Telecom Churn - Assignment | Code Document

Author - Jatin Arora

arorjati@gmail.com

Problem Statement

Problem statement – In telecom domain where the customer acquisition has higher cost than customer retention and where there is a rampant price war, it becomes important to predict and profile the customer behavior and predict them so that offers can be rolled out to retain them prior to switching. Data consist of the various behavior of customers and the last column states if the customer is still with the existing telecom company or not. Objectives – With the help of data visualization & descriptive stats help us understand the current state and various factors (or combination of factors) which is contributing to customer churn. Develop a prediction algorithm to predict customer churn. Explain how you evaluated the model and which metric you chose for your model evaluation and why? What extra features could have made the model better and give a better explanation of the factors leading to churn Based on the descriptive analytics and model results help us to come up with a strategy to arrest customer churn. Deliverables- Code (ipython notebook or RMarkdown notebook) PPT explaining the overall approach(starting from current state to why your model should be used) and the strategy of arresting customer churn

Approach

I am approaching this exercise in stages:

1. Data Import
2. Data Analysis - EDA
    - Data Preparation
    - Preliminary Analysis
    - Univariate Analysis
    - Multivariate Analysis
3. Feature Engineering
4. Modelling -Mental Model Preparation
    - Unsupervised Approach
    - Supervised Approach
5. Validation & Model Selection

Stage 1

Libarary Import

In [1]:
```
#Import Libararies
import numpy as np
import pandas as pd
import os
import sklearn

from matplotlib import pyplot as plt
import seaborn as sns
from scipy import stats
import seaborn as sns

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

Reading Data

Original Data Columns: ['Id', 'State', 'Account length', 'Area code', 'International plan', 'Voice mail plan', 'Number vmail messages', 'Total day minutes', 'Total day calls', 'Total day charge', 'Total eve minutes', 'Total eve calls', 'Total eve charge', 'Total night minutes', 'Total night calls', 'Total night charge', 'Total intl minutes', 'Total intl calls', 'Total intl charge', 'Customer service calls', 'Churn'] Updated Data Columns: ['id', 'state', 'acc_len', 'area_code', 'int_plan', 'voice_mail_plan', 'num_vmail_msg', 'total_day_minutes', "total_day_calls", "total_day_charge", "total_eve_min", "total_eve_calls", "total_eve_charge", "total_night_min", "total_night_calls", "total_night_charge", "total_int_min", "total_int_calls", "total_int_charge", "customer_service_calls", "churn"]

In [2]:
```
data_cols = ['id', 'state', 'acc_len', 'area_code', 'int_plan', 'voice_mail_pl
an', 'num_vmail_msg', 'total_day_minutes',
"total_day_calls", "total_day_charge", "total_eve_min", "total_eve_calls", "to
tal_eve_charge", "total_night_min",
"total_night_calls", "total_night_charge", "total_int_min", "total_int_calls",
 "total_int_charge", "customer_service_calls",
"churn"]

num_data = ['acc_len', 'num_vmail_msg', 'total_day_minutes',
"total_day_calls", "total_day_charge", "total_eve_min", "total_eve_calls", "to
tal_eve_charge", "total_night_min",
"total_night_calls", "total_night_charge", "total_int_min", "total_int_calls",
 "total_int_charge", "customer_service_calls"]

cat_data = ['state', 'area_code', 'int_plan', 'voice_mail_plan']

target =['churn']
```

In [3]:
```
data_link = "/local/home/arorjati/"
data = "data.csv"
data = pd.read_csv(data_link+data, header = 0, names = data_cols)
data.head()
```

Out[3]:

	id	state	acc_len	area_code	int_plan	voice_mail_plan	num_vmail_msg	total_da
0	CUST-1	KS	128	415	No	Yes	25	265.1
1	CUST-2	OH	107	415	No	Yes	26	161.6
2	CUST-3	NJ	137	415	No	No	0	243.4
3	CUST-4	OH	84	408	Yes	No	0	299.4
4	CUST-5	OK	75	415	Yes	No	0	166.7

5 rows × 21 columns

Stage 2

2.1 Data Preparation

Data Munging

In [4]:
```
# checking Null Values or missing values
data.isnull().sum()
```

Out[4]:
```
id                      0
state                   0
acc_len                 0
area_code               0
int_plan                0
voice_mail_plan         0
num_vmail_msg           0
total_day_minutes       0
total_day_calls         0
total_day_charge        0
total_eve_min           0
total_eve_calls         0
total_eve_charge        0
total_night_min         0
total_night_calls       0
total_night_charge      0
total_int_min           0
total_int_calls         0
total_int_charge        0
customer_service_calls  0
churn                   0
dtype: int64
```

In [5]:
```
#data shape
print(data.shape) #(3333, 21)

#unique customers
print(len(data['id'].unique())) #3333

print(data.groupby('churn').size().reset_index)
# False    2850
# True      483

#unique int_plan
print(len(data['int_plan'].unique())) #2

#unique voice_mail_plan
print(len(data['voice_mail_plan'].unique())) #2

data['churn'].unique()
```

```
(3333, 21)
3333
<bound method Series.reset_index of churn
False    2850
True      483
dtype: int64>
2
2
```

Out[5]:
```
array([False,  True])
```

In [6]:
```
#converting categorical to numerical data
data['int_plan'] = np.where(data['int_plan'] == "No", 0,1)
data['voice_mail_plan'] = np.where(data['voice_mail_plan'] == "No", 0,1)
```

2.2 Preliminary Analysis

In [7]:
```
#data at glance
describe_data = data.describe()
describe_data.to_csv("describe_data.csv")
describe_data.head()
```

Out[7]:

	acc_len	area_code	int_plan	voice_mail_plan	num_vmail_msg	total_c
count	3333.000000	3333.000000	3333.000000	3333.000000	3333.000000	3333.0
mean	101.064806	437.182418	0.096910	0.276628	8.099010	179.77
std	39.822106	42.371290	0.295879	0.447398	13.688365	54.467
min	1.000000	408.000000	0.000000	0.000000	0.000000	0.0000
25%	74.000000	408.000000	0.000000	0.000000	0.000000	143.70

In [8]:
```
#data w.r.t the target variable
churn_describe = data.groupby('churn').describe()
churn_describe.to_csv("churn_describe.csv")
churn_describe.head()
```

Out[8]:

	acc_len								area_code		..
	count	**mean**	**std**	**min**	**25%**	**50%**	**75%**	**max**	**count**	**mean**	..
churn											
False	2850.0	100.793684	39.88235	1.0	73.0	100.0	127.0	243.0	2850.0	437.074737	..
True	483.0	102.664596	39.46782	1.0	76.0	103.0	127.0	225.0	483.0	437.817805	..

2 rows × 144 columns

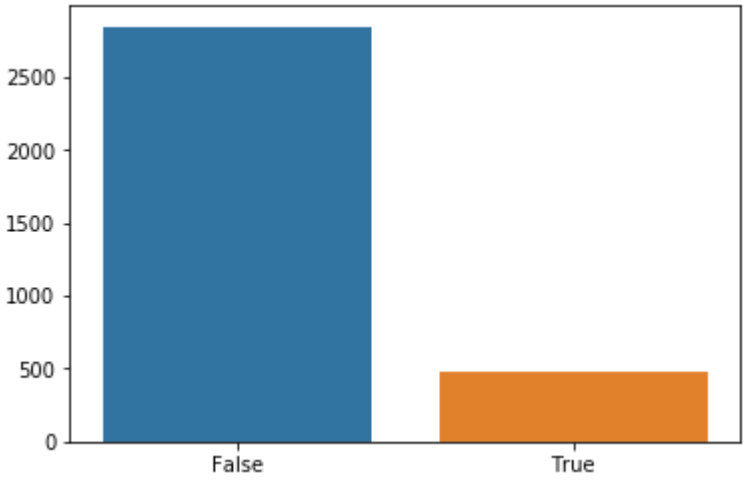
2.3 Univariate Analysis

In [9]:
```
data.columns
```

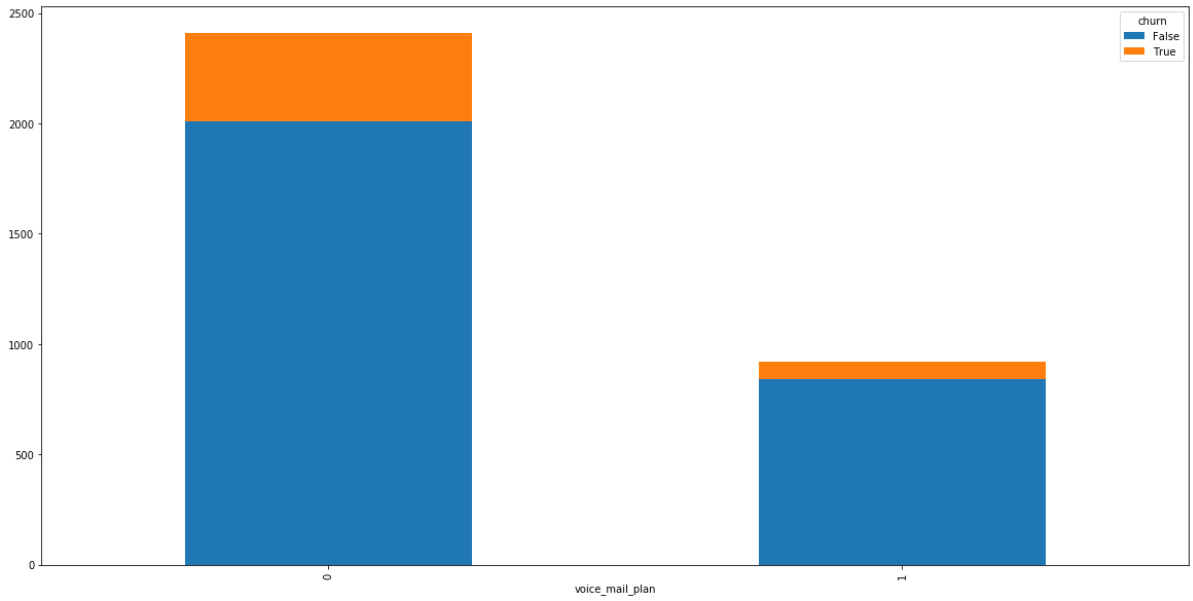
Out[9]:
```
Index(['id', 'state', 'acc_len', 'area_code', 'int_plan', 'voice_mail_plan',
       'num_vmail_msg', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_min', 'total_eve_calls',
       'total_eve_charge', 'total_night_min', 'total_night_calls',
       'total_night_charge', 'total_int_min', 'total_int_calls',
       'total_int_charge', 'customer_service_calls', 'churn'],
      dtype='object')
```

Plots

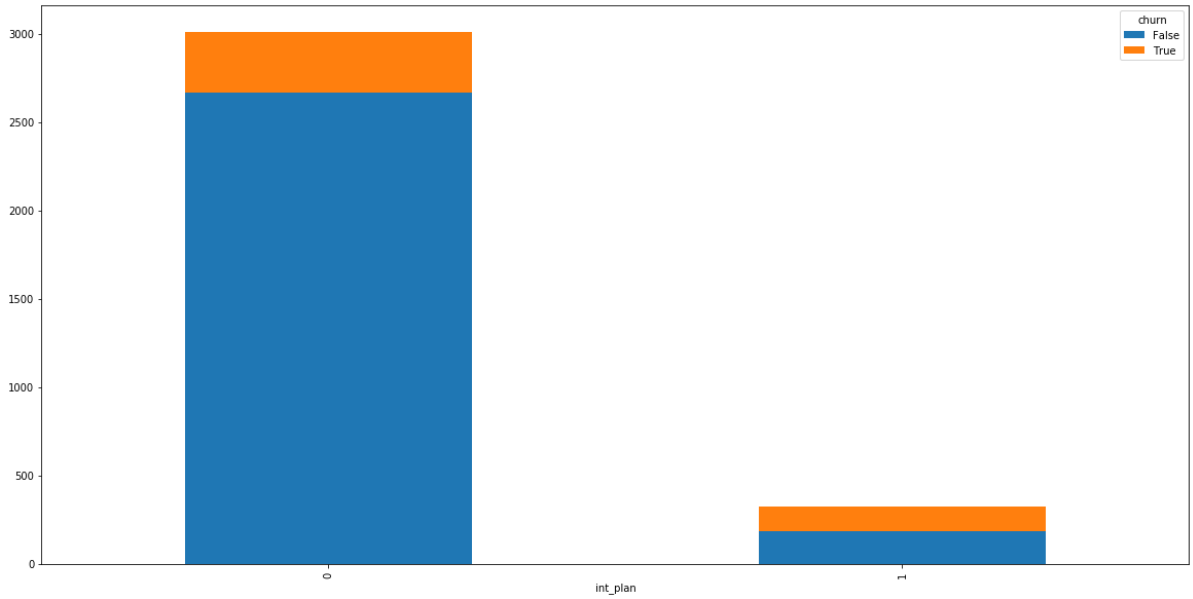
In [10]:
```
sns.barplot(data["churn"].value_counts().index, data["churn"].value_counts().v
alues)
plt.savefig("churn.png")
#class - imbalance
```



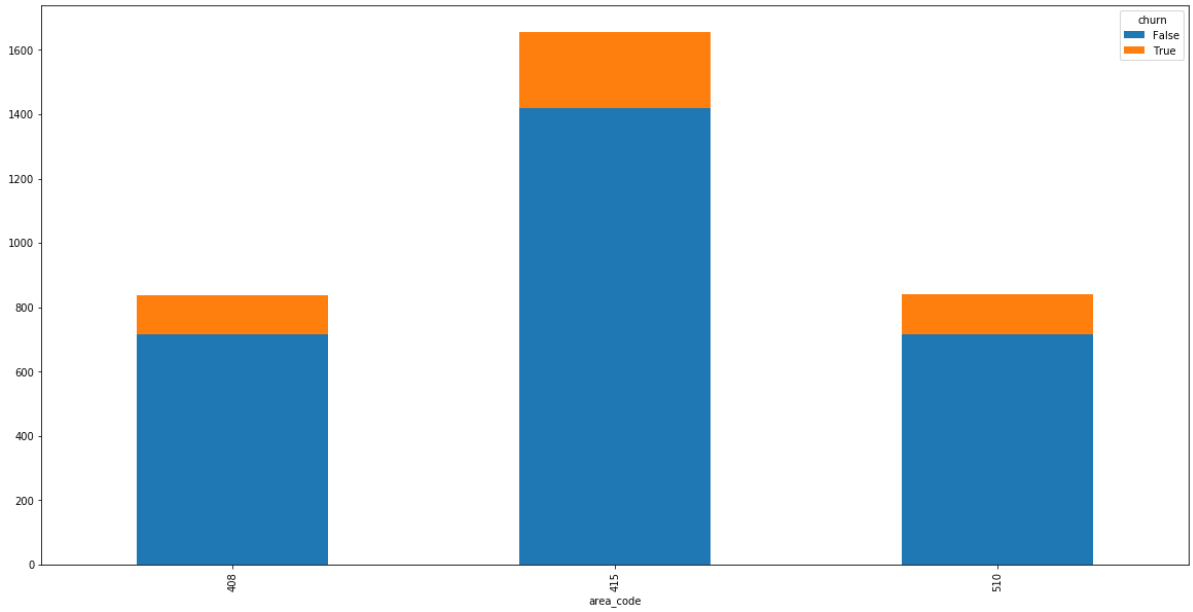
In [11]:
```
data.groupby(["voice_mail_plan", "churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(20,10))

