847
5	Total_Bills_last6mth	7.944735
6	ttl_Data_last6mth	4.401605
7	avg_Data_last6mth	6.327977
8	avg_online_days	126.800510
9	Percentage_full_payment	12.282893

Export df_reduced as csv

In [33]:
```
df_reduced.to_csv('telus_reduced_sept14')
```

End of data cleaning

3. Model building

Treating imbalanced data

Since we have An imbalanced dataset 5971 out of 85748 (0.0696%) rows are churnid = 1. The rest of the data is churnid = 0. To deal with an imbalanced class we can do the following:

1. If we keep data as is, use different evaluation matrices like F1 score, AUC, Precision/Specificity, Recall/Sensitivity
2. If we want to treat the data by resampling:

- Resample the training set: Over-sampling the minority class
- Use K-fold Cross-Validation in the right way
- Ensemble different resampled datasets: The problem is that out-of-the-box classifiers like logistic regression or random forest tend to generalize by discarding the rare class. One easy best practice is building n models that use all the samples of the rare class and n-differing samples of the abundant class. Given that you want to ensemble 10 models, you would keep e.g. the 1.000 cases of the rare class and randomly sample 10.000 cases of the abundant class. Then you just split the 10.000 cases in 10 chunks and train 10 different models.

I will be employing three major techniques:

1. SMOTE (Synthetic Minority Oversampling Technique) to perform oversample the minority class
2. Cross Validation with train/test split
3. Cross Validation with stratified sampling. Stratified k fold will ensure that the percentages of each class in your entire data will be the same (or very close to) within each individual fold.

In [8]:
```
from sklearn import model_selection, preprocessing
from sklearn.datasets import make_classification
from sklearn.model_selection import StratifiedKFold, train_test_split, cross_validate, cross_val_predict, cross_val_score, StratifiedKFold, KFold, LeaveOneOut
from imblearn.over_sampling import SMOTE
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report, f1_score, recall_score
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import make_scorer, accuracy_score, precision_score, recall_score, f1_score
```

In [5]:
```
df = pd.read_csv('telus_reduced_sept14')
```

In [6]:
```
df
```

Out[6]:

	Unnamed: 0	Age	LocID	PkgID	Mean_Mthly_Paid	Total_Bills	Total_Bills_last6mth	ttl_Data
0	0	30.0	238	15	109.20	4	4	
1	1	29.0	265	6	72.11	16	4	
2	2	29.0	240	8	384.42	44	8	
3	3	35.0	238	12	245.07	35	4	
4	4	38.0	234	13	54.61	3	3	
...	...	...	...	...	...	...	...	
85743	86677	43.0	237	11	128.98	6	6	
85744	86678	47.0	239	273	60.35	7	5	
85745	86679	31.0	242	68	80.17	26	5	
85746	86680	25.0	242	6	77.65	43	7	
85747	86681	48.0	242	10	96.02	7	7	

85748 rows × 12 columns

In [9]:
```
#shuffle the rows
df = df.sample(frac=1).reset_index(drop=True)
#normalize the dataframe

x = df.values #returns a numpy array
min_max_scaler = preprocessing.MinMaxScaler()
x_scaled = min_max_scaler.fit_transform(x)
df = pd.DataFrame(x_scaled, columns=df.columns)
```

In [10]:
```
df = df.drop(columns = ['Unnamed: 0'])
```

Previously we talked about how `avg_online_days` is a very strong predictor. In my opinion, it is too overpowering that the whole model only depends on this predictor.

In [11]:
```
X = df.iloc[:, 0:10]
y = df.iloc[:,-1]
```

In [21]:
```
smote = SMOTE(random_state=10)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=5)
X_train_res, y_train_res = smote.fit_sample(X_train, y_train)

X2_train, X2_test, y2_train, y2_test = train_test_split(X_2, y_2, test_s
ize=0.3, random_state=7)
X2_train_res, y2_train_res = smote.fit_sample(X2_train, y2_train)
```

Logistic Regression

In [13]:
```
scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score)}
Num_folds = 10
kfold = KFold(n_splits = Num_folds)
strat_kfold = StratifiedKFold(n_splits = Num_folds)
lr_model=LogisticRegression(max_iter=10000)
```

10 folds CV

In [14]:
```
kfold_lr_results = model_selection.cross_validate(estimator=lr_model,
                                                  X=X,
                                                  y=y,
                                                  cv=kfold,
                                                  scoring=scoring)

print('test accuracy: ' + str(np.mean(kfold_lr_results['test_accuracy'
])))
print('test_precision: ' + str(np.mean(kfold_lr_results['test_precision'
])))
print('test_recall: ' + str(np.mean(kfold_lr_results['test_recall'])))
print('test_f1_score: ' + str(np.mean(kfold_lr_results['test_f1_score'
])))
```

```
test accuracy: 0.9998483937811853
test_precision: 1.0
test_recall: 0.9977933737672384
test_f1_score: 0.9988948622511234
```

