Telecom Churn: Logistic Regression with PCA

With 21 predictor variables, we need to predict whether a particular customer will switch to another telecom provider or not. In telecom terminology, customer attrition is referred to as 'churn'.

Importing and Merging Data

In [144]:
```
# Importing Pandas and NumPy
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [145]:
```
# Importing all datasets
churn_data = pd.read_csv("churn_data.csv")
customer_data = pd.read_csv("customer_data.csv")
internet_data = pd.read_csv("internet_data.csv")
```

In [146]:
```
print(len(churn_data))
print(len(customer_data))
print(len(internet_data))
```

```
7043
7043
7043
```

In [147]:
```
#Merging on 'customerID'
df_1 = pd.merge(churn_data, customer_data, how='inner', on='customerID')
```

In [148]:
```
#Final dataframe with all predictor variables
telecom = pd.merge(df_1, internet_data, how='inner', on='customerID')
```

Let's understand the structure of our dataframe

In [149]:
```
# Let's see the head of our master dataset
telecom.head()
```

Out[149]:

	customerID	tenure	PhoneService	Contract	PaperlessBilling	PaymentMethod	Montl
0	7590-VHVEG	1	No	Month-to-month	Yes	Electronic check	29.85
1	5575-GNVDE	34	Yes	One year	No	Mailed check	56.95
2	3668-QPYBK	2	Yes	Month-to-month	Yes	Mailed check	53.85
3	7795-CFOCW	45	No	One year	No	Bank transfer (automatic)	42.30
4	9237-HQITU	2	Yes	Month-to-month	Yes	Electronic check	70.70

5 rows × 21 columns

In [150]:
```
telecom.describe()
```

Out[150]:

	tenure	MonthlyCharges	SeniorCitizen
count	7043.000000	7043.000000	7043.000000
mean	32.371149	64.761692	0.162147
std	24.559481	30.090047	0.368612
min	0.000000	18.250000	0.000000
25%	9.000000	35.500000	0.000000
50%	29.000000	70.350000	0.000000
75%	55.000000	89.850000	0.000000
max	72.000000	118.750000	1.000000

Data Preparation

In [151]:
```
# Converting Yes to 1 and No to 0
telecom['PhoneService'] = telecom['PhoneService'].map({'Yes': 1, 'No': 0})
telecom['PaperlessBilling'] = telecom['PaperlessBilling'].map({'Yes': 1, 'No':
0})
telecom['Churn'] = telecom['Churn'].map({'Yes': 1, 'No': 0})
telecom['Partner'] = telecom['Partner'].map({'Yes': 1, 'No': 0})
telecom['Dependents'] = telecom['Dependents'].map({'Yes': 1, 'No': 0})
```

Dummy Variable Creation

In [152]:
```
# Creating a dummy variable for the variable 'Contract' and dropping the first
one.
cont = pd.get_dummies(telecom['Contract'],prefix='Contract',drop_first=True)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,cont],axis=1)

# Creating a dummy variable for the variable 'PaymentMethod' and dropping the
 first one.
pm = pd.get_dummies(telecom['PaymentMethod'],prefix='PaymentMethod',drop_first
=True)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,pm],axis=1)

# Creating a dummy variable for the variable 'gender' and dropping the first o
ne.
gen = pd.get_dummies(telecom['gender'],prefix='gender',drop_first=True)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,gen],axis=1)

# Creating a dummy variable for the variable 'MultipleLines' and dropping the
 first one.
ml = pd.get_dummies(telecom['MultipleLines'],prefix='MultipleLines')
#  dropping MultipleLines_No phone service column
ml1 = ml.drop(['MultipleLines_No phone service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ml1],axis=1)

# Creating a dummy variable for the variable 'InternetService' and dropping th
e first one.
iser = pd.get_dummies(telecom['InternetService'],prefix='InternetService',drop
_first=True)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,iser],axis=1)

# Creating a dummy variable for the variable 'OnlineSecurity'.
os = pd.get_dummies(telecom['OnlineSecurity'],prefix='OnlineSecurity')
os1= os.drop(['OnlineSecurity_No internet service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,os1],axis=1)

# Creating a dummy variable for the variable 'OnlineBackup'.
ob =pd.get_dummies(telecom['OnlineBackup'],prefix='OnlineBackup')
ob1 =ob.drop(['OnlineBackup_No internet service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,ob1],axis=1)

# Creating a dummy variable for the variable 'DeviceProtection'.
dp =pd.get_dummies(telecom['DeviceProtection'],prefix='DeviceProtection')
dp1 = dp.drop(['DeviceProtection_No internet service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,dp1],axis=1)

# Creating a dummy variable for the variable 'TechSupport'.
ts =pd.get_dummies(telecom['TechSupport'],prefix='TechSupport')
ts1 = ts.drop(['TechSupport_No internet service'],1)
#Adding the results to the master dataframe
```

```
telecom = pd.concat([telecom,ts1],axis=1)

# Creating a dummy variable for the variable 'StreamingTV'.
st =pd.get_dummies(telecom['StreamingTV'],prefix='StreamingTV')
st1 = st.drop(['StreamingTV_No internet service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,st1],axis=1)

# Creating a dummy variable for the variable 'StreamingMovies'.
sm =pd.get_dummies(telecom['StreamingMovies'],prefix='StreamingMovies')
sm1 = sm.drop(['StreamingMovies_No internet service'],1)
#Adding the results to the master dataframe
telecom = pd.concat([telecom,sm1],axis=1)
```

Dropping the repeated variables

In [153]:

```
# We have created dummies for the below variables, so we can drop them
telecom = telecom.drop(['Contract','PaymentMethod','gender','MultipleLines','I
nternetService', 'OnlineSecurity', 'OnlineBackup', 'DeviceProtection',
       'TechSupport', 'StreamingTV', 'StreamingMovies'], 1)
```

In [154]:

```
#The varaible was imported as a string we need to convert it to float
telecom['TotalCharges'] =telecom['TotalCharges'].convert_objects(convert_numer
ic=True)
#telecom['tenure'] = telecom['tenure'].astype(int).astype(float)
```

```
C:\ProgramData\Anaconda3\lib\site-packages\ipykernel_launcher.py:2: FutureWar
ning: convert_objects is deprecated.  Use the data-type specific converters p
d.to_datetime, pd.to_timedelta and pd.to_numeric.
```

In [155]:
```
telecom.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 7043 entries, 0 to 7042
Data columns (total 32 columns):
customerID                                 7043 non-null object
tenure                                     7043 non-null int64
PhoneService                               7043 non-null int64
PaperlessBilling                           7043 non-null int64
MonthlyCharges                             7043 non-null float64
TotalCharges                               7032 non-null float64
Churn                                      7043 non-null int64
SeniorCitizen                              7043 non-null int64
Partner                                    7043 non-null int64
Dependents                                 7043 non-null int64
Contract_One year                          7043 non-null uint8
Contract_Two year                          7043 non-null uint8
PaymentMethod_Credit card (automatic)      7043 non-null uint8
PaymentMethod_Electronic check             7043 non-null uint8
PaymentMethod_Mailed check                 7043 non-null uint8
gender_Male                                7043 non-null uint8
MultipleLines_No                           7043 non-null uint8
MultipleLines_Yes                          7043 non-null uint8
InternetService_Fiber optic                7043 non-null uint8
InternetService_No                         7043 non-null uint8
OnlineSecurity_No                          7043 non-null uint8
OnlineSecurity_Yes                         7043 non-null uint8
OnlineBackup_No                            7043 non-null uint8
OnlineBackup_Yes                           7043 non-null uint8
DeviceProtection_No                        7043 non-null uint8
DeviceProtection_Yes                       7043 non-null uint8
TechSupport_No                             7043 non-null uint8
TechSupport_Yes                            7043 non-null uint8
StreamingTV_No                             7043 non-null uint8
StreamingTV_Yes                            7043 non-null uint8
StreamingMovies_No                         7043 non-null uint8
StreamingMovies_Yes                        7043 non-null uint8
dtypes: float64(2), int64(7), object(1), uint8(22)
memory usage: 756.6+ KB
```

Now we can see we have all variables as integer.

Checking for Outliers

In [156]:
```
# Checking for outliers in the continuous variables
num_telecom = telecom[['tenure','MonthlyCharges','SeniorCitizen','TotalCharge
s']]
```

In [157]:
```
# Checking outliers at 25%,50%,75%,90%,95% and 99%
num_telecom.describe(percentiles=[.25,.5,.75,.90,.95,.99])
```

Out[157]:

	tenure	MonthlyCharges	SeniorCitizen	TotalCharges
count	7043.000000	7043.000000	7043.000000	7032.000000
mean	32.371149	64.761692	0.162147	2283.300441
std	24.559481	30.090047	0.368612	2266.771362
min	0.000000	18.250000	0.000000	18.800000
25%	9.000000	35.500000	0.000000	401.450000
50%	29.000000	70.350000	0.000000	1397.475000
75%	55.000000	89.850000	0.000000	3794.737500
90%	69.000000	102.600000	1.000000	5976.640000
95%	72.000000	107.400000	1.000000	6923.590000
99%	72.000000	114.729000	1.000000	8039.883000
max	72.000000	118.750000	1.000000	8684.800000

From the distribution shown above, you can see that there no outliner in your data. The numbers are gradually increasing.

Checking for Missing Values and Inputing Them

It means that 11/7043 = 0.001561834 i.e 0.1%, best is to remove these observations from the analysis

In [158]:
```
# Checking the percentage of missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Out[158]:
```
customerID                                0.00
tenure                                    0.00
PhoneService                              0.00
PaperlessBilling                          0.00
MonthlyCharges                            0.00
TotalCharges                              0.16
Churn                                     0.00
SeniorCitizen                             0.00
Partner                                   0.00
Dependents                                0.00
Contract_One year                         0.00
Contract_Two year                         0.00
PaymentMethod_Credit card (automatic)    0.00
PaymentMethod_Electronic check            0.00
PaymentMethod_Mailed check                0.00
gender_Male                               0.00
MultipleLines_No                          0.00
MultipleLines_Yes                         0.00
InternetService_Fiber optic               0.00
InternetService_No                        0.00
OnlineSecurity_No                         0.00
OnlineSecurity_Yes                        0.00
OnlineBackup_No                           0.00
OnlineBackup_Yes                          0.00
DeviceProtection_No                       0.00
DeviceProtection_Yes                      0.00
TechSupport_No                            0.00
TechSupport_Yes                           0.00
StreamingTV_No                            0.00
StreamingTV_Yes                           0.00
StreamingMovies_No                        0.00
StreamingMovies_Yes                       0.00
dtype: float64
```