pd.crosstab(data['voice_mail_plan'], data['churn'])

plt.savefig("v_mail_churn.png")
```



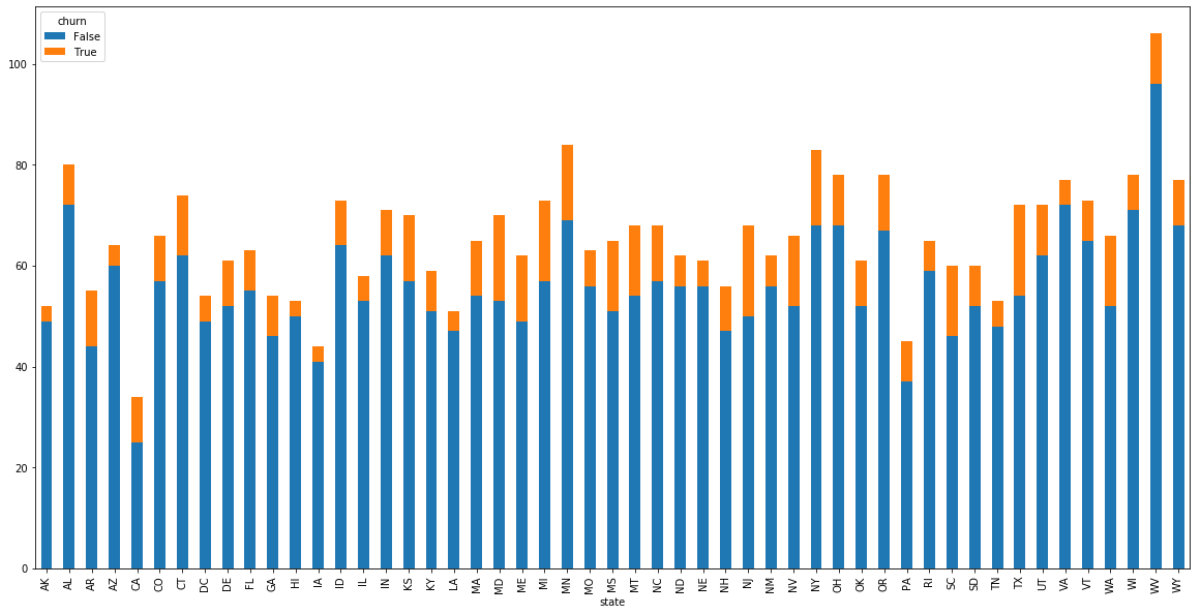
In [12]:
```
data.groupby(["int_plan", "churn"]).size().unstack().plot(kind='bar', stacked=True, figsize=(20,10))

pd.crosstab(data['int_plan'], data['churn'])
plt.savefig("int_plan_churn.png")
```



In [13]:
```
data.groupby(["area_code", "churn"]).size().unstack().plot(kind='bar', stacked
=True, figsize=(20,10))

pd.crosstab(data['area_code'], data['churn'])
plt.savefig("area_code_churn.png")
```

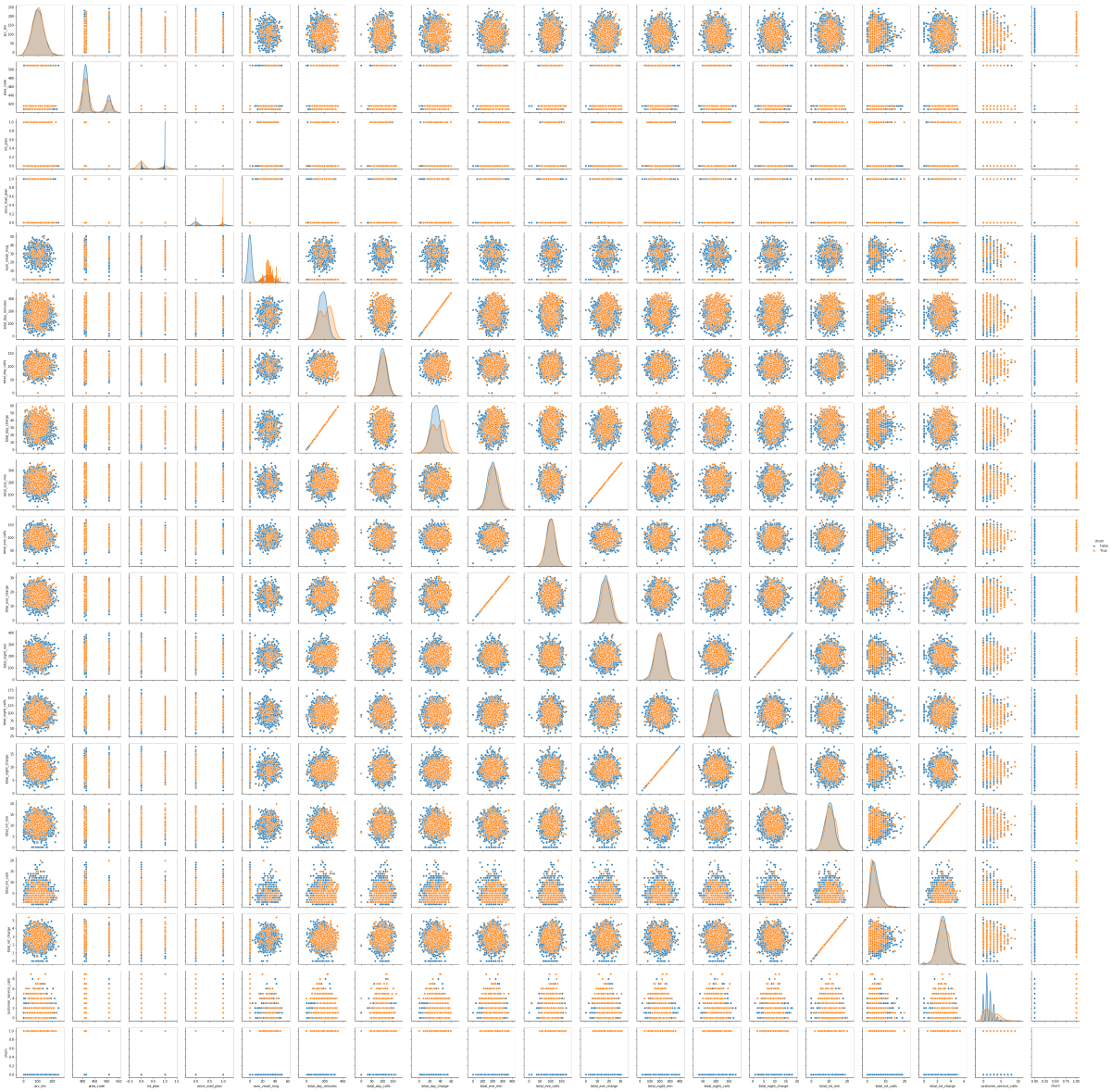


In [14]:
```
data.groupby(["state", "churn"]).size().unstack().plot(kind='bar', stacked=Tru
e, figsize=(20,10))