Stratified sampling

In [16]:
```
strat_kfold_lr_results = model_selection.cross_validate(estimator=lr_mod
el,
                                                         X=X,
                                                         y=y,
                                                         cv=strat_kfold,
                                                         scoring=scoring)

print('test accuracy: ' + str(np.mean(strat_kfold_lr_results['test_accur
acy'])))
print('test_precision: ' + str(np.mean(strat_kfold_lr_results['test_prec
ision'])))
print('test_recall: ' + str(np.mean(strat_kfold_lr_results['test_recall'
])))
print('test_f1_score: ' + str(np.mean(strat_kfold_lr_results['test_f1_sc
ore'])))
```

```
test accuracy: 0.9998483937811853
test_precision: 1.0
test_recall: 0.9978131454602043
test_f1_score: 0.9989048004669889
```

SMOTE

In [23]:
```
smote_lr = lr_model.fit(X_train_res, y_train_res)
smote_lr_pred = smote_lr.predict(X_test)

print('test accuracy: ' + str(accuracy_score(y_test, smote_lr_pred)))
print('test_precision: ' + str(precision_score(y_test, smote_lr_pred)))
print('test_recall: ' + str(recall_score(y_test, smote_lr_pred)))
print('test_f1_score: ' + str(f1_score(y_test, smote_lr_pred)))
```

```
test accuracy: 1.0
test_precision: 1.0
test_recall: 1.0
test_f1_score: 1.0
```

Strafied sampling and Random Forest

In [18]:
```
strat_kfold = StratifiedKFold(n_splits=10)
scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score)}

model=RandomForestClassifier(n_estimators=50)

results = model_selection.cross_validate(estimator=model,
                                         X=X,
                                         y=y,
                                         cv=strat_kfold,
                                         scoring=scoring)

print('test_accuracy: ' + str(np.mean(results['test_accuracy'])))
print('test_precision: ' + str(np.mean(results['test_precision'])))
print('test_recall: ' + str(np.mean(results['test_recall'])))
print('test_f1_score: ' + str(np.mean(results['test_f1_score'])))
```

```
test_accuracy: 1.0
test_precision: 1.0
test_recall: 1.0
test_f1_score: 1.0
```

Observation:

including `avg_online_days` can give us a very good prediction and can be overpowering. However there's no such thing as a perfect model like we saw above. We need to consider other predictors as well, solely depending on one variable can be dangerous. If we some data does not follow the typical trend of such variable, we will incurr loss in accuracy, precision, recall, f1_score.

Removing `avg_online_days`

In [25]:
```
df_2 = df.drop(columns = ['avg_online_days'])
X_2 = df_2.iloc[:, 0:9]
y_2 = df_2.iloc[:,-1]

scoring = {'accuracy' : make_scorer(accuracy_score),
           'precision' : make_scorer(precision_score),
           'recall' : make_scorer(recall_score),
           'f1_score' : make_scorer(f1_score)}

kfold = model_selection.KFold(n_splits=10, random_state=42, shuffle = Tr
ue)
rfc_model=RandomForestClassifier(n_estimators=50)
```

SMOTE with Logistic regression

In [22]:
```
smote2_lr = lr_model.fit(X2_train_res, y2_train_res)
smote2_lr_pred = smote2_lr.predict(X2_test)

print('test_accuracy: ' + str(accuracy_score(y2_test, smote2_lr_pred)))
print('test_precision: ' + str(precision_score(y2_test, smote2_lr_pred
)))
print('test_recall: ' + str(recall_score(y2_test, smote2_lr_pred)))
print('test_f1_score: ' + str(f1_score(y2_test, smote2_lr_pred)))
```

```
test accuracy: 0.6818270165208941
test_precision: 0.17275985663082438
test_recall: 0.9403567447045708
test_f1_score: 0.29189376243619697
```

Stratified sampling and Random Forest (to combat imbalance class distribution)

In [26]:
```
results = model_selection.cross_validate(estimator=rfc_model,
                                         X=X_2,
                                         y=y_2,
                                         cv=strat_kfold,
                                         scoring=scoring)

print('test accuracy: ' + str(np.mean(results['test_accuracy'])))
print('test_precision: ' + str(np.mean(results['test_precision'])))
print('test_recall: ' + str(np.mean(results['test_recall'])))
print('test_f1_score: ' + str(np.mean(results['test_f1_score'])))
```

```
test accuracy: 0.9485352435085801
test_precision: 0.7313888251637811
test_recall: 0.40756557168321866
test_f1_score: 0.523287905776333
```

4. Result / Conclusion:

Choosing with a model with a almost perfect score might be ideal in a perfect world. However, in real life we cannot ensure every data that comes in will satisfy that one predictor that's overpowering the rest.

The model should be general enough but at the same time catches important trends in the data. Therefore my final is going to be the Random forest with stratified k fold (k=10) model.

There are a few things we can do to improve our simple model. (increase f1 and recall)

1. Find a way to include `avg_online_days` but weigh down the predictive strength in order to compliment the rest of the predictors.
2. Ensemble (employing multiple different models to vote on the churn status)
3. GridSearchCV for Hyperparameter Tuning
4. Try different sampling techniques, perhaps try undersampling