In [159]:
```
# Removing NaN TotalCharges rows
telecom = telecom[~np.isnan(telecom['TotalCharges'])]
```

In [160]:
```
# Checking percentage of missing values after removing the missing values
round(100*(telecom.isnull().sum()/len(telecom.index)), 2)
```

Out[160]:
```
customerID                               0.0
tenure                                   0.0
PhoneService                             0.0
PaperlessBilling                         0.0
MonthlyCharges                           0.0
TotalCharges                             0.0
Churn                                    0.0
SeniorCitizen                            0.0
Partner                                  0.0
Dependents                               0.0
Contract_One year                        0.0
Contract_Two year                        0.0
PaymentMethod_Credit card (automatic)    0.0
PaymentMethod_Electronic check           0.0
PaymentMethod_Mailed check               0.0
gender_Male                              0.0
MultipleLines_No                         0.0
MultipleLines_Yes                        0.0
InternetService_Fiber optic              0.0
InternetService_No                       0.0
OnlineSecurity_No                        0.0
OnlineSecurity_Yes                       0.0
OnlineBackup_No                          0.0
OnlineBackup_Yes                         0.0
DeviceProtection_No                      0.0
DeviceProtection_Yes                     0.0
TechSupport_No                           0.0
TechSupport_Yes                          0.0
StreamingTV_No                           0.0
StreamingTV_Yes                          0.0
StreamingMovies_No                       0.0
StreamingMovies_Yes                      0.0
dtype: float64
```

Now we don't have any missing values

Feature Standardisation

In [161]:
```
# Normalising continuous features
df = telecom[['tenure','MonthlyCharges','TotalCharges']]
```

In [162]:
```
normalized_df=(df-df.mean())/df.std()
telecom = telecom.drop(['tenure','MonthlyCharges','TotalCharges'], 1)
telecom = pd.concat([telecom,normalized_df],axis=1)
telecom.head()
```

Out[162]:

	customerID	PhoneService	PaperlessBilling	Churn	SeniorCitizen	Partner	Dependen
0	7590-VHVEG	0	1	0	0	1	0
1	5575-GNVDE	1	0	0	0	0	0
2	3668-QPYBK	1	1	1	0	0	0
3	7795-CFOCW	0	0	0	0	0	0
4	9237-HQITU	1	1	1	0	0	0

5 rows × 32 columns

Checking the Churn Rate

In [163]:
```
churn = (sum(telecom['Churn'])/len(telecom['Churn'].index))*100
churn
```

Out[163]: 26.578498293515356

We have almost 27% churn rate

Model Building

Let's start by splitting our data into a training set and a test set.

Splitting Data into Training and Test Sets

In [164]:
```
from sklearn.model_selection import train_test_split

# Putting feature variable to X
X = telecom.drop(['Churn','customerID'],axis=1)

# Putting response variable to y
y = telecom['Churn']

y.head()
```

Out[164]:
```
0    0
1    0
2    1
3    0
4    1
Name: Churn, dtype: int64
```

In [165]:
```
# Splitting the data into train and test
X_train, X_test, y_train, y_test = train_test_split(X,y, train_size=0.7,test_s
ize=0.3,random_state=100)
```

Running Your First Training Model

In [166]:
```
import statsmodels.api as sm
```

In [167]:
```
# Logistic regression model
logm1 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomi
al())
logm1.fit().summary()
```

Out[167]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4898
Model Family:	Binomial	**Df Model:**	23
Link Function:	logit	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-2004.7
Date:	Wed, 25 Apr 2018	**Deviance:**	4009.4
Time:	20:12:47	**Pearson chi2:**	6.07e+03
No. Iterations:	7		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.2783	1.187	-2.762	0.006	-5.605	-0.952
PhoneService	0.8213	0.588	1.396	0.163	-0.332	1.974
PaperlessBilling	0.3254	0.090	3.614	0.000	0.149	0.502
SeniorCitizen	0.3984	0.102	3.924	0.000	0.199	0.597
Partner	0.0374	0.094	0.399	0.690	-0.146	0.221
Dependents	-0.1430	0.107	-1.332	0.183	-0.353	0.067
Contract_One year	-0.6578	0.129	-5.106	0.000	-0.910	-0.405
Contract_Two year	-1.2455	0.212	-5.874	0.000	-1.661	-0.830
PaymentMethod_Credit card (automatic)	-0.2577	0.137	-1.883	0.060	-0.526	0.011
PaymentMethod_Electronic check	0.1615	0.113	1.434	0.152	-0.059	0.382
PaymentMethod_Mailed check	-0.2536	0.137	-1.845	0.065	-0.523	0.016
gender_Male	-0.0346	0.078	-0.442	0.658	-0.188	0.119
MultipleLines_No	0.1295	0.205	0.632	0.527	-0.272	0.531
MultipleLines_Yes	0.6918	0.392	1.763	0.078	-0.077	1.461
InternetService_Fiber optic	2.5124	0.967	2.599	0.009	0.618	4.407
InternetService_No	-3.4348	1.324	-2.594	0.009	-6.030	-0.839
OnlineSecurity_No	0.0905	0.058	1.558	0.119	-0.023	0.204
OnlineSecurity_Yes	0.0660	0.174	0.380	0.704	-0.275	0.407
OnlineBackup_No	-0.0088	0.055	-0.161	0.872	-0.116	0.098
OnlineBackup_Yes	0.1653	0.172	0.960	0.337	-0.172	0.503
DeviceProtection_No	-0.0832	0.056	-1.487	0.137	-0.193	0.026
DeviceProtection_Yes	0.2397	0.174	1.379	0.168	-0.101	0.580

TechSupport_No	0.0935	0.058	1.604	0.109	-0.021	0.208
TechSupport_Yes	0.0630	0.174	0.362	0.717	-0.278	0.404
StreamingTV_No	-0.4016	0.133	-3.027	0.002	-0.662	-0.142
StreamingTV_Yes	0.5581	0.267	2.094	0.036	0.036	1.081
StreamingMovies_No	-0.3459	0.133	-2.609	0.009	-0.606	-0.086
StreamingMovies_Yes	0.5024	0.266	1.886	0.059	-0.020	1.025
tenure	-1.5198	0.190	-8.015	0.000	-1.891	-1.148
MonthlyCharges	-2.1817	1.160	-1.880	0.060	-4.456	0.092
TotalCharges	0.7329	0.198	3.705	0.000	0.345	1.121

Correlation Matrix

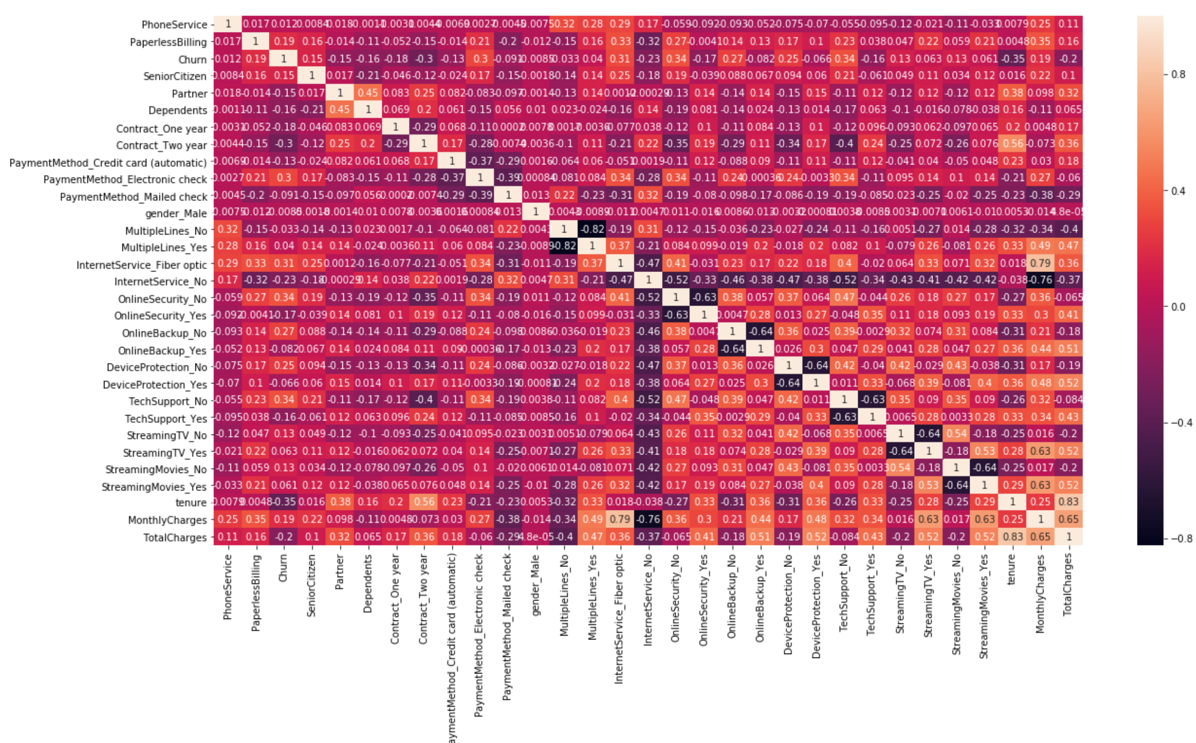
In [168]:

```
# Importing matplotlib and seaborn
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

In [169]:

```
# Let's see the correlation matrix
plt.figure(figsize = (20,10))        # Size of the figure
sns.heatmap(telecom.corr(),annot = True)
```

Out[169]: <matplotlib.axes._subplots.AxesSubplot at 0x290250a59b0>



Dropping highly correlated variables.

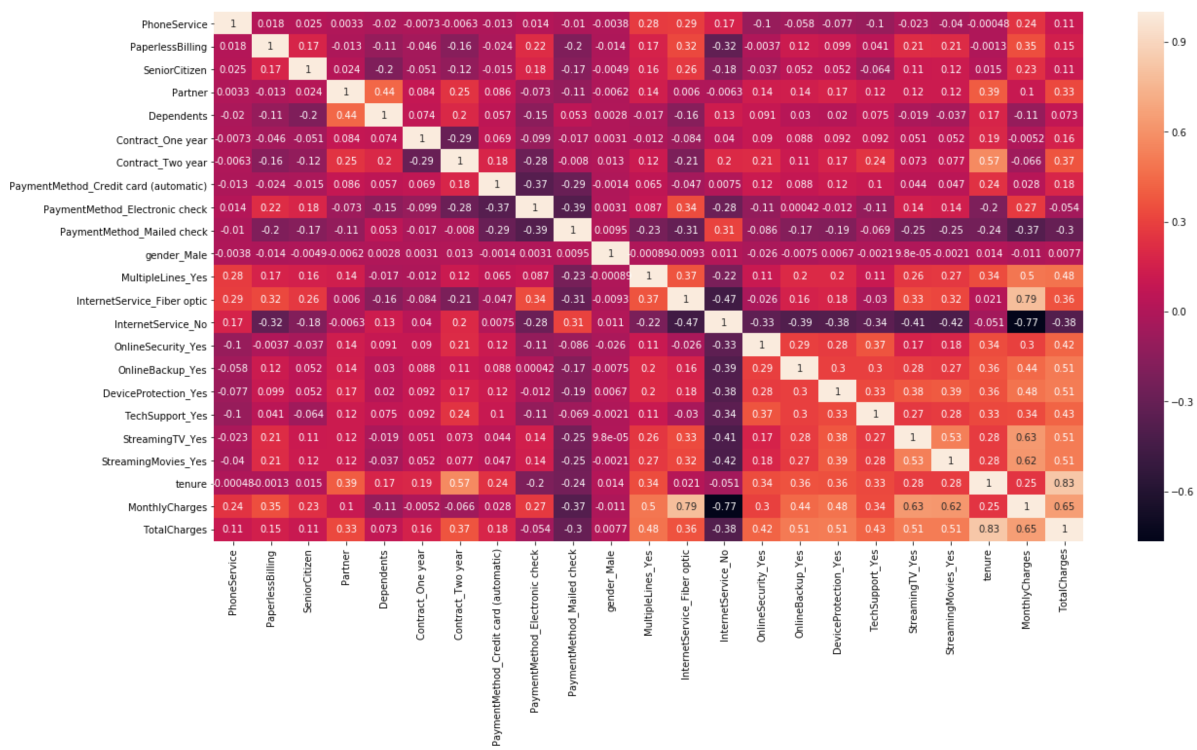
In [170]:
```
X_test2 = X_test.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No','StreamingTV_No','StreamingMovies_No'],1)
X_train2 = X_train.drop(['MultipleLines_No','OnlineSecurity_No','OnlineBackup_No','DeviceProtection_No','TechSupport_No','StreamingTV_No','StreamingMovies_No'],1)
```

Checking the Correlation Matrix

After dropping highly correlated variables now let's check the correlation matrix again.

In [171]:
```
plt.figure(figsize = (20,10))
sns.heatmap(X_train2.corr(),annot = True)
```

Out[171]: <matplotlib.axes._subplots.AxesSubplot at 0x290291cdb70>



Re-Running the Model

Now let's run our model again after dropping highly correlated variables

In [172]:
```
logm2 = sm.GLM(y_train,(sm.add_constant(X_train2)), family = sm.families.Binom
ial())
logm2.fit().summary()
```

Out[172]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4898
Model Family:	Binomial	**Df Model:**	23
Link Function:	logit	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-2004.7
Date:	Wed, 25 Apr 2018	**Deviance:**	4009.4
Time:	20:12:54	**Pearson chi2:**	6.07e+03
No. Iterations:	7		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.9338	1.545	-2.545	0.011	-6.963	-0.905
PhoneService	0.9507	0.789	1.205	0.228	-0.595	2.497
PaperlessBilling	0.3254	0.090	3.614	0.000	0.149	0.502
SeniorCitizen	0.3984	0.102	3.924	0.000	0.199	0.597
Partner	0.0374	0.094	0.399	0.690	-0.146	0.221
Dependents	-0.1430	0.107	-1.332	0.183	-0.353	0.067
Contract_One year	-0.6578	0.129	-5.106	0.000	-0.910	-0.405
Contract_Two year	-1.2455	0.212	-5.874	0.000	-1.661	-0.830
PaymentMethod_Credit card (automatic)	-0.2577	0.137	-1.883	0.060	-0.526	0.011
PaymentMethod_Electronic check	0.1615	0.113	1.434	0.152	-0.059	0.382
PaymentMethod_Mailed check	-0.2536	0.137	-1.845	0.065	-0.523	0.016
gender_Male	-0.0346	0.078	-0.442	0.658	-0.188	0.119
MultipleLines_Yes	0.5623	0.214	2.628	0.009	0.143	0.982
InternetService_Fiber optic	2.5124	0.967	2.599	0.009	0.618	4.407
InternetService_No	-2.7792	0.982	-2.831	0.005	-4.703	-0.855
OnlineSecurity_Yes	-0.0245	0.216	-0.113	0.910	-0.448	0.399
OnlineBackup_Yes	0.1740	0.212	0.822	0.411	-0.241	0.589
DeviceProtection_Yes	0.3229	0.215	1.501	0.133	-0.099	0.744
TechSupport_Yes	-0.0305	0.216	-0.141	0.888	-0.455	0.394
StreamingTV_Yes	0.9598	0.396	2.423	0.015	0.183	1.736
StreamingMovies_Yes	0.8484	0.396	2.143	0.032	0.072	1.624
tenure	-1.5198	0.190	-8.015	0.000	-1.891	-1.148

MonthlyCharges	-2.1817	1.160	-1.880	0.060	-4.456	0.092
TotalCharges	0.7329	0.198	3.705	0.000	0.345	1.121

Feature Selection Using RFE

In [173]:
```
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
from sklearn.feature_selection import RFE
rfe = RFE(logreg, 13)             # running RFE with 13 variables as output
rfe = rfe.fit(X,y)
print(rfe.support_)           # Printing the boolean results
print(rfe.ranking_)           # Printing the ranking
```