pd.crosstab(data['state'], data['churn']).transpose()
plt.savefig("state_churn.png")
```

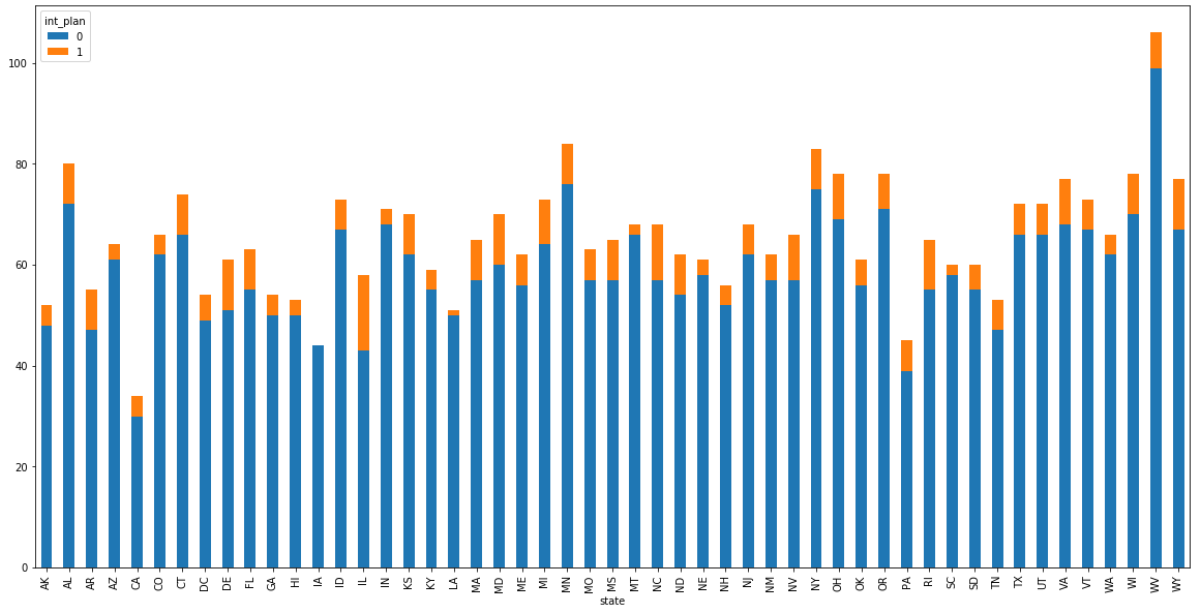


2.4 Multivariate Analysis

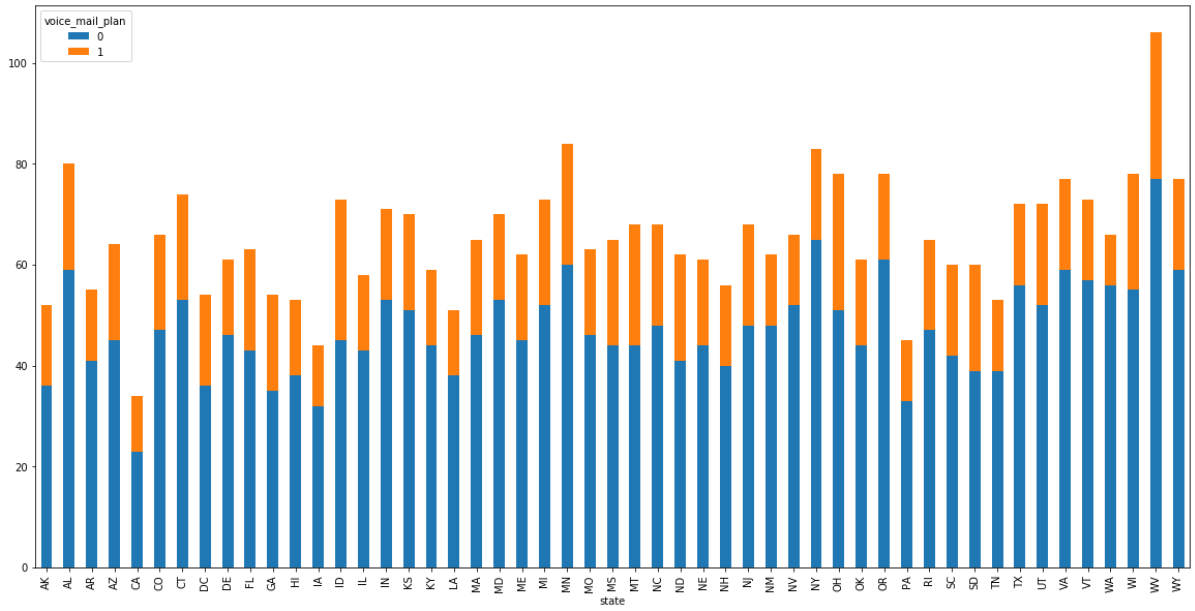
In [15]:
```
sns.pairplot(data, hue = 'churn')
plt.savefig("corr_1.png")
```



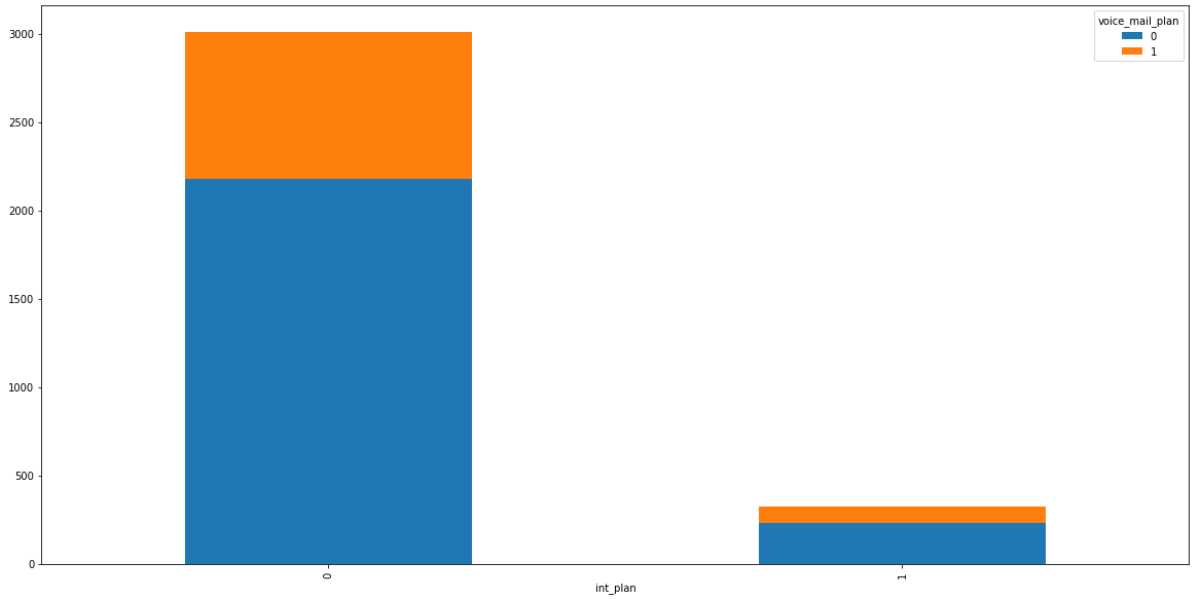
In [16]:
```
data.groupby(["state", "int_plan"]).size().unstack().plot(kind='bar', stacked=True, figsize=(20,10))
plt.savefig('state__int_plan.png')
```



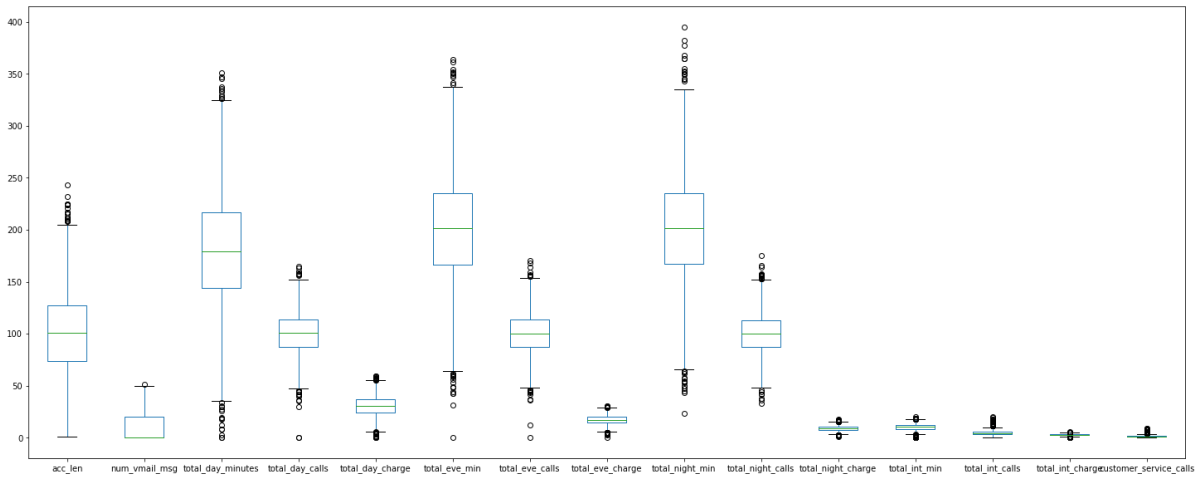
In [17]:
```
data.groupby(["state", "voice_mail_plan"]).size().unstack().plot(kind='bar', stacked=True, figsize=(20,10))
plt.savefig('state__voice_mail_plan.png')
```



In [18]:
```
data.groupby(["int_plan", "voice_mail_plan"]).size().unstack().plot(kind='bar'
, stacked=True, figsize=(20,10))
plt.savefig("int_vmail.png")
```



In [19]:
```
#box-whisker plot
data[num_data].plot(kind='box', subplots=False, figsize=(25,10))
plt.savefig("box_plot.png")
plt.show()
```



In [20]:
```
data.corr()['churn'].sort_values(ascending=False)
```

Out[20]:
```
churn                     1.000000
int_plan                  0.259852
customer_service_calls    0.208750
total_day_minutes         0.205151
total_day_charge          0.205151
total_eve_min             0.092796
total_eve_charge          0.092786
total_int_charge          0.068259
total_int_min             0.068239
total_night_charge        0.035496
total_night_min           0.035493
total_day_calls           0.018459
acc_len                   0.016541
total_eve_calls           0.009233
area_code                 0.006174
total_night_calls         0.006141
total_int_calls          -0.052844
num_vmail_msg            -0.089728
voice_mail_plan          -0.102148
Name: churn, dtype: float64
```

Stage 3. Feature Engineering

In [21]:
```
data.columns
```

Out[21]:
```
Index(['id', 'state', 'acc_len', 'area_code', 'int_plan', 'voice_mail_plan',
       'num_vmail_msg', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_min', 'total_eve_calls',
       'total_eve_charge', 'total_night_min', 'total_night_calls',
       'total_night_charge', 'total_int_min', 'total_int_calls',
       'total_int_charge', 'customer_service_calls', 'churn'],
      dtype='object')
```

In [22]:
```
dummies = pd.get_dummies(data['area_code'], prefix="area_code")
dummies.head()
data = pd.concat([data,dummies], axis=1)
data.head()
```

Out[22]:

	id	state	acc_len	area_code	int_plan	voice_mail_plan	num_vmail_msg	total_da
0	CUST-1	KS	128	415	0	1	25	265.1
1	CUST-2	OH	107	415	0	1	26	161.6
2	CUST-3	NJ	137	415	0	0	0	243.4
3	CUST-4	OH	84	408	1	0	0	299.4
4	CUST-5	OK	75	415	1	0	0	166.7

5 rows × 24 columns

In [23]:
```
num_data = ['acc_len', 'prem_customer','int_plan', 'voice_mail_plan','num_vmail_msg', 'total_day_minutes', 'total_day_calls',
       'total_day_charge', 'total_eve_min', 'total_eve_calls',
       'total_eve_charge', 'total_night_min', 'total_night_calls',
       'total_night_charge', 'total_int_min', 'total_int_calls',
       'total_int_charge', 'customer_service_calls',
       'day_avg_charge', 'eve_avg_charge', 'night_avg_charge', 'total_calls',
       'total_charge', 'total_min', 'day_avg_charge[data]',
       'eve_avg_charge[data]', 'night_avg_charge[data]',
       'day_avg_charge[data_state]', 'eve_avg_charge[data_state]',
       'night_avg_charge[data_state]', 'day_avg_charge[data_area_code]',
       'eve_avg_charge[data_area_code]', 'night_avg_charge[data_area_code]',
       'deviation_day_charge_state', 'deviation_eve_charge_state',
       'deviation_night_charge_state', 'deviation_day_charge_area_code',
       'deviation_eve_charge_area_code', 'cus_dev_day_charge',
       'cus_dev_eve_charge', 'cus_dev_night_charge', 'area_code_408', 'area_code_415', 'area_code_510']

cat_data = ['id', 'state']

target_data = ['churn']
```

In [24]:
```
#feature generation by business intelligence

#defining the customer as premium customer if he/she is availing bpth the inte
rnation plan and voice mail plan
data['prem_customer'] = np.where((data['int_plan'] == 1) & (data['voice_mail_p
lan'] == 1), 1,0)
#calculating the day average charge for day calls
data['day_avg_charge'] = data["total_day_charge"]/data["total_day_calls"]
#calculating the day average charge for evening calls
data['eve_avg_charge'] = data["total_eve_charge"]/data["total_eve_calls"]
#calculating the day average charge for night calls
data['night_avg_charge'] = data["total_night_charge"]/data["total_night_calls"
]

#total calls made by the customer
data['total_calls']= data['total_day_calls'] + data['total_eve_calls'] + data[
'total_night_calls'] + data['total_int_calls'] + data['customer_service_calls'
]

#total charge to the customer
data['total_charge']= data['total_day_charge'] + data['total_eve_charge'] + da
ta['total_night_charge'] + data['total_int_charge']

#total call minutes
data['total_min']= data['total_day_minutes'] + data['total_eve_min'] + data['t
otal_night_min'] + data['total_int_min']


data['day_avg_charge[data]'] = data["total_day_charge"].mean()
#calculating the day average charge for evening calls
data['eve_avg_charge[data]'] = data["total_eve_charge"].mean()
#calculating the day average charge for night calls
data['night_avg_charge[data]'] = data["total_night_charge"].mean()

data = data.merge(data.groupby(["state"])["total_day_charge"].mean().reset_ind
ex().rename(columns = {'state': 'state',"total_day_charge":"day_avg_charge[dat
a_state]"})
                                                , on = ['state'])

data = data.merge(data.groupby(["state"])["total_eve_charge"].mean().reset_ind
ex().rename(columns = {'state': 'state',"total_eve_charge":"eve_avg_charge[dat
a_state]"})
                                                , on = ['state'])

data = data.merge(data.groupby(["state"])["total_night_charge"].mean().reset_i
ndex().rename(columns = {'state': 'state',"total_night_charge":"night_avg_char
ge[data_state]"})
                                                , on = ['state'])


data = data.merge(data.groupby(["area_code"])["total_day_charge"].mean().reset
_index().rename(columns = {'area_code': 'area_code',"total_day_charge":"day_av
g_charge[data_area_code]"})
                                                , on = ['area_code'])

data = data.merge(data.groupby(["area_code"])["total_eve_charge"].mean().reset
```

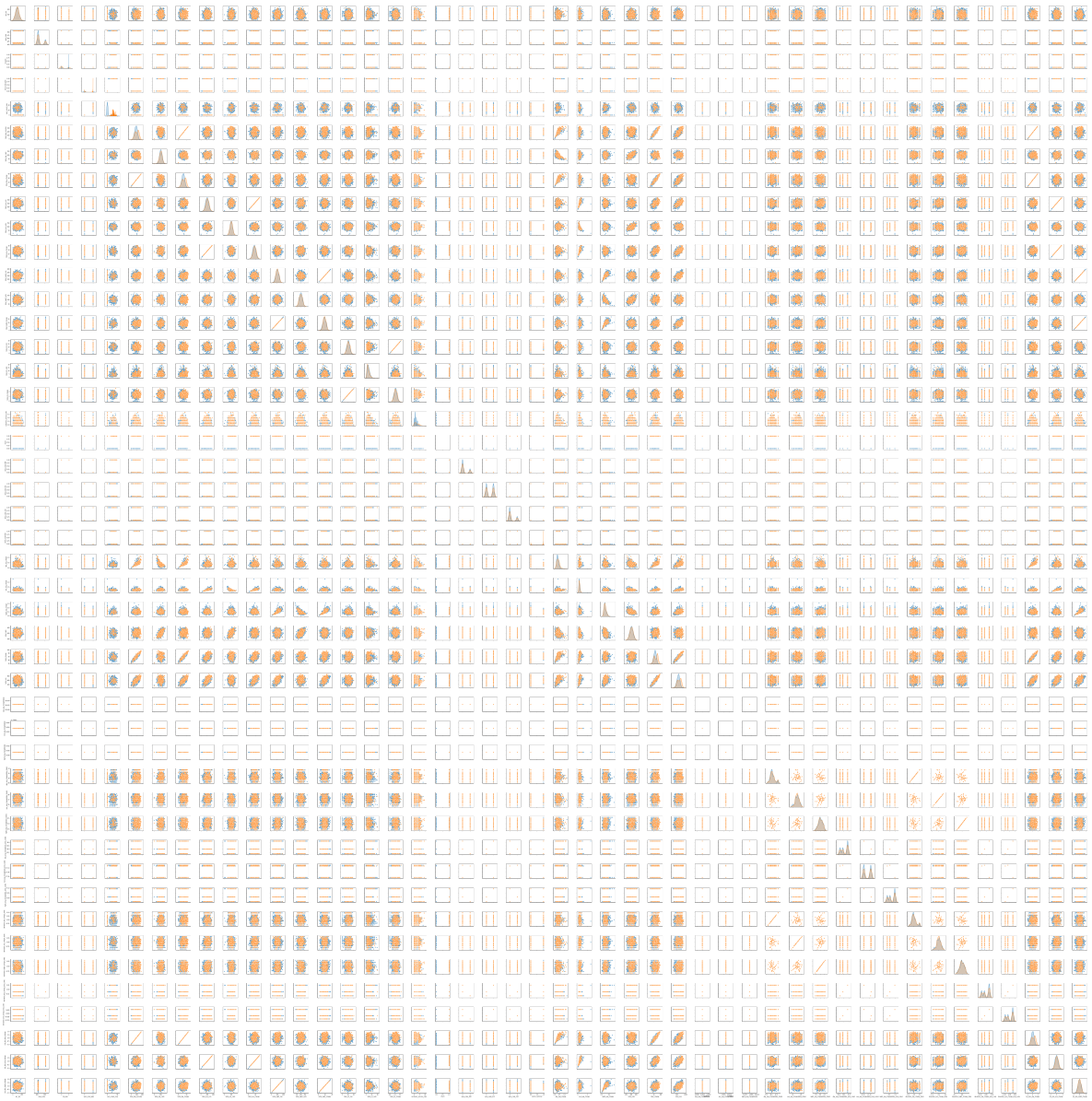
```
_index().rename(columns = {'area_code': 'area_code',"total_eve_charge":"eve_av
g_charge[data_area_code]"})
                                                    , on = ['area_code'])

data = data.merge(data.groupby(["area_code"])["total_night_charge"].mean().res
et_index().rename(columns = {'area_code': 'area_code',"total_night_charge":"ni
ght_avg_charge[data_area_code]"})
                                                    , on = ['area_code'])
```

In [25]:
```
data["deviation_day_charge_state"] = (data["day_avg_charge[data_state]"]/data[
"day_avg_charge[data]"])-1
data["deviation_eve_charge_state"] = (data["eve_avg_charge[data_state]"]/data[
"eve_avg_charge[data]"])-1
data["deviation_night_charge_state"] = (data["night_avg_charge[data_state]"]/d
ata["night_avg_charge[data]"])-1