```
[ True  True False False False  True  True False  True False False  True
 False  True  True False  True False False False False False  True False
 False  True False  True False  True]
[ 1  1  2 18  6  1  1 11  1 12 14  1  8  1  1  4  1 15  5 13 10  7  1  3 16
  1 17  1  9  1]
```

In [174]:
```
# Variables selected by RFE
col = ['PhoneService', 'PaperlessBilling', 'Contract_One year', 'Contract_Two
 year',
       'PaymentMethod_Electronic check','MultipleLines_No','InternetService_Fi
ber optic', 'InternetService_No',
       'OnlineSecurity_Yes','TechSupport_Yes','StreamingMovies_No','tenure','T
otalCharges']
```

In [175]:
```
# Let's run the model using the selected variables
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logsk = LogisticRegression(C=1e9)
#logsk.fit(X_train[col], y_train)
logsk.fit(X_train, y_train)
```

Out[175]:
```
LogisticRegression(C=1000000000.0, class_weight=None, dual=False,
          fit_intercept=True, intercept_scaling=1, max_iter=100,
          multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False)
```

In [176]:
```
#Comparing the model with StatsModels
#logm4 = sm.GLM(y_train,(sm.add_constant(X_train[col])), family = sm.families.
Binomial())
logm4 = sm.GLM(y_train,(sm.add_constant(X_train)), family = sm.families.Binomi
al())
modres = logm4.fit()
logm4.fit().summary()
```

Out[176]:

Generalized Linear Model Regression Results

Dep. Variable:	Churn	**No. Observations:**	4922
Model:	GLM	**Df Residuals:**	4898
Model Family:	Binomial	**Df Model:**	23
Link Function:	logit	**Scale:**	1.0
Method:	IRLS	**Log-Likelihood:**	-2004.7
Date:	Wed, 25 Apr 2018	**Deviance:**	4009.4
Time:	20:12:55	**Pearson chi2:**	6.07e+03
No. Iterations:	7		

	coef	std err	z	P>|z|	[0.025	0.975]
const	-3.2783	1.187	-2.762	0.006	-5.605	-0.952
PhoneService	0.8213	0.588	1.396	0.163	-0.332	1.974
PaperlessBilling	0.3254	0.090	3.614	0.000	0.149	0.502
SeniorCitizen	0.3984	0.102	3.924	0.000	0.199	0.597
Partner	0.0374	0.094	0.399	0.690	-0.146	0.221
Dependents	-0.1430	0.107	-1.332	0.183	-0.353	0.067
Contract_One year	-0.6578	0.129	-5.106	0.000	-0.910	-0.405
Contract_Two year	-1.2455	0.212	-5.874	0.000	-1.661	-0.830
PaymentMethod_Credit card (automatic)	-0.2577	0.137	-1.883	0.060	-0.526	0.011
PaymentMethod_Electronic check	0.1615	0.113	1.434	0.152	-0.059	0.382
PaymentMethod_Mailed check	-0.2536	0.137	-1.845	0.065	-0.523	0.016
gender_Male	-0.0346	0.078	-0.442	0.658	-0.188	0.119
MultipleLines_No	0.1295	0.205	0.632	0.527	-0.272	0.531
MultipleLines_Yes	0.6918	0.392	1.763	0.078	-0.077	1.461
InternetService_Fiber optic	2.5124	0.967	2.599	0.009	0.618	4.407
InternetService_No	-3.4348	1.324	-2.594	0.009	-6.030	-0.839
OnlineSecurity_No	0.0905	0.058	1.558	0.119	-0.023	0.204
OnlineSecurity_Yes	0.0660	0.174	0.380	0.704	-0.275	0.407
OnlineBackup_No	-0.0088	0.055	-0.161	0.872	-0.116	0.098
OnlineBackup_Yes	0.1653	0.172	0.960	0.337	-0.172	0.503
DeviceProtection_No	-0.0832	0.056	-1.487	0.137	-0.193	0.026
DeviceProtection_Yes	0.2397	0.174	1.379	0.168	-0.101	0.580

TechSupport_No	0.0935	0.058	1.604	0.109	-0.021	0.208
TechSupport_Yes	0.0630	0.174	0.362	0.717	-0.278	0.404
StreamingTV_No	-0.4016	0.133	-3.027	0.002	-0.662	-0.142
StreamingTV_Yes	0.5581	0.267	2.094	0.036	0.036	1.081
StreamingMovies_No	-0.3459	0.133	-2.609	0.009	-0.606	-0.086
StreamingMovies_Yes	0.5024	0.266	1.886	0.059	-0.020	1.025
tenure	-1.5198	0.190	-8.015	0.000	-1.891	-1.148
MonthlyCharges	-2.1817	1.160	-1.880	0.060	-4.456	0.092
TotalCharges	0.7329	0.198	3.705	0.000	0.345	1.121

In [177]:
```
X_test[col].shape
#res = modres.predict(X_test[col])
```

Out[177]: (2110, 13)

Making Predictions

In [178]:
```
# Predicted probabilities
y_pred = logsk.predict_proba(X_test)
# Converting y_pred to a dataframe which is an array
y_pred_df = pd.DataFrame(y_pred)
# Converting to column dataframe
y_pred_1 = y_pred_df.iloc[:,[1]]
# Let's see the head
y_pred_1.head()
```

Out[178]:

	1
0	0.437703
1	0.326774
2	0.004873
3	0.576256
4	0.007747

In [179]:
```
# Converting y_test to dataframe
y_test_df = pd.DataFrame(y_test)
y_test_df.head()
```

Out[179]:

	Churn
942	0
3730	1
1761	0
2283	1
1872	0

In [180]:
```
# Putting CustID to index
y_test_df['CustID'] = y_test_df.index
# Removing index for both dataframes to append them side by side
y_pred_1.reset_index(drop=True, inplace=True)
y_test_df.reset_index(drop=True, inplace=True)
# Appending y_test_df and y_pred_1
y_pred_final = pd.concat([y_test_df,y_pred_1],axis=1)
# Renaming the column
y_pred_final= y_pred_final.rename(columns={ 1 : 'Churn_Prob'})
# Rearranging the columns
y_pred_final = y_pred_final.reindex_axis(['CustID','Churn','Churn_Prob'], axis
=1)
# Let's see the head of y_pred_final
y_pred_final.head()
```

Out[180]:

	CustID	Churn	Churn_Prob
0	942	0	0.437703
1	3730	1	0.326774
2	1761	0	0.004873
3	2283	1	0.576256
4	1872	0	0.007747

In [181]:
```
# Creating new column 'predicted' with 1 if Churn_Prob>0.5 else 0
y_pred_final['predicted'] = y_pred_final.Churn_Prob.map( lambda x: 1 if x > 0.
5 else 0)
# Let's see the head
y_pred_final.head()
```

Out[181]:

	CustID	Churn	Churn_Prob	predicted
0	942	0	0.437703	0
1	3730	1	0.326774	0
2	1761	0	0.004873	0
3	2283	1	0.576256	1
4	1872	0	0.007747	0

Model Evaluation

In [182]:
```
from sklearn import metrics
```

In [183]:
```
# Confusion matrix
confusion = metrics.confusion_matrix( y_pred_final.Churn, y_pred_final.predict
ed )
confusion
```

Out[183]:
```
array([[1354,  174],
       [ 248,  334]], dtype=int64)
```

In [184]:
```
# Predicted     Churn  not_churn  __all__
# Actual
# Churn            1359   169     1528
# not_churn         256   326      582
# __all__          1615   751     2110
```

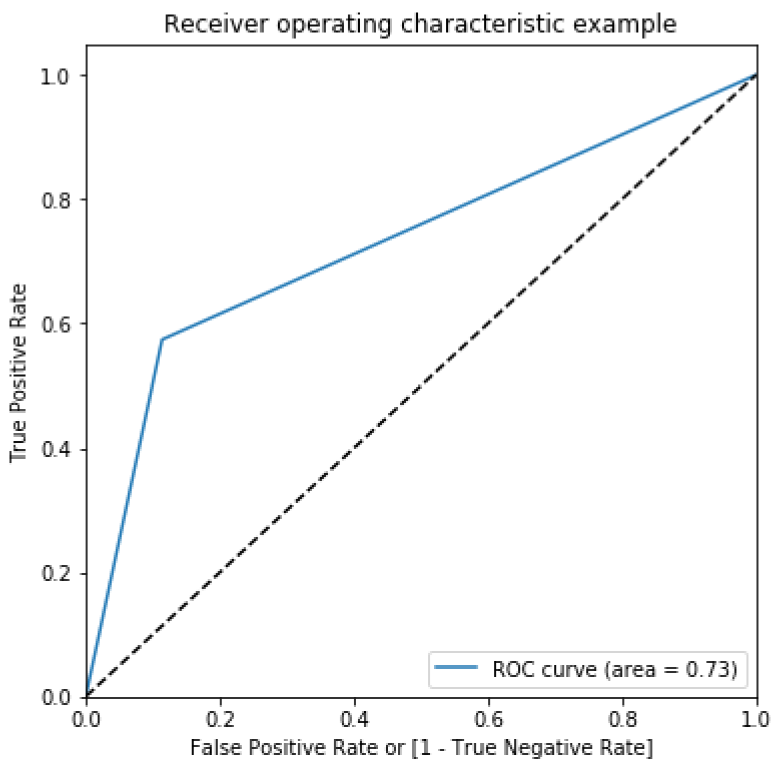
In [185]:
```
#Let's check the overall accuracy.
metrics.accuracy_score(y_pred_final.Churn, y_pred_final.predicted)
```

Out[185]:
```
0.8000000000000004
```

In [186]:
```
def draw_roc( actual, probs ):
    fpr, tpr, thresholds = metrics.roc_curve( actual, probs,
                                              drop_intermediate = False )
    auc_score = metrics.roc_auc_score( actual, probs )
    plt.figure(figsize=(6, 6))
    plt.plot( fpr, tpr, label='ROC curve (area = %0.2f)' % auc_score )
    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate or [1 - True Negative Rate]')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()

    return fpr, tpr, thresholds
```

In [187]:
```
draw_roc(y_pred_final.Churn, y_pred_final.predicted)
```



Out[187]:
```
(array([ 0.        ,  0.11387435,  1.        ]),
 array([ 0.        ,  0.57388316,  1.        ]),
 array([2, 1, 0], dtype=int64))
```

In [188]:
```
#draw_roc(y_pred_final.Churn, y_pred_final.predicted)
"{:2.2f}".format(metrics.roc_auc_score(y_pred_final.Churn, y_pred_final.Churn_
Prob))
```

Out[188]:
```
'0.83'
```

We see an overall AUC score of 0.83 looks like we did a decent job.

- But we did spend a lot of effort on the features and their selection.
- Can PCA help reduce our effort?

PCA on the data

Note -

- While computng the principal components, we must not include the entire dataset. Model building is all about doing well on the data we haven't seen yet!
- So we'll calculate the PCs using the train data, and apply them later on the test data

In [189]:
```
X_train.shape
# We have 30 variables after creating our dummy variables for our categories
```

Out[189]:
```
(4922, 30)
```

In [190]:
```
#Improting the PCA module
from sklearn.decomposition import PCA
pca = PCA(svd_solver='randomized', random_state=42)
```

In [191]:
```
#Doing the PCA on the train data
pca.fit(X_train)
```

Out[191]:
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=42,
  svd_solver='randomized', tol=0.0, whiten=False)
```