data["deviation_day_charge_area_code"] = (data["day_avg_charge[data_area_cod
e]"]/data["day_avg_charge[data]"])-1
data["deviation_eve_charge_area_code"] = (data["eve_avg_charge[data_area_cod
e]"]/data["eve_avg_charge[data]"])-1
data["deviation_eve_charge_area_code"] = (data["night_avg_charge[data_area_cod
e]"]/data["night_avg_charge[data]"])-1
```

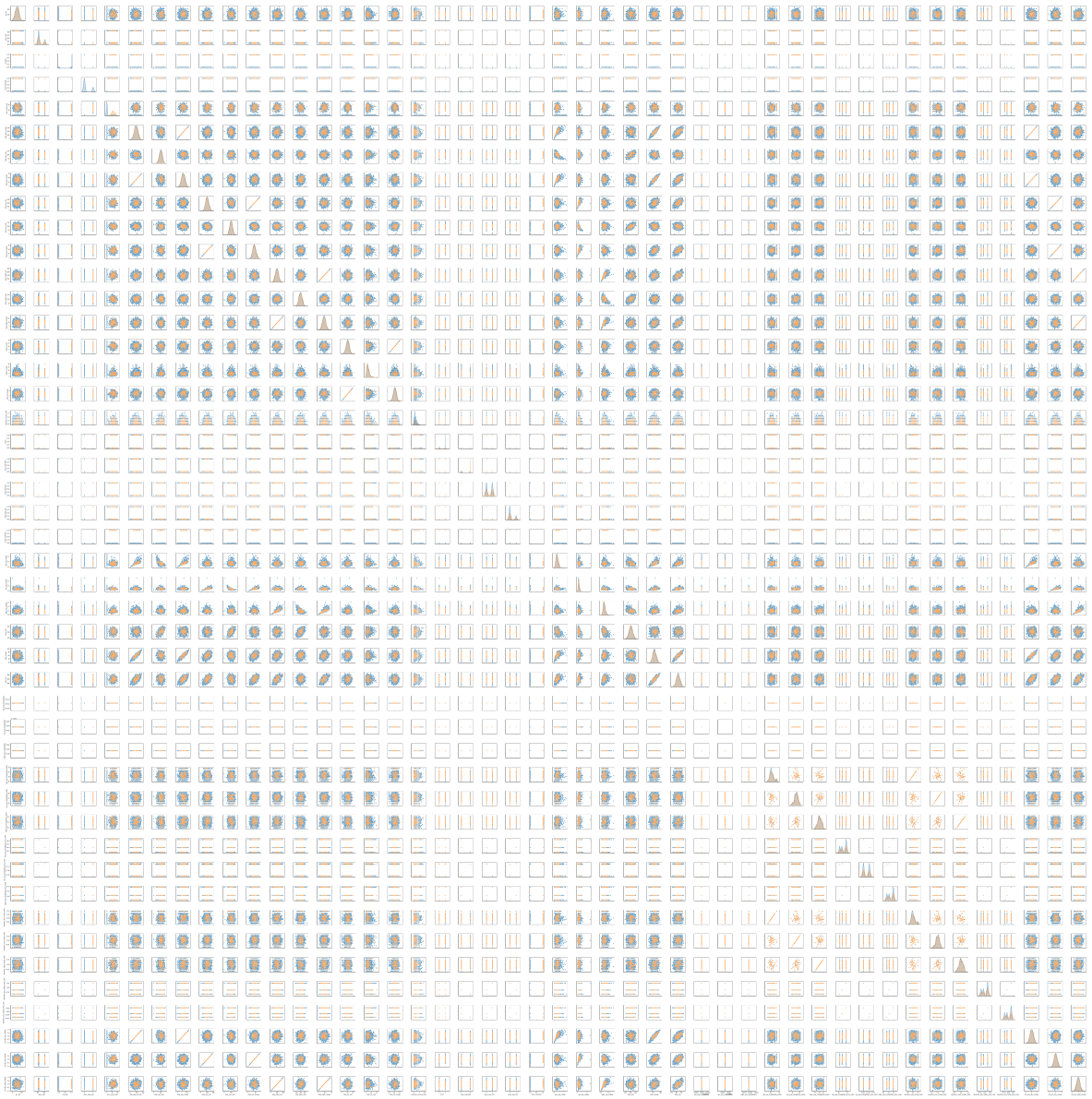
In [26]:
```
data['cus_dev_day_charge'] = (data['total_day_charge']/data['day_avg_charge[da
ta]'])-1
data['cus_dev_eve_charge'] = (data['total_eve_charge']/data['eve_avg_charge[da
ta]'])-1
data['cus_dev_night_charge'] = (data['total_night_charge']/data['night_avg_cha
rge[data]'])-1
```

In [27]:
```
data.to_csv('featured_data.csv')
```

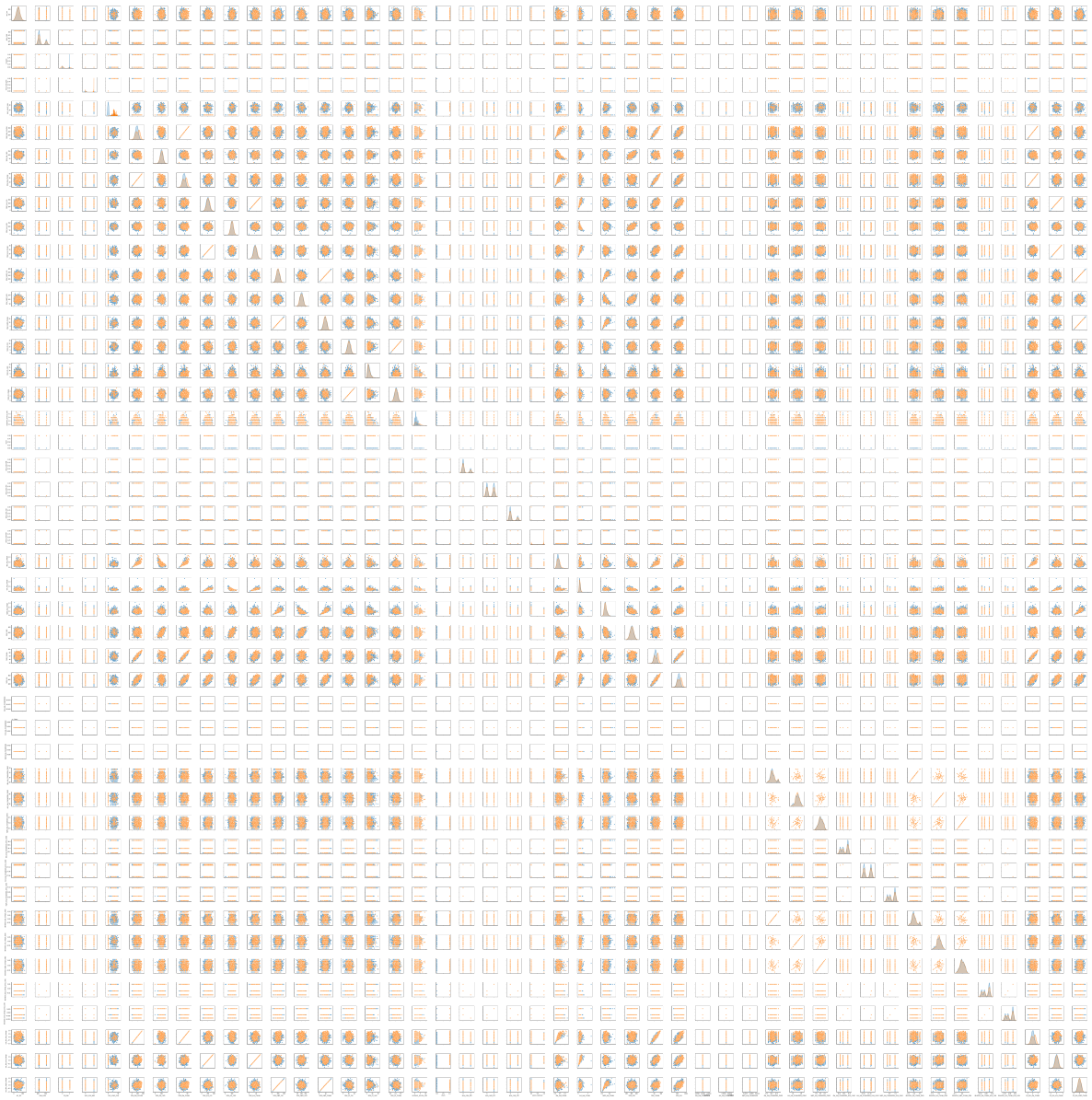
In [28]:
```
sns.pairplot(data, hue = 'churn')
plt.savefig("corr_2.png")
```



In [29]:
```
sns.pairplot(data, hue = 'prem_customer')
plt.savefig("corr3.png")
```



In [30]:
```
sns.pairplot(data, hue = 'churn')
plt.savefig("corr4.png")
```



In [31]:
```
data.groupby(['area_code']).describe()
```

Out[31]:

	acc_len								area_code_408	
	count	mean	std	min	25%	50%	75%	max	count	mean
area_code										
408	838.0	101.877088	40.393825	1.0	74.0	102.0	130.0	232.0	838.0	1.0
415	1655.0	101.068882	39.256637	1.0	74.0	100.0	126.0	225.0	1655.0	0.0
510	840.0	100.246429	40.381347	3.0	72.0	100.0	128.0	243.0	840.0	0.0

3 rows × 352 columns

In [32]:
```
data.groupby(['state']).describe()
```

Out[32]:

	acc_len								area_code	
	count	mean	std	min	25%	50%	75%	max	count	mean
state										
AK	52.0	97.192308	38.300176	1.0	60.50	102.0	126.25	177.0	52.0	438.692308
AL	80.0	98.025000	41.539969	8.0	72.75	95.0	121.25	200.0	80.0	430.625000
AR	55.0	96.981818	37.007252	5.0	76.50	98.0	116.00	185.0	55.0	439.254545
AZ	64.0	100.468750	32.263778	12.0	82.75	97.0	122.50	192.0	64.0	432.656250
CA	34.0	99.235294	33.738800	33.0	77.50	95.5	119.00	185.0	34.0	441.500000
CO	66.0	102.545455	37.815822	22.0	74.50	103.5	127.25	192.0	66.0	429.621212
CT	74.0	99.040541	42.682465	3.0	72.25	99.0	124.75	212.0	74.0	429.608108
DC	54.0	105.722222	40.380463	11.0	77.00	105.5	134.50	217.0	54.0	436.055556
DE	61.0	101.918033	36.480266	2.0	80.00	99.0	122.00	224.0	61.0	439.983607
FL	63.0	109.571429	32.525283	22.0	91.50	109.0	130.00	185.0	63.0	443.825397
GA	54.0	103.592593	41.512680	18.0	73.25	98.0	131.75	202.0	54.0	444.722222
HI	53.0	105.471698	39.992116	3.0	87.00	105.0	132.00	190.0	53.0	427.358491
IA	44.0	94.318182	41.270955	1.0	63.75	90.5	126.50	197.0	44.0	448.272727
ID	73.0	104.493151	37.965673	24.0	82.00	99.0	122.00	224.0	73.0	439.876712
IL	58.0	100.517241	40.041313	16.0	71.75	90.5	133.25	183.0	58.0	437.758621
IN	71.0	100.183099	35.821506	7.0	76.50	100.0	118.00	205.0	71.0	439.985915
KS	70.0	106.785714	36.029735	24.0	80.25	110.0	129.50	186.0	70.0	442.300000
KY	59.0	91.830508	31.779495	21.0	71.50	91.0	112.50	172.0	59.0	432.542373
LA	51.0	108.235294	41.334048	9.0	88.00	111.0	141.00	179.0	51.0	433.705882
MA	65.0	91.861538	35.411808	13.0	67.00	93.0	121.00	166.0	65.0	429.953846
MD	70.0	105.271429	39.035072	18.0	79.00	106.0	129.75	204.0	70.0	433.757143
ME	62.0	97.629032	46.705998	19.0	63.00	98.0	128.25	205.0	62.0	447.016129
MI	73.0	98.109589	37.200120	30.0	70.00	97.0	126.00	225.0	73.0	442.479452
MN	84.0	101.571429	41.309086	13.0	64.25	100.5	132.50	186.0	84.0	440.476190
MO	63.0	98.619048	43.412288	6.0	73.00	101.0	127.00	196.0	63.0	429.920635
MS	65.0	99.907692	41.665604	1.0	70.00	103.0	126.00	193.0	65.0	441.153846
MT	68.0	92.676471	42.418118	10.0	60.00	96.5	121.75	185.0	68.0	437.000000
NC	68.0	104.779412	41.059784	26.0	73.75	108.0	127.00	210.0	68.0	433.382353
ND	62.0	106.209677	41.611449	12.0	76.00	104.5	135.75	190.0	62.0	435.838710
NE	61.0	98.721311	38.750540	21.0	73.00	92.0	126.00	194.0	61.0	435.311475

	acc_len								area_code	
	count	mean	std	min	25%	50%	75%	max	count	mean
state										
NH	56.0	101.875000	36.238133	13.0	81.75	104.0	124.25	174.0	56.0	432.232143
NJ	68.0	100.205882	39.445621	1.0	73.50	98.0	129.50	197.0	68.0	440.000000
NM	62.0	104.370968	45.704408	16.0	67.25	104.5	132.00	232.0	62.0	430.048387
NV	66.0	102.803030	41.169152	3.0	77.25	105.0	129.75	183.0	66.0	439.424242
NY	83.0	100.698795	46.897848	9.0	64.50	104.0	134.00	209.0	83.0	432.855422
OH	78.0	102.153846	34.181744	29.0	75.00	99.0	132.75	169.0	78.0	432.512821
OK	61.0	108.262295	42.462494	15.0	84.00	104.0	146.00	194.0	61.0	439.524590
OR	78.0	101.166667	44.424287	6.0	67.50	98.0	135.75	190.0	78.0	438.102564
PA	45.0	104.044444	33.491488	28.0	82.00	101.0	128.00	176.0	45.0	438.155556
RI	65.0	101.907692	37.082983	17.0	75.00	107.0	125.00	190.0	65.0	440.015385
SC	60.0	97.016667	43.621054	1.0	77.75	99.5	112.50	212.0	60.0	440.400000
SD	60.0	105.450000	41.092053	20.0	86.75	104.5	128.00	221.0	60.0	438.466667
TN	53.0	92.716981	41.252215	1.0	68.00	99.0	120.00	196.0	53.0	435.056604
TX	72.0	97.541667	44.562328	1.0	70.00	90.0	125.25	217.0	72.0	432.847222
UT	72.0	100.958333	43.311332	4.0	72.75	96.0	118.75	243.0	72.0	444.180556
VA	77.0	105.935065	41.455219	10.0	84.00	108.0	136.00	210.0	77.0	433.701299
VT	73.0	104.424658	35.091514	38.0	75.00	101.0	128.00	195.0	73.0	439.397260
WA	66.0	102.060606	41.403411	13.0	67.25	103.5	133.50	184.0	66.0	437.030303
WI	78.0	98.884615	36.227987	16.0	72.25	94.5	122.75	184.0	78.0	438.602564
WV	106.0	95.433962	39.983096	11.0	67.00	92.5	124.00	191.0	106.0	444.150943
WY	77.0	105.740260	42.872333	16.0	78.00	104.0	129.00	225.0	77.0	436.896104

51 rows × 360 columns

In [33]:
```
data.groupby(['area_code', 'churn']).describe()
```

Out[33]:

		acc_len								area_c
		count	mean	std	min	25%	50%	75%	max	count
area_code	churn									
408	False	716.0	101.234637	41.020455	1.0	73.00	100.0	130.0	232.0	716.0
	True	122.0	105.647541	36.426746	12.0	79.25	108.5	130.0	178.0	122.0
415	False	1419.0	100.590557	39.213685	1.0	74.00	100.0	126.0	225.0	1419.0
	True	236.0	103.944915	39.475189	1.0	77.00	102.0	126.0	225.0	236.0
510	False	715.0	100.755245	40.095023	3.0	73.00	101.0	128.5	243.0	715.0
	True	125.0	97.336000	42.029165	16.0	70.00	98.0	125.0	224.0	125.0

6 rows × 352 columns

In [34]:
```
data.groupby(['state', 'churn']).describe()
```

Out[34]:

		acc_len								area_cod	
		count	mean	std	min	25%	50%	75%	max	count	me
state	churn										
AK	False	49.0	94.693878	37.393295	1.0	59.00	100.0	126.00	173.0	49.0	44
	True	3.0	138.000000	34.597688	111.0	118.50	126.0	151.50	177.0	3.0	41
AL	False	72.0	98.125000	39.747420	8.0	75.25	97.0	121.25	200.0	72.0	43
	True	8.0	97.125000	58.728035	25.0	58.75	87.5	112.75	197.0	8.0	42
AR	False	44.0	94.840909	39.469118	5.0	72.50	92.5	116.00	185.0	44.0	44
	True	11.0	105.545455	24.414601	54.0	98.50	109.0	116.50	145.0	11.0	42
AZ	False	60.0	102.166667	30.852941	43.0	83.75	97.0	124.75	192.0	60.0	43
	True	4.0	75.000000	47.081490	12.0	52.50	87.5	110.00	113.0	4.0	46
CA	False	25.0	98.160000	34.453689	33.0	75.00	93.0	120.00	185.0	25.0	43
	True	9.0	102.222222	33.476775	37.0	84.00	105.0	112.00	151.0	9.0	44
CO	False	57.0	99.596491	38.625889	22.0	71.00	98.0	125.00	192.0	57.0	43
	True	9.0	121.222222	26.850409	77.0	105.00	121.0	132.00	159.0	9.0	41
CT	False	62.0	99.903226	43.967838	3.0	72.25	99.5	132.75	212.0	62.0	43
	True	12.0	94.583333	36.659262	23.0	73.75	94.5	119.25	160.0	12.0	42
DC	False	49.0	109.448980	39.465735	11.0	81.00	108.0	138.00	217.0	49.0	43
	True	5.0	69.200000	32.820725	24.0	60.00	66.0	82.00	114.0	5.0	41
DE	False	52.0	101.423077	30.854082	30.0	80.75	98.5	119.50	183.0	52.0	43
	True	9.0	104.777778	62.461544	2.0	67.00	113.0	129.00	224.0	9.0	46
FL	False	55.0	109.163636	32.130385	22.0	91.50	109.0	129.50	185.0	55.0	43
	True	8.0	112.375000	37.359021	55.0	92.50	116.0	135.00	166.0	8.0	48
GA	False	46.0	104.195652	43.399754	18.0	72.25	98.0	132.75	202.0	46.0	44
	True	8.0	100.125000	30.385323	44.0	85.75	105.0	123.50	132.0	8.0	43
HI	False	50.0	105.560000	41.186069	3.0	86.25	105.0	133.50	190.0	50.0	42
	True	3.0	104.000000	4.582576	99.0	102.00	105.0	106.50	108.0	3.0	41
IA	False	41.0	97.195122	41.000744	1.0	66.00	98.0	128.00	197.0	41.0	44
	True	3.0	55.000000	22.605309	40.0	42.00	44.0	62.50	81.0	3.0	47
ID	False	64.0	103.718750	38.378428	24.0	82.00	99.0	123.25	224.0	64.0	44
	True	9.0	110.000000	36.530809	77.0	82.00	103.0	119.00	193.0	9.0	42
IL	False	53.0	101.641509	41.002389	16.0	74.00	98.0	134.00	183.0	53.0	43
	True	5.0	88.600000	28.236501	68.0	71.00	78.0	89.00	137.0	5.0	43

		acc_len								area_cod	
		count	mean	std	min	25%	50%	75%	max	count	me
state	churn										
...	...	...	...	...	...	...	...	...	...	...	...
OK	False	52.0	109.538462	44.010008	15.0	82.75	111.0	146.75	194.0	52.0	43
	True	9.0	100.888889	33.220643	44.0	86.00	101.0	105.00	154.0	9.0	46
OR	False	67.0	100.567164	45.553573	6.0	66.50	98.0	134.50	190.0	67.0	44
	True	11.0	104.818182	38.511864	33.0	87.50	99.0	137.00	149.0	11.0	42
PA	False	37.0	105.513514	35.405290	28.0	82.00	101.0	134.00	176.0	37.0	43
	True	8.0	97.250000	23.230214	61.0	78.00	101.5	119.00	122.0	8.0	43
RI	False	59.0	99.796610	38.019859	17.0	74.50	104.0	123.00	190.0	59.0	43
	True	6.0	122.666667	16.256281	103.0	109.75	123.0	134.75	143.0	6.0	45
SC	False	46.0	91.978261	39.407551	1.0	74.25	99.0	111.00	181.0	46.0	43
	True	14.0	113.571429	53.620768	19.0	87.25	110.0	127.75	212.0	14.0	44
SD	False	52.0	107.500000	41.534064	20.0	86.75	109.0	128.00	221.0	52.0	43
	True	8.0	92.125000	37.809438	27.0	81.25	95.5	110.00	144.0	8.0	47
TN	False	48.0	93.270833	41.573181	1.0	68.00	98.5	121.50	196.0	48.0	43
	True	5.0	87.400000	42.140242	17.0	80.00	108.0	112.00	120.0	5.0	45
TX	False	54.0	95.222222	40.857168	19.0	67.00	90.0	123.00	217.0	54.0	42
	True	18.0	104.500000	54.963357	1.0	73.50	94.0	138.25	208.0	18.0	44
UT	False	62.0	100.500000	43.964125	4.0	75.00	95.5	117.75	243.0	62.0	44
	True	10.0	103.800000	41.082032	49.0	68.75	104.0	127.25	170.0	10.0	43
VA	False	72.0	107.500000	42.062663	10.0	85.50	113.0	136.50	210.0	72.0	43
	True	5.0	83.400000	23.849528	50.0	68.00	92.0	99.00	108.0	5.0	43
VT	False	65.0	104.246154	35.810277	38.0	75.00	101.0	128.00	195.0	65.0	44
	True	8.0	105.875000	30.638153	63.0	89.50	106.0	119.75	159.0	8.0	42
WA	False	52.0	100.442308	40.271828	13.0	65.25	106.5	132.50	184.0	52.0	43
	True	14.0	108.071429	46.468292	45.0	78.00	96.5	155.50	177.0	14.0	43
WI	False	71.0	97.591549	34.114755	20.0	72.50	94.0	120.00	184.0	71.0	43
	True	7.0	112.000000	55.181519	16.0	79.00	147.0	151.00	161.0	7.0	45
WV	False	96.0	95.145833	40.062840	11.0	67.00	92.5	124.00	191.0	96.0	44
	True	10.0	98.200000	41.225127	44.0	65.25	100.5	121.75	161.0	10.0	47
WY	False	68.0	104.985294	44.690809	16.0	76.25	104.0	129.50	225.0	68.0	43

		acc_len								area_cod	
		count	mean	std	min	25%	50%	75%	max	count	me
state	churn										
	True	9.0	111.444444	26.320672	76.0	94.00	106.0	127.00	159.0	9.0	45

102 rows × 360 columns

In [35]:
```
pd.crosstab(data['prem_customer'], data['churn'])
```

Out[35]:

churn	False	True
prem_customer		
0	2794	447
1	56	36

In [36]:
```
data.groupby(["churn"]).describe().transpose()
```

Out[36]:

	churn	False	True
acc_len	**count**	2850.000000	483.000000
	mean	100.793684	102.664596
	std	39.882350	39.467820
	min	1.000000	1.000000
	25%	73.000000	76.000000
	50%	100.000000	103.000000
	75%	127.000000	127.000000
	max	243.000000	225.000000
area_code	**count**	2850.000000	483.000000
	mean	437.074737	437.817805
	std	42.306156	42.792270
	min	408.000000	408.000000
	25%	408.000000	408.000000
	50%	415.000000	415.000000
	75%	510.000000	510.000000
	max	510.000000	510.000000
area_code_408	**count**	2850.000000	483.000000
	mean	0.251228	0.252588
	std	0.433796	0.434947
	min	0.000000	0.000000
	25%	0.000000	0.000000
	50%	0.000000	0.000000
	75%	1.000000	1.000000
	max	1.000000	1.000000
area_code_415	**count**	2850.000000	483.000000
	mean	0.497895	0.488613
	std	0.500083	0.500389
	min	0.000000	0.000000
	25%	0.000000	0.000000
	50%	0.000000	0.000000
...	**...**	...	...
total_night_calls	**std**	19.506246	19.950659

	churn	False	True
	min	33.000000	49.000000
	25%	87.000000	85.000000
	50%	100.000000	100.000000
	75%	113.000000	115.000000
	max	175.000000	158.000000
total_night_charge	**count**	2850.000000	483.000000
	mean	9.006074	9.235528
	std	2.299768	2.121081
	min	1.040000	2.130000
	25%	7.470000	7.705000
	50%	9.010000	9.220000
	75%	10.570000	10.795000
	max	17.770000	15.970000
total_night_min	**count**	2850.000000	483.000000
	mean	200.133193	205.231677
	std	51.105032	47.132825
	min	23.200000	47.400000
	25%	165.900000	171.250000
	50%	200.250000	204.800000
	75%	234.900000	239.850000
	max	395.000000	354.900000
voice_mail_plan	**count**	2850.000000	483.000000
	mean	0.295439	0.165631
	std	0.456320	0.372135
	min	0.000000	0.000000
	25%	0.000000	0.000000
	50%	0.000000	0.000000
	75%	1.000000	0.000000
	max	1.000000	1.000000

360 rows × 2 columns

In [37]:
```
data.corr()['churn'].sort_values(ascending=False)
```

Out[37]:
```
churn                                1.000000e+00
int_plan                             2.598518e-01
total_charge                         2.315487e-01
customer_service_calls               2.087500e-01
total_day_minutes                    2.051508e-01
cus_dev_day_charge                   2.051507e-01
total_day_charge                     2.051507e-01
total_min                            1.986074e-01
day_avg_charge                       1.540306e-01
prem_customer                        1.179279e-01
total_eve_min                        9.279579e-02
total_eve_charge                     9.278604e-02
cus_dev_eve_charge                   9.278604e-02
total_int_charge                     6.825863e-02
total_int_min                        6.823878e-02
deviation_day_charge_state           5.223081e-02
day_avg_charge[data_state]           5.223081e-02
eve_avg_charge                       5.001727e-02
cus_dev_night_charge                 3.549556e-02
total_night_charge                   3.549556e-02
total_night_min                      3.549285e-02
total_calls                          2.377784e-02
night_avg_charge                     2.343369e-02
deviation_eve_charge_state           2.130952e-02
eve_avg_charge[data_state]           2.130952e-02
total_day_calls                      1.845931e-02
acc_len                              1.654074e-02
total_eve_calls                      9.233132e-03
eve_avg_charge[data_area_code]       6.678490e-03
area_code_510                        6.422866e-03
area_code                            6.174233e-03
total_night_calls                    6.141203e-03
area_code_408                        1.103450e-03
eve_avg_charge[data]                 6.629039e-16
day_avg_charge[data]                 9.081053e-17
night_avg_charge[data]              -1.822751e-16
day_avg_charge[data_area_code]      -5.247677e-03
deviation_day_charge_area_code      -5.247677e-03
deviation_eve_charge_area_code      -5.279774e-03
night_avg_charge[data_area_code]    -5.279774e-03
area_code_415                       -6.534885e-03
deviation_night_charge_state        -6.658842e-03
night_avg_charge[data_state]        -6.658842e-03
total_int_calls                     -5.284434e-02
num_vmail_msg                       -8.972797e-02
voice_mail_plan                     -1.021481e-01
Name: churn, dtype: float64
```

Stage 4. Modelling

In [38]:
```
data = data.fillna(0)
final_data = data.copy()
final_data = final_data.fillna(0)
Z = final_data.copy()
Z.head()
```

Out[38]:

	id	state	acc_len	area_code	int_plan	voice_mail_plan	num_vmail_msg	total_da
0	CUST-1	KS	128	415	0	1	25	265.1
1	CUST-369	KS	132	415	0	0	0	83.4
2	CUST-380	KS	127	415	0	0	0	221.0
3	CUST-386	KS	137	415	0	0	0	230.2
4	CUST-620	KS	110	415	1	0	0	293.3

5 rows × 48 columns

In [39]:
```
#normalize the data

from sklearn.preprocessing import StandardScaler

X = StandardScaler().fit_transform(Z[num_data])
X = pd.DataFrame(X, index=Z[num_data].index, columns=Z[num_data].columns)
# # add non-feature target column to dataframe
X['churn'] = Z['churn']
X.head(10)
```

Out[39]:

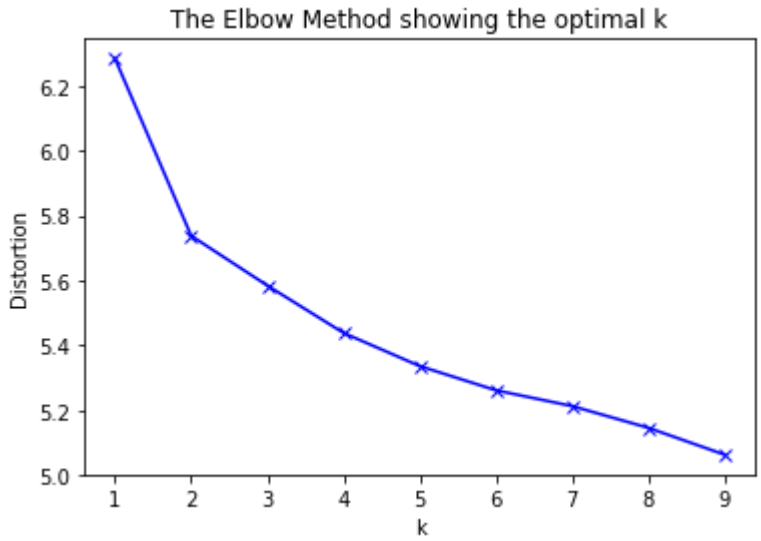
	acc_len	prem_customer	int_plan	voice_mail_plan	num_vmail_msg	total_day_mi
0	0.676489	-0.168482	-0.327580	1.617086	1.234883	1.566767
1	0.776951	-0.168482	-0.327580	-0.618396	-0.591760	-1.769675
2	0.651374	-0.168482	-0.327580	-0.618396	-0.591760	0.756987
3	0.902529	-0.168482	-0.327580	-0.618396	-0.591760	0.925920
4	0.224411	-0.168482	3.052685	-0.618396	-0.591760	2.084586
5	1.555530	-0.168482	-0.327580	-0.618396	-0.591760	0.211624
6	-0.328128	-0.168482	-0.327580	-0.618396	-0.591760	0.184081
7	-0.051859	-0.168482	-0.327580	-0.618396	-0.591760	0.209788
8	0.224411	5.935340	3.052685	1.617086	1.381014	1.618182
9	-0.780207	-0.168482	-0.327580	-0.618396	-0.591760	0.790039

10 rows × 45 columns

4.1 Unsupervised approach >> Mental Model

Hypothesis: As there are estimating two classes churn: True - False This helps us to establish a mental model to check if the data is actually giving us that pattern.

In [40]:
```
#this clearly shows that we are looking for two classes in our data set. Menta
l Model
from sklearn.cluster import KMeans
from scipy.spatial.distance import cdist