Let's plot the principal components and try to make sense of them

- We'll plot original features on the first 2 principal components as axes

In [192]:
```
pca.components_
```

```
Out[192]: array([[  1.76113503e-02,   6.73236050e-02,   3.33558780e-02,
                    9.81025382e-02,   1.45144894e-02,   2.99711219e-02,
                    8.03474721e-02,   4.45137169e-02,   1.03838839e-02,
                   -9.64778216e-02,   5.46455027e-04,  -1.44992416e-01,
                    1.62603767e-01,   1.38408762e-01,  -1.23559445e-01,
                    1.07552300e-02,   1.12804215e-01,  -2.93144081e-02,
                    1.52873853e-01,  -3.66586107e-02,   1.60218056e-01,
                    2.26069096e-03,   1.21298754e-01,  -5.19779044e-02,
                    1.75537350e-01,  -5.32208106e-02,   1.76780256e-01,
                    4.50041974e-01,   4.64768058e-01,   5.64467383e-01],
                 [ -1.58225213e-02,  -1.51382114e-01,  -7.60391859e-02,
                    1.10808074e-01,   1.15262476e-01,   5.70934546e-02,
                    1.99196237e-01,   6.98211144e-02,  -1.94953309e-01,
                    7.22696236e-02,   6.84716202e-03,   3.64391587e-02,
                   -5.22616800e-02,  -2.53405153e-01,   2.28888352e-01,
                   -2.80064773e-01,   5.11764215e-02,  -2.29155531e-01,
                    2.67178910e-04,  -2.30417558e-01,   1.52920583e-03,
                   -2.74825798e-01,   4.59374458e-02,  -1.48754387e-01,
                   -8.01339654e-02,  -1.50215959e-01,  -7.86723933e-02,
                    4.33270268e-01,  -4.56258849e-01,   1.27410027e-01],
                 [ -9.12291058e-02,  -2.84369927e-02,  -1.24001243e-02,
                    5.63239688e-02,   4.78714602e-02,  -7.62225596e-04,
                    2.57781910e-02,   5.81002273e-02,  -7.61267399e-02,
                   -2.15070492e-02,  -8.51174807e-03,  -8.76630128e-02,
                   -3.56609297e-03,  -9.44615765e-02,  -1.92982686e-01,
                   -2.70477190e-02,   2.20030405e-01,   7.09799469e-02,
                    1.22002739e-01,   2.79335424e-01,  -8.63527377e-02,
                    1.14699466e-01,   7.82832203e-02,   4.96232463e-01,
                   -3.03249777e-01,   4.89684449e-01,  -2.96701763e-01,
                    2.43036444e-01,  -1.22710705e-01,   8.38171907e-02],
                 [ -1.52582548e-02,  -8.53395795e-02,  -1.24482551e-01,
                   -1.15607472e-01,   1.35706616e-02,   6.18957244e-02,
                   -2.55122762e-02,   1.08439476e-02,  -1.54943819e-01,
                    1.50132655e-01,  -3.27194885e-02,   3.35259920e-01,
                   -3.50518175e-01,  -9.95677606e-02,  -1.49234454e-01,
                   -1.94633301e-01,   3.43867755e-01,   1.37892828e-02,
                    1.35445172e-01,  -7.12015820e-02,   2.20436036e-01,
                   -2.81687409e-01,   4.30921863e-01,   6.78808741e-02,
                    8.13535803e-02,   6.63634814e-02,   8.28709730e-02,
                   -2.84331268e-01,   2.16875383e-01,  -5.64112933e-03],
                 [ -1.14905477e-01,  -3.91805880e-02,  -2.43453437e-02,
                    9.22236587e-02,   8.22694189e-02,  -7.39508732e-02,
                    1.11130926e-01,   5.20967416e-02,  -4.54904532e-02,
                   -1.28556843e-02,   2.36253723e-02,  -2.38918946e-01,
                    1.24013469e-01,  -1.13486010e-01,  -7.56624541e-02,
                   -2.67674094e-03,   7.83391950e-02,   6.40827206e-01,
                   -5.65164752e-01,  -1.26242983e-01,   2.01905438e-01,
                   -9.84497332e-02,   1.74112187e-01,   4.01689953e-02,
                    3.54934588e-02,  -2.25276324e-02,   9.81900865e-02,
                    3.70542100e-02,  -6.07852630e-02,  -6.31576961e-02],
                 [  3.64322765e-02,   2.12864636e-02,   4.16857747e-03,
                    3.12327904e-01,   2.07866485e-01,   8.80938644e-02,
                   -2.10506892e-02,   1.98347068e-02,   8.59334323e-02,
                   -1.24909455e-01,   4.63903329e-02,   5.41058303e-01,
                   -5.04626027e-01,   7.84512080e-02,  -3.51118051e-02,
                    1.62939067e-01,  -1.27827262e-01,   1.91948712e-01,
                   -1.56836907e-01,  -3.60056963e-02,   7.11175015e-02,
```

```
  2.33630834e-01, -1.98519029e-01, -9.70663787e-03,
  4.48184430e-02,  4.16890950e-02, -6.57728987e-03,
  2.28607186e-01, -2.06491922e-02,  8.76347710e-02],
[ 2.37558672e-02,  3.68307201e-02, -5.24480693e-02,
  2.34251271e-01,  2.68532524e-01, -4.31941644e-02,
  4.59426788e-02, -9.63796664e-02,  7.86145743e-02,
  1.24589062e-02, -1.11887496e-01,  3.03343294e-02,
 -6.57846219e-03, -1.05198190e-02,  6.34620160e-03,
 -1.37533195e-01,  1.31186993e-01,  7.11834413e-02,
 -7.75296429e-02,  5.65951236e-01, -5.72297438e-01,
 -1.62317117e-01,  1.55970916e-01, -1.66257206e-01,
  1.59911004e-01, -1.35056505e-01,  1.28710304e-01,
  1.91083205e-03,  3.49478688e-02, -1.28038995e-02],
[ -4.49367286e-02, -4.47541098e-02, -7.15283654e-02,
  5.72645409e-01,  5.63228733e-01,  1.39533786e-02,
 -4.78769773e-02, -5.80186925e-02,  1.86761580e-02,
  4.75212017e-02, -8.56435722e-02, -2.04751136e-01,
  1.59814408e-01, -2.78756283e-02, -3.39057422e-02,
  4.73805254e-02, -1.34747832e-02, -1.71471202e-01,
  2.05376944e-01, -1.90547868e-01,  2.24453611e-01,
  3.77595251e-02, -3.85378286e-03,  3.01358694e-02,
  3.76987280e-03,  6.72221021e-02, -3.33163599e-02,
 -2.56902570e-01,  6.08435868e-02, -1.48787319e-01],
[ -1.99321058e-01,  1.28143552e-01, -2.76612593e-02,
  1.18692098e-03, -1.33525239e-03,  2.09147058e-02,
  2.03651182e-02,  3.75548892e-03,  1.98669459e-02,
 -2.50873854e-02,  3.35839251e-01, -8.30605070e-02,
 -1.16260550e-01, -2.00194898e-01, -1.12524144e-01,
  5.27226188e-01, -4.14702044e-01, -5.97458675e-02,
  1.72270012e-01,  1.11507832e-01,  1.01631195e-03,
 -2.65655371e-01,  3.78179515e-01,  3.26632791e-02,
  7.98608653e-02,  7.22825848e-03,  1.05295886e-01,
  8.04802828e-02, -1.41092824e-01, -3.25688162e-02],
[ -1.31187206e-01,  1.42382500e-01,  2.93814462e-02,
 -4.22359444e-02, -9.13274583e-02,  4.45203489e-02,
 -1.97375623e-02,  6.77191208e-02,  2.39395079e-02,
 -1.02780272e-01, -9.06612283e-01, -3.49326686e-02,
 -9.62545372e-02, -1.18359923e-01, -5.26355284e-02,
  1.67901386e-01, -1.15265857e-01, -1.50972398e-03,
  5.41452523e-02, -1.23184461e-02,  6.49539744e-02,
 -2.30498121e-02,  7.56853405e-02,  3.20045531e-02,
  2.06309753e-02, -3.91320120e-02,  9.17675404e-02,
  7.35224250e-02, -1.08014857e-01, -2.66003355e-02],
[ -3.49879546e-01,  1.04381424e-01,  2.80953095e-02,
 -3.48183263e-02, -4.90519748e-02,  2.13964832e-01,
 -1.60535217e-01, -5.50706965e-02,  2.54308048e-01,
 -1.75624151e-01,  1.72038848e-01, -1.91412168e-01,
 -1.58467378e-01, -3.24634497e-01, -1.37460083e-01,
 -1.90128855e-01,  3.27588938e-01, -1.66314733e-02,
  1.54091557e-01,  4.98402187e-02,  8.76198648e-02,
  2.92912229e-01, -1.55452146e-01, -3.15985635e-03,
  1.40619940e-01, -1.71289326e-01,  3.08749409e-01,
  1.95901009e-02, -1.96864140e-01, -7.31480884e-02],
[ -9.12683105e-02,  2.24720585e-02, -2.37045985e-02,
 -7.69677984e-02, -1.95091796e-02,  1.87880802e-02,
 -4.57773605e-03,  5.66215034e-02, -3.12527703e-02,
 -1.99335109e-02, -2.05789615e-02, -7.97160167e-02,
```

```
        -1.15522938e-02,  -8.85214748e-02,  -4.39111566e-02,
         2.92010275e-04,   4.36191463e-02,   3.23120593e-02,
         1.15990973e-02,   1.02217625e-02,   3.36893941e-02,
         6.15661271e-02,  -1.76549705e-02,  -5.01861411e-01,
         5.45772568e-01,   4.69601826e-01,  -4.25690669e-01,
         9.92700901e-03,  -4.78727339e-02,  -1.96081921e-02],
       [ 5.15736298e-02,   1.92090527e-01,   7.78486130e-02,
         3.21859309e-02,  -6.91660131e-02,  -1.81790324e-01,
         1.00480781e-01,  -4.73708557e-01,   6.61754494e-01,
        -1.77428321e-01,  -1.08878739e-02,   6.12083829e-02,
        -9.63475311e-03,   4.78667618e-02,   7.18831540e-02,
        -1.28569541e-01,   5.66863866e-02,  -3.62635139e-02,
        -3.56196400e-02,  -1.56663195e-01,   8.47800406e-02,
        -2.39522731e-01,   1.67639577e-01,   1.67545497e-02,
        -8.86377037e-02,   1.14493456e-01,  -1.86376610e-01,
         5.83606761e-02,  -3.41088604e-02,  -1.08141420e-02],
       [ 1.12540380e-01,   1.04754644e-01,   2.01958885e-02,
         2.61566597e-02,   4.35742917e-03,   7.49475248e-01,
        -5.17810424e-01,  -7.60966376e-02,   3.19631316e-02,
         2.67713561e-02,  -9.65693153e-04,  -1.84411426e-02,
         1.30981523e-01,   6.60001206e-02,   8.40532859e-02,
        -3.03312662e-02,  -5.37220197e-02,   6.20662858e-02,
        -1.46119572e-01,  -3.11032646e-02,  -5.29500214e-02,
        -1.83563066e-01,   9.95097797e-02,  -1.01356512e-03,
        -8.30397208e-02,   4.08872568e-02,  -1.24940543e-01,
         2.34534386e-02,  -1.16417906e-02,   8.78640864e-02],
       [ 6.64399911e-02,   6.64868623e-01,   8.36378984e-02,
         7.71690147e-02,   4.28265468e-02,  -7.37325025e-02,
         2.37133513e-02,   5.23718670e-01,  -9.87360270e-02,
        -3.76083796e-01,   1.04019679e-01,   4.72330915e-02,
         1.92068996e-02,   5.96400549e-02,   6.97585182e-02,
        -1.54318265e-01,   8.45597470e-02,  -5.62667300e-02,
        -1.34917881e-02,  -5.48129319e-02,  -1.49455862e-02,
        -1.19891424e-01,   5.01329063e-02,   1.16683113e-03,
        -7.09253493e-02,  -9.12739390e-03,  -6.06311242e-02,
        -8.20950938e-02,   1.19639617e-02,  -1.23848529e-01],
       [-3.75219113e-02,   6.36070910e-01,  -1.07982597e-02,
         4.22922840e-04,   2.38155130e-02,  -7.76788055e-02,
         2.32114140e-02,  -3.79907809e-01,  -2.61143105e-01,
         5.72999577e-01,   2.47005651e-03,  -1.35320618e-03,
        -3.61687052e-02,  -3.96543612e-02,  -1.32212820e-02,
        -9.22748752e-03,   2.24487695e-02,   3.97368647e-02,
        -2.65155827e-02,  -2.11254955e-02,   3.43467776e-02,
         1.14758865e-01,  -1.01537583e-01,   6.61264229e-03,
         6.60863972e-03,  -3.71701054e-03,   1.69382926e-02,
         4.10971250e-02,  -5.67868728e-02,   9.77400645e-02],
       [-6.42314627e-02,  -1.07042095e-01,   6.96289265e-01,
         4.94894995e-01,  -4.42672947e-01,  -4.63966136e-02,
        -5.57445836e-02,   1.37992977e-03,  -1.26834655e-01,
         1.12181055e-01,   1.85530168e-02,  -5.03047899e-03,
        -5.92009838e-02,  -5.91700299e-03,  -7.52148739e-03,
        -3.73900725e-02,   4.49115599e-02,  -1.09502296e-02,
         1.84717170e-02,   1.99115090e-02,  -1.23900216e-02,
        -6.28547597e-02,   7.03762471e-02,  -2.89567467e-02,
         3.64782340e-02,   1.26199850e-02,  -5.09849758e-03,
        -6.57729166e-02,  -2.13310254e-02,  -1.87746975e-02],
       [-6.29527960e-02,  -4.23815881e-02,   6.68576397e-01,
```

```
        -4.58402078e-01,   5.72297440e-01,  -6.56620264e-03,
        -6.05907281e-03,  -1.28634041e-03,   1.11261262e-03,
         2.93631520e-02,  -8.00267719e-03,  -8.38644638e-03,
        -5.45663496e-02,   4.32867338e-02,   5.29293625e-03,
         1.87739836e-03,  -7.17033461e-03,   1.03662971e-03,
        -6.32956597e-03,  -1.48353179e-02,   9.54238164e-03,
        -2.54509470e-02,   2.01580107e-02,   2.94534341e-03,
        -8.23827966e-03,  -4.82270721e-03,  -4.70229044e-04,
         2.94223402e-02,  -1.62910120e-02,   3.06362337e-02],
      [ 5.72948911e-01,   3.30290813e-02,   1.29919132e-01,
        -1.31384672e-02,   2.49030224e-03,   6.32435773e-02,
         1.04575808e-01,  -4.25297703e-02,   2.11843581e-02,
        -5.14680269e-02,  -9.21577354e-04,   2.49503821e-01,
         3.23445090e-01,  -6.39775067e-01,  -9.67647857e-02,
         8.81962776e-02,   8.56850815e-03,   4.71397337e-02,
         4.96250521e-02,   4.11403389e-02,   5.56244469e-02,
         1.11477293e-01,  -1.47125072e-02,   2.51103720e-02,
         7.16544137e-02,   2.41516547e-02,   7.26131310e-02,
         1.20404317e-02,   5.13460683e-02,  -6.18374046e-02],
      [ -1.92741060e-02,   1.48925758e-02,   3.72041945e-02,
        -2.05683940e-02,  -3.68803457e-02,   3.20881446e-01,
         3.61033750e-01,  -2.53245289e-01,  -2.27688221e-01,
        -1.35707348e-01,   8.43728098e-03,  -1.91981617e-02,
        -7.59443712e-05,   1.99858786e-01,  -3.37085941e-02,
        -1.17991290e-02,   4.55077231e-02,  -2.70212256e-02,
         6.07298197e-02,   3.06810135e-03,   3.06404928e-02,
         1.07425357e-02,   2.29660584e-02,   1.38938473e-03,
         3.23192094e-02,   8.19388964e-04,   3.28892051e-02,
         3.34907489e-01,   2.27639259e-01,  -6.42407120e-01],
      [ -2.72767838e-03,  -1.93018158e-02,  -1.93627564e-02,
        -1.57236941e-02,   2.82987476e-02,  -2.28946550e-01,
        -2.96986396e-01,  -4.84588230e-01,  -5.10231870e-01,
        -5.76418652e-01,   7.75029934e-03,   2.92320591e-03,
        -5.65088429e-03,  -4.74306198e-02,   1.49011092e-02,
        -7.03288986e-03,  -7.86821932e-03,   1.96849149e-03,
        -1.68696007e-02,  -1.11816550e-02,  -3.71945417e-03,
        -1.87140919e-02,   3.81298272e-03,  -1.35589379e-02,
        -1.34217131e-03,  -1.24243714e-02,  -2.47673773e-03,
        -6.99823286e-02,  -5.78788956e-02,   1.34497113e-01],
      [ 7.38350600e-02,  -3.96135956e-02,  -4.43430271e-02,
        -1.46608268e-02,   2.69936275e-02,  -4.00447291e-01,
        -6.29924666e-01,   1.16084030e-01,   6.22954483e-02,
         1.81976312e-01,   2.27652840e-03,   3.74165169e-02,
         3.64185431e-02,  -1.79010164e-02,  -1.83435439e-02,
        -2.69524656e-02,   4.52960096e-02,  -1.19466536e-02,
         3.02901975e-02,  -2.73269414e-02,   4.56704854e-02,
        -4.16517296e-02,   5.99952735e-02,  -2.09229565e-02,
         3.92665004e-02,  -2.21235347e-02,   4.04670787e-02,
         4.50602734e-01,   1.09097778e-01,  -3.84693458e-01],
      [ -2.99022929e-01,   5.35042308e-04,  -1.11219964e-03,
         1.13356455e-03,  -1.43359627e-03,  -9.85106280e-04,
        -1.51274106e-04,   6.35650382e-04,   9.91521055e-04,
         4.11139360e-04,  -1.97154554e-04,  -1.00028207e-01,
        -1.98994722e-01,  -4.95024115e-01,   2.23630891e-01,
        -6.21749560e-02,  -1.61455935e-01,  -6.24676364e-02,
        -1.61163254e-01,  -6.16188725e-02,  -1.62012018e-01,
        -6.23321503e-02,  -1.61298740e-01,  -1.28236736e-02,
```

```
 -2.10807217e-01, -1.28750684e-02, -2.10755822e-01,
  1.27331923e-03,  5.97464532e-01, -1.82730782e-03],
[ 0.00000000e+00, -1.40905131e-16, -1.43410469e-16,
  1.41191596e-16,  2.00553844e-16,  1.82560934e-18,
 -1.96421234e-16, -6.78502107e-17, -1.14531033e-16,
  1.05614642e-16,  9.26471013e-17,  9.22424395e-16,
  1.17098423e-15,  1.16953775e-15, -3.24776739e-01,
 -5.29781333e-01, -5.29781333e-01,  1.50670388e-01,
  1.50670388e-01, -3.05603035e-01, -3.05603035e-01,
  5.19421952e-02,  5.19421952e-02,  1.67571901e-01,
  1.67571901e-01,  1.40423145e-01,  1.40423145e-01,
 -4.20725378e-17, -1.43849774e-15,  1.59998040e-17],
[ 0.00000000e+00, -5.83384446e-17, -6.95872810e-18,
 -8.55200389e-17,  1.31373668e-16, -2.27252330e-17,
 -2.09945117e-16, -1.33530098e-16, -4.74472624e-17,
  1.37359212e-16,  3.49138630e-18,  3.38745992e-16,
  4.88662654e-16,  2.46927133e-16, -2.48426864e-01,
 -1.62652515e-01, -1.62652515e-01, -6.42559144e-02,
 -6.42559144e-02,  3.19885027e-01,  3.19885027e-01,
 -2.29207095e-01, -2.29207095e-01,  3.16327010e-01,
  3.16327010e-01, -4.28523376e-01, -4.28523376e-01,
 -3.87461060e-17, -4.70990476e-16,  8.33129929e-17],
[ 0.00000000e+00, -7.96309904e-17,  8.75102375e-17,
  4.86244156e-17,  1.53128592e-17,  6.09799837e-17,
  4.42240075e-17,  2.28241718e-16, -1.04989321e-16,
  8.21764340e-17, -1.04200286e-16,  2.50855407e-16,
  3.32545872e-16,  3.44531205e-16,  9.24267395e-02,
 -8.42502665e-02, -8.42502665e-02,  3.94341795e-01,
  3.94341795e-01,  2.10781858e-01,  2.10781858e-01,
 -4.57937403e-01, -4.57937403e-01, -1.83439613e-01,
 -1.83439613e-01,  2.12930369e-01,  2.12930369e-01,
 -1.50346748e-17, -4.35218008e-16, -2.34914971e-18],
[-0.00000000e+00, -1.13634729e-16,  3.31737269e-17,
  2.03782917e-17, -4.80371507e-17, -5.10574161e-17,
  9.53585315e-17, -5.80622738e-17, -2.24765004e-16,
 -1.86747312e-16, -1.78034193e-16,  4.69241362e-16,
  5.93470267e-16,  5.18606846e-16, -1.42222563e-01,
  3.05682769e-01,  3.05682769e-01, -1.02590725e-01,
 -1.02590725e-01, -3.58054329e-01, -3.58054329e-01,
 -3.94732439e-01, -3.94732439e-01,  3.00911635e-01,
  3.00911635e-01,  1.06560527e-01,  1.06560527e-01,
 -1.23263775e-16, -6.53362345e-16, -6.48535461e-17],
[ 0.00000000e+00,  2.37438197e-16, -2.03598756e-16,
  4.89193059e-17,  8.05365288e-17, -6.17053949e-17,
  8.62892515e-18,  8.75270630e-17,  6.41313673e-17,
  1.62188764e-16, -5.07031490e-17,  4.34569671e-16,
  4.23398229e-16,  4.95206451e-16,  7.38958565e-01,
 -2.84064215e-02, -2.84064215e-02,  1.99406164e-01,
  1.99406164e-01,  1.53836324e-02,  1.53836324e-02,
  1.05321735e-01,  1.05321735e-01,  4.17344597e-01,
  4.17344597e-01,  2.99088569e-02,  2.99088569e-02,
  6.43474022e-17, -4.54451866e-16, -1.07250377e-16],
[ 0.00000000e+00,  9.17159453e-17, -6.40719927e-17,
  1.40478525e-16,  3.61353009e-17,  6.55893398e-17,
 -5.83609547e-19, -2.19386115e-16, -7.70983454e-18,
  9.40644405e-17,  8.37665765e-17, -5.55044958e-17,
 -9.02151703e-17, -1.88983788e-16,  8.92997367e-02,
```

```
       -1.70378728e-01,  -1.70378728e-01,  -4.52632837e-01,
       -4.52632837e-01,   2.62510579e-01,   2.62510579e-01,
       -7.51869990e-02,  -7.51869990e-02,   1.04797254e-01,
        1.04797254e-01,   4.20190468e-01,   4.20190468e-01,
       -6.45357414e-17,   1.27167910e-16,   2.26890377e-17],
     [ -5.77350269e-01,  -2.22044605e-16,   2.77555756e-17,
        0.00000000e+00,  -1.66533454e-16,  -1.11022302e-16,
        1.11022302e-16,  -1.24900090e-16,   4.85722573e-17,
        7.11236625e-17,  -8.19656842e-17,   5.77350269e-01,
        5.77350269e-01,   4.99600361e-16,  -1.52655666e-16,
        7.52869989e-16,   7.63278329e-16,  -1.76941795e-16,
       -9.71445147e-17,   5.62050406e-16,   6.52256027e-16,
        5.62050406e-16,   6.38378239e-16,  -4.97865638e-16,
       -3.88578059e-16,   2.60208521e-17,   2.08166817e-16,
       -1.15359111e-16,  -8.04911693e-16,  -1.87350135e-16]])
```

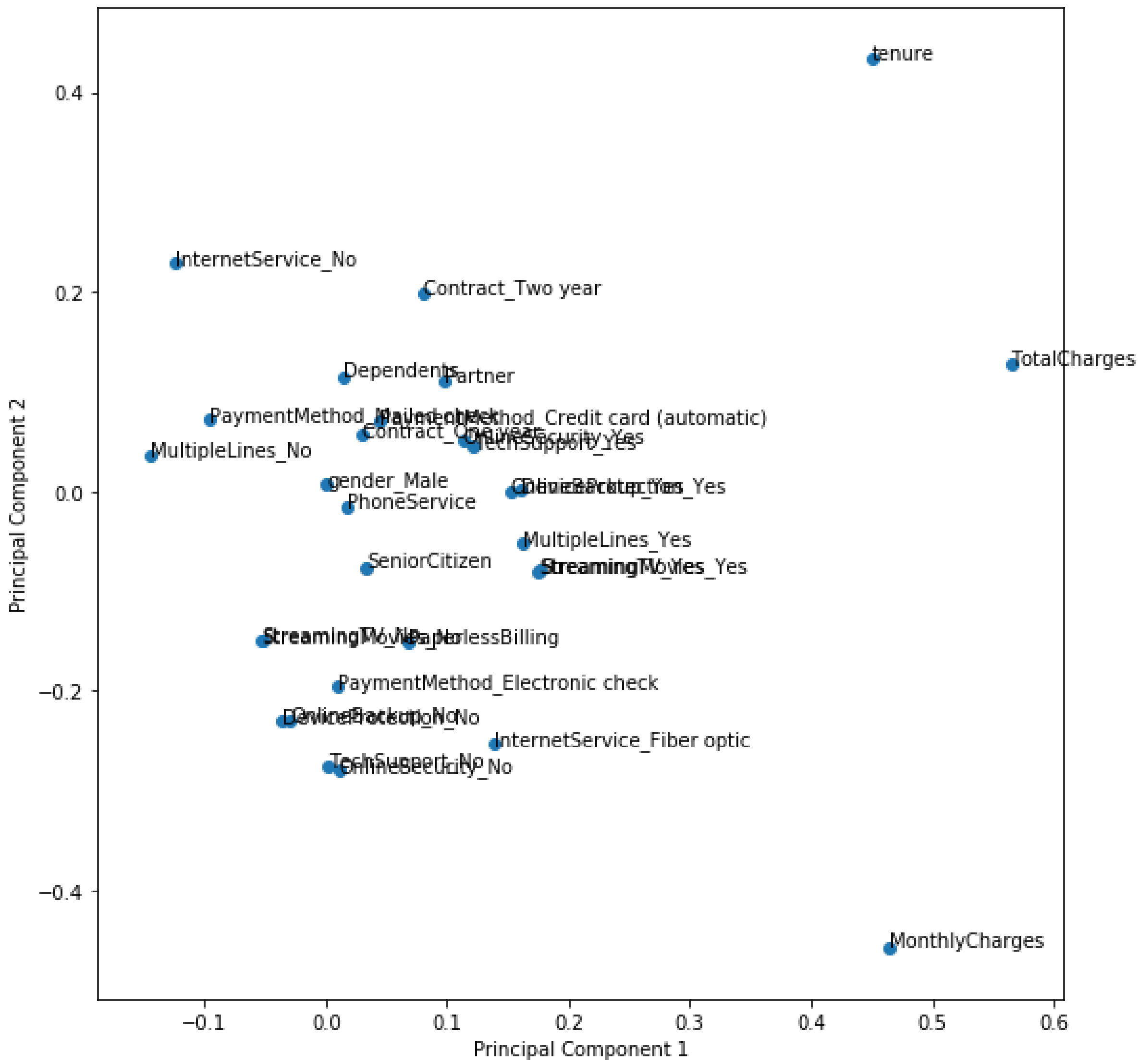
In [193]:

```
colnames = list(X_train.columns)
pcs_df = pd.DataFrame({'PC1':pca.components_[0],'PC2':pca.components_[1], 'Fea
ture':colnames})
pcs_df.head()
```