# k means determine k
distortions = []
K = range(1,10)
for k in K:
    kmeanModel = KMeans(n_clusters=k).fit(X[num_data])
    kmeanModel.fit(X[num_data])
    distortions.append(sum(np.min(cdist(X[num_data], kmeanModel.cluster_center
s_, 'euclidean'), axis=1)) / X[num_data].shape[0])

# Plot the elbow
plt.plot(K, distortions, 'bx-')
plt.xlabel('k')
plt.ylabel('Distortion')
plt.title('The Elbow Method showing the optimal k')
plt.savefig("elbow.png")
plt.show()
```



In [41]:
```
from sklearn.metrics import classification_report
# create kmeans object
kmeans = KMeans(n_clusters=2, random_state= 123)
# fit kmeans object to data
kmeans.fit(X[num_data])

# save new clusters for chart
predicted1 = kmeans.fit_predict(X[num_data])
predicted1 = np.choose(predicted1, [0,1]).astype(int)
predicted1 = pd.DataFrame(predicted1, columns=['target'])

val_df = pd.concat([X[target], predicted1],axis=1)

print(classification_report(val_df['churn'], val_df['target']))
pd.crosstab(val_df['churn'], val_df['target'])
```

```
             precision    recall  f1-score   support

      False       0.86      0.50      0.63      2850
       True       0.15      0.51      0.23       483

avg / total       0.75      0.50      0.57      3333
```

Out[41]:

target	0	1
churn		
False	1419	1431
True	236	247

4.2 Supervised Approach

In [42]:
```
## first make a model function for modeling with confusion matrix
#During a classification task, Confusion Matric brings about better insights a
s to what group should we target based on the problem.

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score, f1_score, precision_score, recall_
score, classification_report, confusion_matrix, roc_curve, precision_recall_cu
rve, average_precision_score, precision_recall_fscore_support
from sklearn.metrics import roc_auc_score, auc

def report_model(model,features_train,features_test,labels_train,labels_test):
    clf= model
    clf.fit(features_train,labels_train.values.ravel())
    pred=clf.predict(features_test)
    cnf_matrix=confusion_matrix(labels_test,pred)
    print("the recall for this model is :",cnf_matrix[1,1]/(cnf_matrix[1,1]+cn
f_matrix[1,0]))
    fig= plt.figure(figsize=(6,3))# to plot the graph
    print("TP",cnf_matrix[1,1,]) # no of fraud transaction which are predicted
 fraud
    print("TN",cnf_matrix[0,0]) # no. of normal transaction which are predited
 normal
    print("FP",cnf_matrix[0,1]) # no of normal transaction which are predicted
 fraud
    print("FN",cnf_matrix[1,0]) # no of fraud Transaction which are predicted
 normal
    sns.heatmap(cnf_matrix,cmap="coolwarm_r",annot=True,linewidths=0.5)
    plt.title("Confusion_matrix")
    plt.xlabel("Predicted_class")
    plt.ylabel("Real class")
    plt.show()
    print("\n----------Classification Report------------------------------
--")
    print(classification_report(labels_test,pred))
```

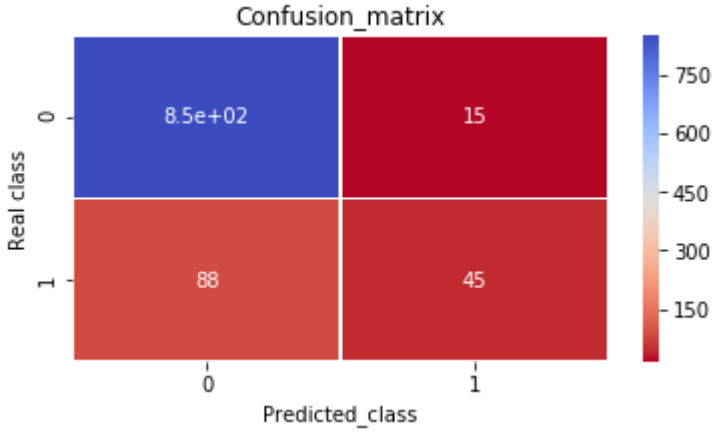
In [43]:
```
#Data Split 70:30
from sklearn.cross_validation import train_test_split

train, test = sklearn.cross_validation.train_test_split(X, train_size = 0.7, r
andom_state=123)
```

In [44]:
```
train_x = train[num_data]
test_x = test[num_data]

train_y =train[target]
test_y =test[target]
```

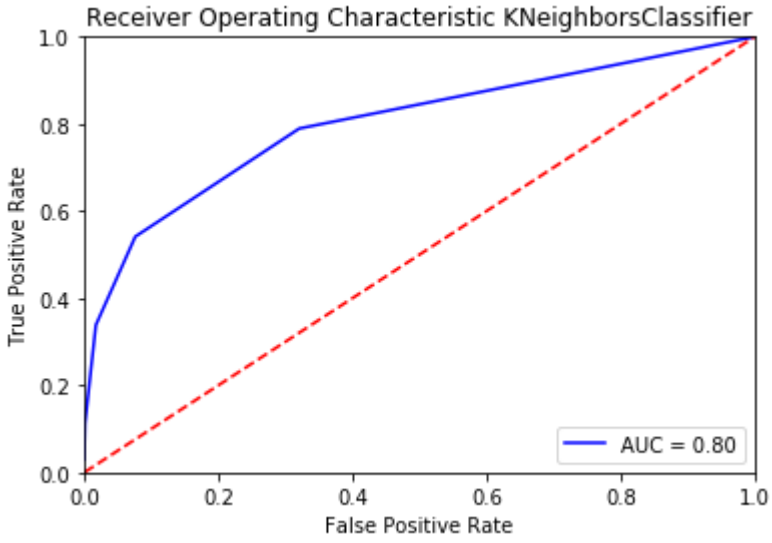
In [45]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LogisticRegressionCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn.ensemble import GradientBoostingClassifier

classifiers = [
    KNeighborsClassifier(n_jobs = 20),
    RandomForestClassifier(n_jobs = 20),
    GaussianNB(),
    LogisticRegression(n_jobs= 5, random_state=123),
    LogisticRegressionCV(random_state=111, n_jobs=50),
    DecisionTreeClassifier(criterion='gini'),
    DecisionTreeClassifier(criterion='entropy'),
    RandomForestClassifier(max_depth=100, random_state=123, n_jobs=20, n_estim
ators=120),
    svm.SVC(random_state=123, kernel='rbf',probability=True),
    GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=
1, random_state=123)
]
```

In [46]:
```
# iterate over classifiers
for item in classifiers:
    classifier_name = ((str(item)[:(str(item).find("("))]))
    print (classifier_name)

    # Create classifier, train it and test it.
    clf = item
    print(item)

    clf.fit(train_x, train_y)
    pred = clf.predict(test_x)
    score = clf.score(test_x, test_y)
    report_model(clf, train_x, test_x, train_y,test_y)

    if classifier_name in ["RandomForestClassifier", 'DecisionTreeClassifier'
]:
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        features = train_x.columns

        importances = clf.feature_importances_
        indices = np.argsort(importances)
        plt.title('Feature Importances')
        plt.barh(range(len(indices)), importances[indices], color='b', align=
'center')
        plt.yticks(range(len(indices)), [features[i] for i in indices], fontsi
ze = 6)
        plt.xlabel('Relative Importance')
        plt.figure(figsize=(20,40))
        plt.savefig(classifier_name+ "_feature_imp.png")
        plt.show()


    # calculate the fpr and tpr for all thresholds of the classification
    probs = clf.predict_proba(test_x)
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(test_y, preds)
    roc_auc = auc(fpr, tpr)

    plt.title('Receiver Operating Characteristic '+classifier_name)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig(classifier_name+ "_roc_auc.png")
    plt.show()

    print ("Score: ", round(score,3),"\nAccuracy score: ", round(accuracy_scor
e(test_y, pred), 3),"\nF1 score: ", round(f1_score(test_y, pred), 3), "\n- - -
 - - ", "\n")
```

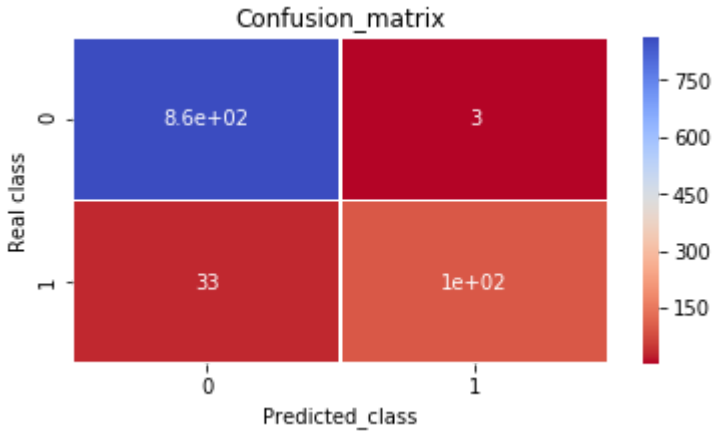
```
KNeighborsClassifier
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=20, n_neighbors=5, p=2,
           weights='uniform')
the recall for this model is : 0.3383458646616541
TP 45
TN 852
FP 15
FN 88
```



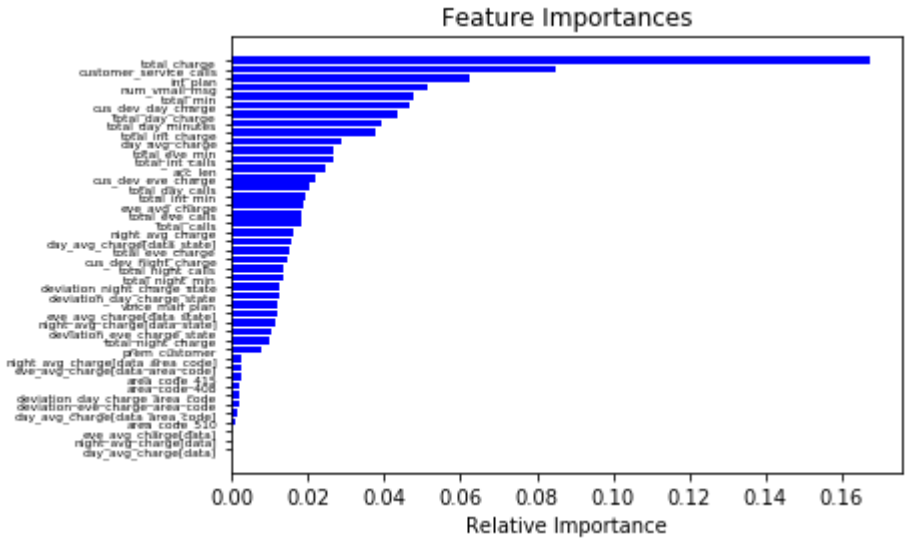

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.91      0.98      0.94       867
       True       0.75      0.34      0.47       133