Out[193]:

	Feature	PC1	PC2
0	PhoneService	0.017611	-0.015823
1	PaperlessBilling	0.067324	-0.151382
2	SeniorCitizen	0.033356	-0.076039
3	Partner	0.098103	0.110808
4	Dependents	0.014514	0.115262

In [194]:
```
%matplotlib inline
fig = plt.figure(figsize = (8,8))
plt.scatter(pcs_df.PC1, pcs_df.PC2)
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
for i, txt in enumerate(pcs_df.Feature):
    plt.annotate(txt, (pcs_df.PC1[i],pcs_df.PC2[i]))
plt.tight_layout()
plt.show()
```



We see that the fist component is in the direction where the 'charges' variables are heavy

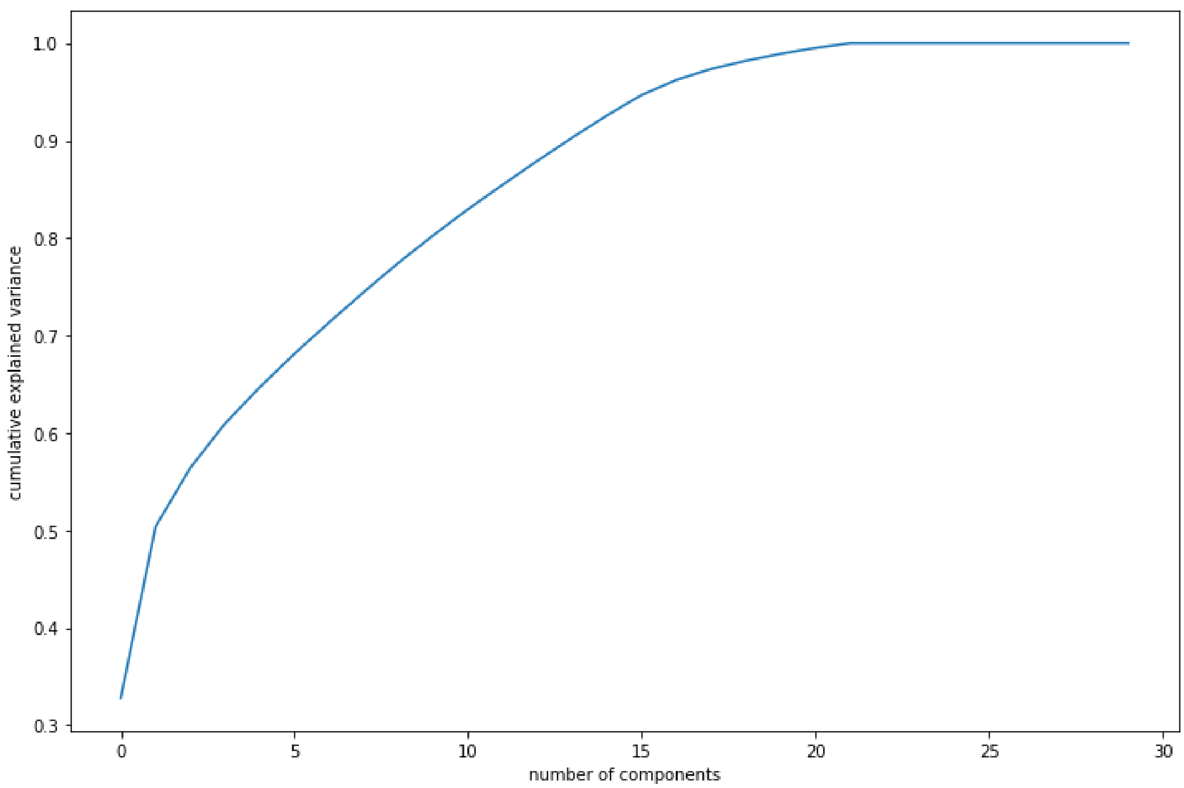
- These 3 components also have the highest of the loadings

Looking at the screeplot to assess the number of needed principal components

In [195]:
```
pca.explained_variance_ratio_
```

Out[195]:
```
array([  3.27847827e-01,   1.75765007e-01,   6.03450801e-02,
         4.55648825e-02,   3.71944481e-02,   3.44939196e-02,
         3.21953215e-02,   3.12085042e-02,   3.00721929e-02,
         2.81455178e-02,   2.67941203e-02,   2.51671298e-02,
         2.45802775e-02,   2.36931522e-02,   2.27788300e-02,
         2.10982912e-02,   1.53183204e-02,   1.12792045e-02,
         8.45904722e-03,   7.12080502e-03,   5.92788522e-03,
         4.90401247e-03,   4.62235942e-05,   2.05310042e-33,
         2.05310042e-33,   2.05310042e-33,   2.05310042e-33,
         2.05310042e-33,   2.05310042e-33,   2.18456080e-35])
```

In [196]:
```
#Making the screeplot - plotting the cumulative variance against the number of components
%matplotlib inline
fig = plt.figure(figsize = (12,8))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```



Looks like 16 components are enough to describe 95% of the variance in the dataset

- We'll choose 16 components for our modeling

In [197]:
```
#Using incremental PCA for efficiency - saves a lot of time on larger datasets
from sklearn.decomposition import IncrementalPCA
pca_final = IncrementalPCA(n_components=16)
```

Basis transformation - getting the data onto our PCs

In [198]:
```
df_train_pca = pca_final.fit_transform(X_train)
df_train_pca.shape
```

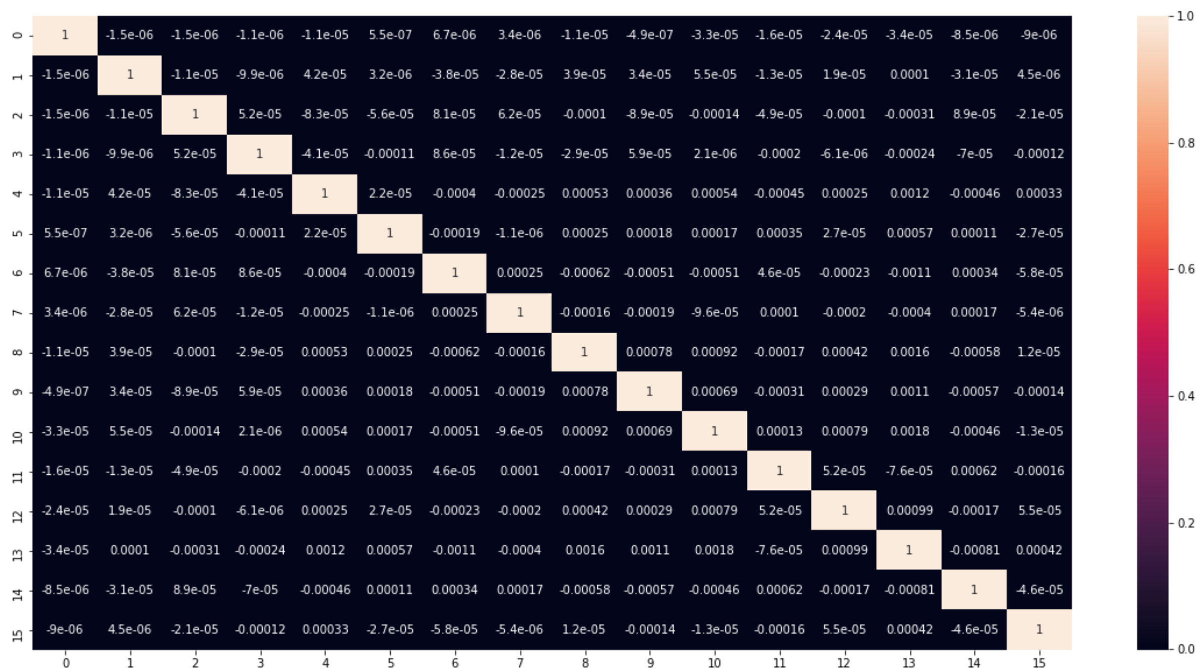
Out[198]:
```
(4922, 16)
```

Creating correlation matrix for the principal components - we expect little to no correlation

In [199]:
```
#creating correlation matrix for the principal components
corrmat = np.corrcoef(df_train_pca.transpose())
```

In [200]:
```
#plotting the correlation matrix
%matplotlib inline
plt.figure(figsize = (20,10))
sns.heatmap(corrmat,annot = True)
```

Out[200]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x29027e52320>
```



In [201]:
```
# 1s -> 0s in diagonals
corrmat_nodiag = corrmat - np.diagflat(corrmat.diagonal())
print("max corr:",corrmat_nodiag.max(), ", min corr: ", corrmat_nodiag.min(),)
# we see that correlations are indeed very close to 0
```

```
max corr: 0.00181652621921 , min corr:  -0.00107516919037
```

Indeed - there is no correlation between any two components! Good job, PCA!

- We effectively have removed multicollinearity from our situation, and our models will be much more stable

In [202]:
```
#Applying selected components to the test data - 16 components
df_test_pca = pca_final.transform(X_test)
df_test_pca.shape
```

Out[202]: (2110, 16)

Applying a logistic regression on our Principal Components

- We expect to get similar model performance with significantly lower features
- If we can do so, we would have done effective dimensionality reduction without losing any import information

In [203]:
```
#Training the model on the train data
from sklearn.linear_model import LogisticRegression
from sklearn import metrics

learner_pca = LogisticRegression()
model_pca = learner_pca.fit(df_train_pca,y_train)
```

Note

Note that we are fitting the original variable y with the transformed variables (principal components). This is not a problem becuase the transformation done in PCA is *linear*, which implies that you've only changed the way the new x variables are represented, though the nature of relationship between X and Y is still linear.

In [204]:
```
#Making prediction on the test data
pred_probs_test = model_pca.predict_proba(df_test_pca)[:,1]
"{:2.2}".format(metrics.roc_auc_score(y_test, pred_probs_test))
```

Out[204]: '0.83'

Impressive! The same result, without all the hard work on feature selection!

Why not take it a step further and get a little more 'unsupervised' in our approach? This time, we'll let PCA select the number of components basen on a variance cutoff we provide

In [205]:
```
pca_again = PCA(0.90)
```

In [206]:
```
df_train_pca2 = pca_again.fit_transform(X_train)
df_train_pca2.shape
# we see that PCA selected 14 components
```

Out[206]: (4922, 14)

In [207]:
```
#training the regression model
learner_pca2 = LogisticRegression()
model_pca2 = learner_pca2.fit(df_train_pca2,y_train)
```

In [208]:
```
df_test_pca2 = pca_again.transform(X_test)
df_test_pca2.shape
```

Out[208]: (2110, 14)

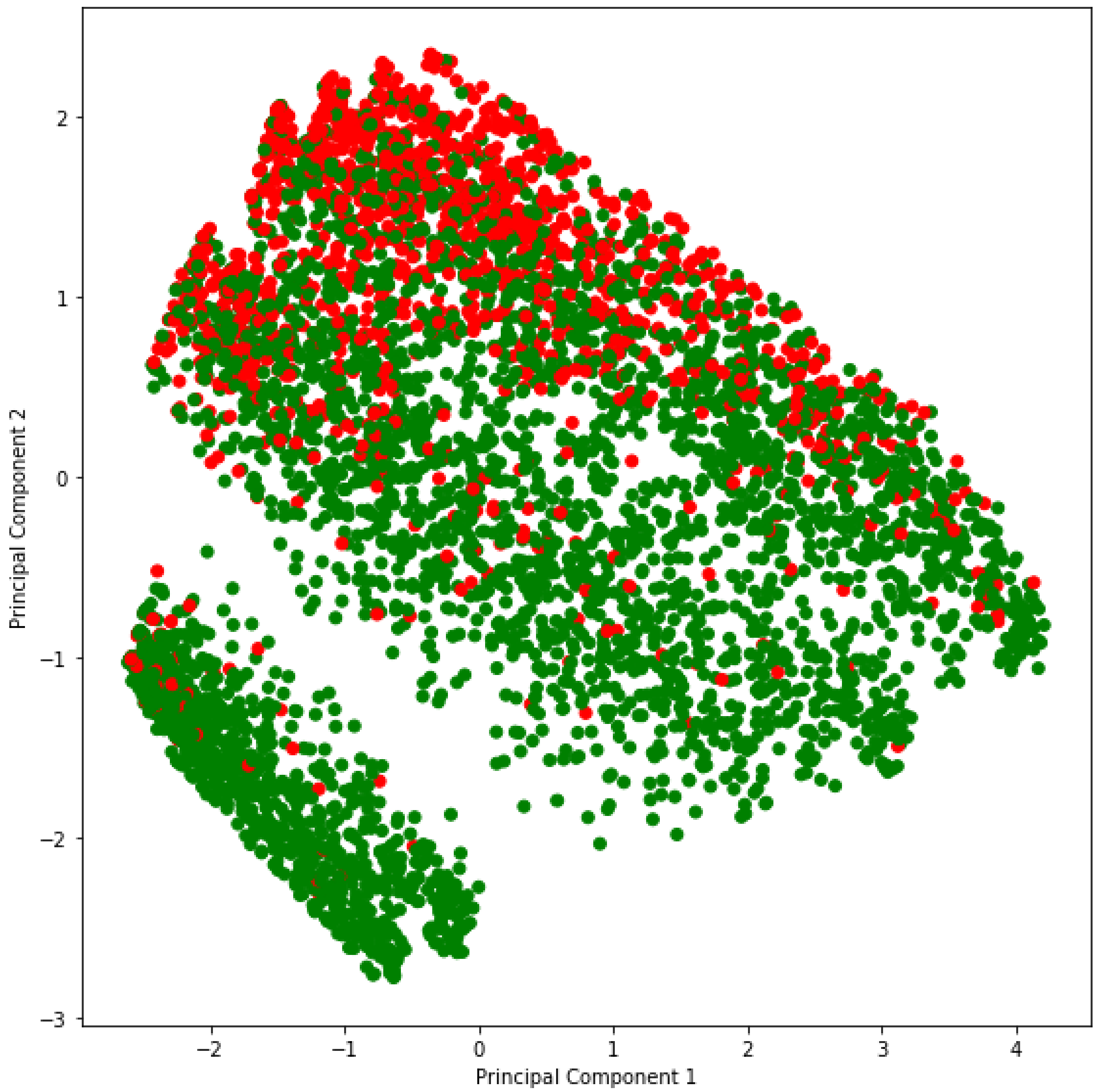
In [209]:
```
#Making prediction on the test data
pred_probs_test2 = model_pca2.predict_proba(df_test_pca2)[:,1]
"{:2.2f}".format(metrics.roc_auc_score(y_test, pred_probs_test2))
```

Out[209]: '0.83'

So there it is - a very similar result, without all the hassles. We have not only achieved dimensionality reduction, but also saved a lot of effort on feature selection.

Before closing, let's also visualize the data to see if we can spot any patterns

In [210]:
```
%matplotlib inline
fig = plt.figure(figsize = (8,8))
plt.scatter(df_train_pca[:,0], df_train_pca[:,1], c = y_train.map({0:'green',1
:'red'}))
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.tight_layout()
plt.show()
```



Looks like there is a good amount of separation in 2D, but probably not enough

Let's look at it in 3D, and we expect spread to be better (dimensions of variance, remember?)

In [211]:
```
%matplotlib notebook
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(8,8))
ax = Axes3D(fig)
# ax = plt.axes(projection='3d')
ax.scatter(df_train_pca[:,2], df_train_pca[:,0], df_train_pca[:,1], c=y_train.
map({0:'green',1:'red'}))
```

```
Out[211]: <mpl_toolkits.mplot3d.art3d.Path3DCollection at 0x29028f9a320>
```

So let's try building the model with just 3 principal components!

In [212]:
```
pca_last = PCA(n_components=3)
df_train_pca3 = pca_last.fit_transform(X_train)
df_test_pca3 = pca_last.transform(X_test)
df_test_pca3.shape
```

Out[212]: (2110, 3)

In [213]:
```
#training the regression model
learner_pca3 = LogisticRegression()
model_pca3 = learner_pca3.fit(df_train_pca3,y_train)
#Making prediction on the test data
pred_probs_test3 = model_pca3.predict_proba(df_test_pca3)[:,1]
"{:2.2f}".format(metrics.roc_auc_score(y_test, pred_probs_test3))
```

Out[213]: '0.82'

0.82! Isn't that just amazing!