avg / total       0.89      0.90      0.88      1000
```



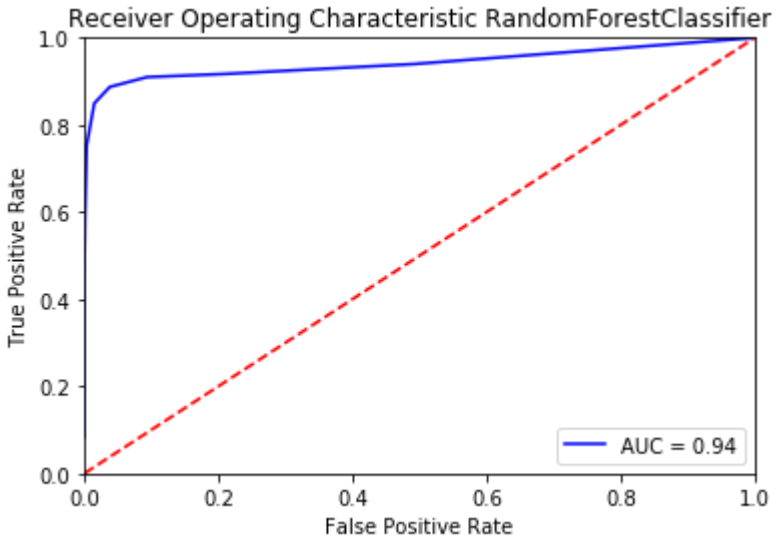
```
Score:  0.897
Accuracy score:  0.897
F1 score:  0.466
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=20,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
the recall for this model is : 0.7518796992481203
TP 100
TN 864
FP 3
FN 33
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.96      1.00      0.98       867
       True       0.97      0.75      0.85       133

avg / total       0.96      0.96      0.96      1000
```

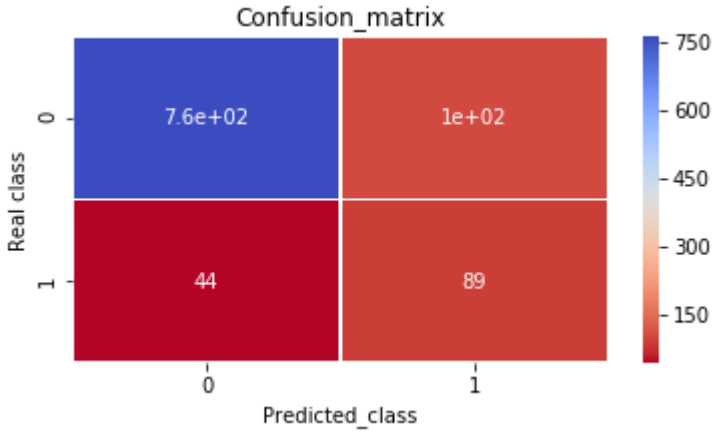


<Figure size 1440x2880 with 0 Axes>

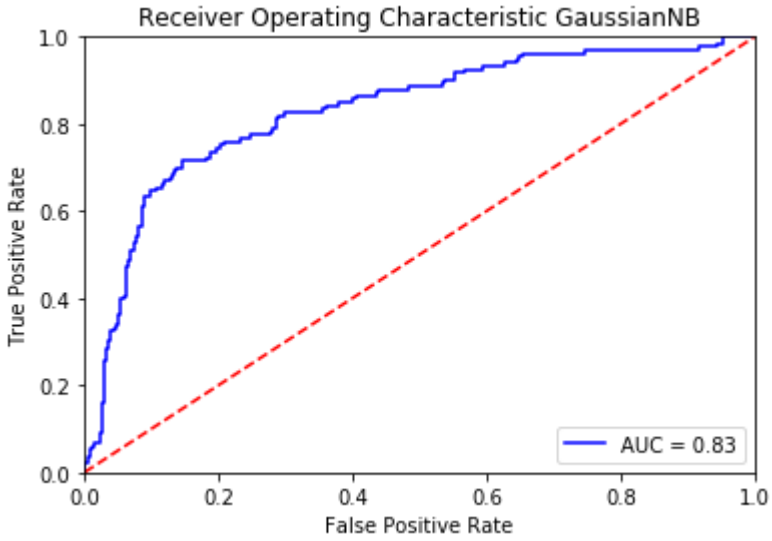


Score: 0.971
Accuracy score: 0.971
F1 score: 0.879
- - - - -

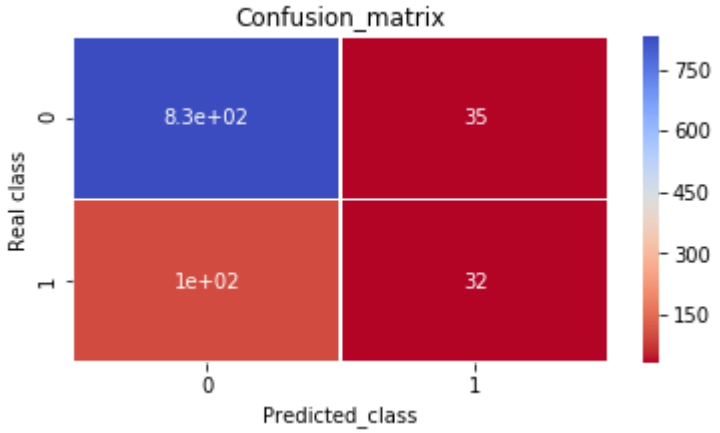
GaussianNB
GaussianNB(priors=None)
the recall for this model is : 0.6691729323308271
TP 89
TN 763
FP 104
FN 44



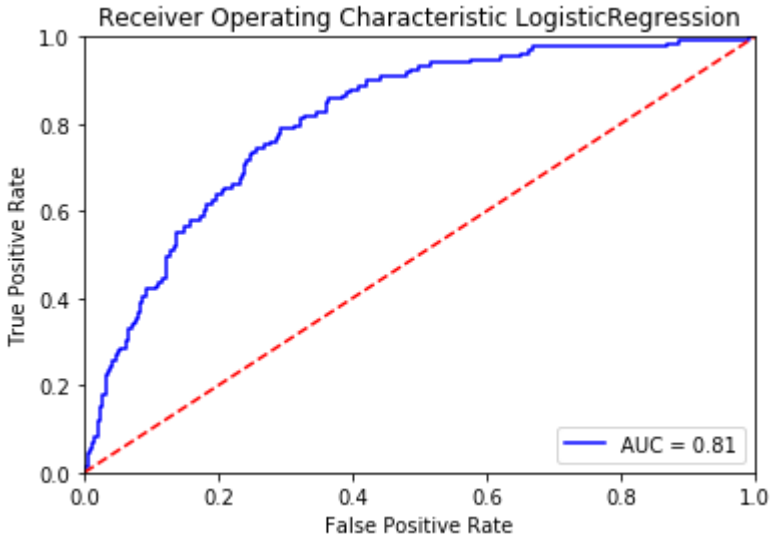

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.95      0.88      0.91       867
       True       0.46      0.67      0.55       133

avg / total       0.88      0.85      0.86      1000
```



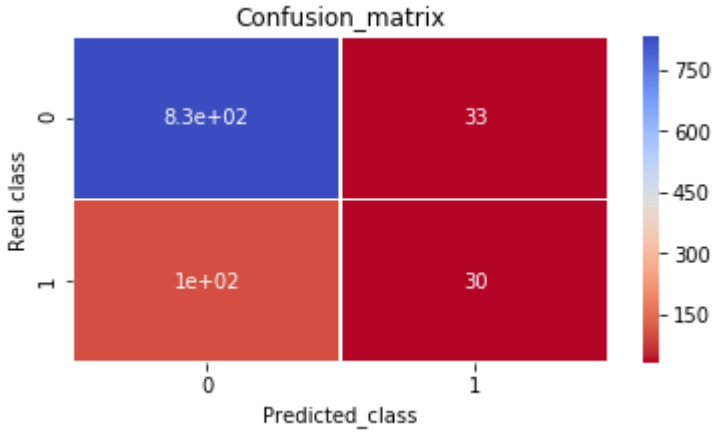

```
Score:  0.852
Accuracy score:  0.852
F1 score:  0.546
- - - - -

LogisticRegression
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=5,
          penalty='l2', random_state=123, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
the recall for this model is : 0.24060150375939848
TP 32
TN 832
FP 35
FN 101
```



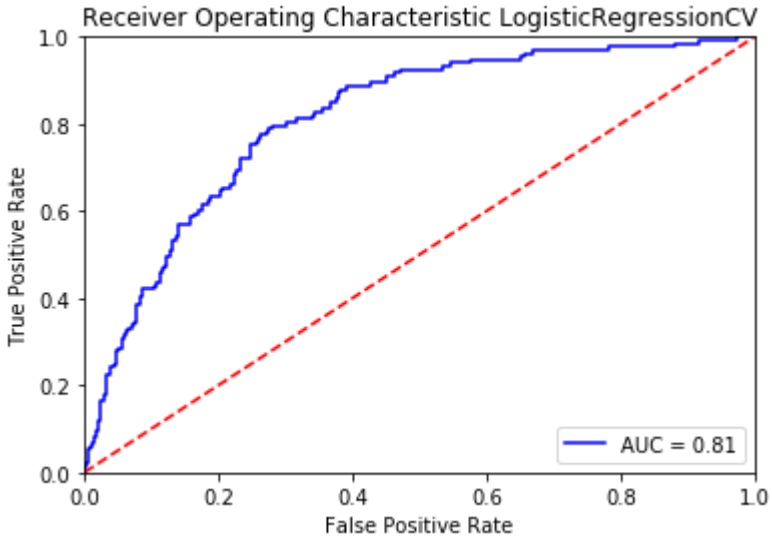

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.89      0.96      0.92       867
       True       0.48      0.24      0.32       133

avg / total       0.84      0.86      0.84      1000
```



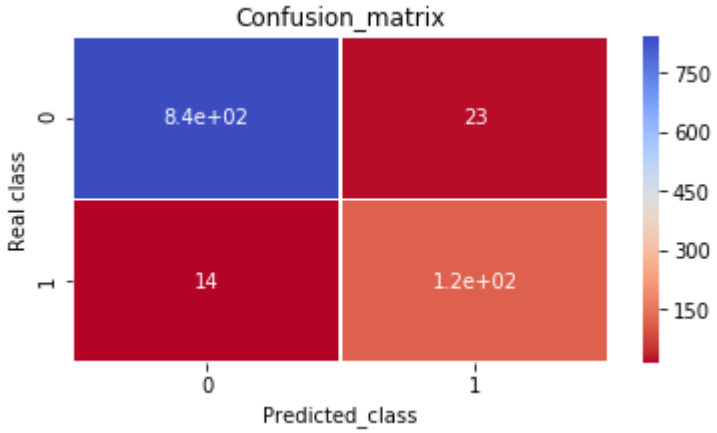

```
Score:  0.864
Accuracy score:  0.864
F1 score:  0.32
- - - - -

LogisticRegressionCV
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=50, penalty='l2', random_state=111,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
the recall for this model is : 0.22556390977443608
TP 30
TN 834
FP 33
FN 103
```



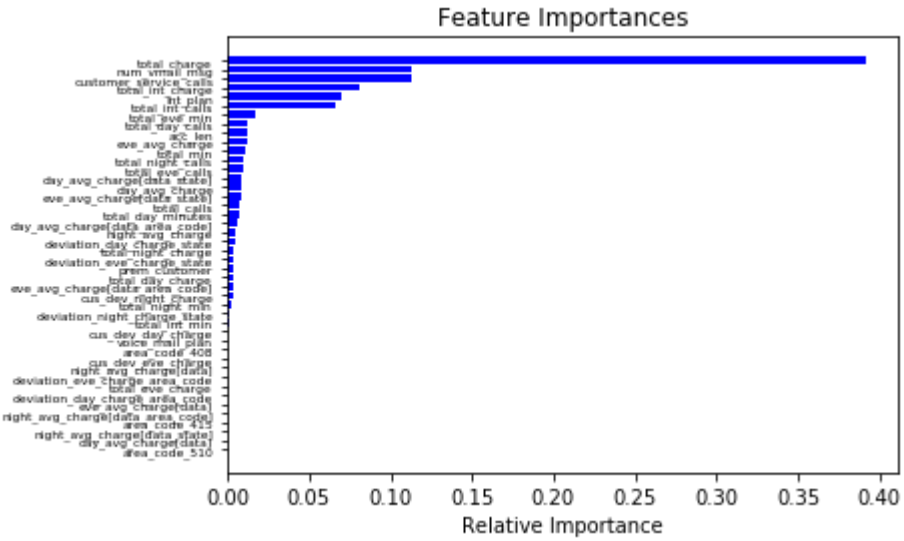

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.89      0.96      0.92       867
       True       0.48      0.23      0.31       133

avg / total       0.84      0.86      0.84      1000
```



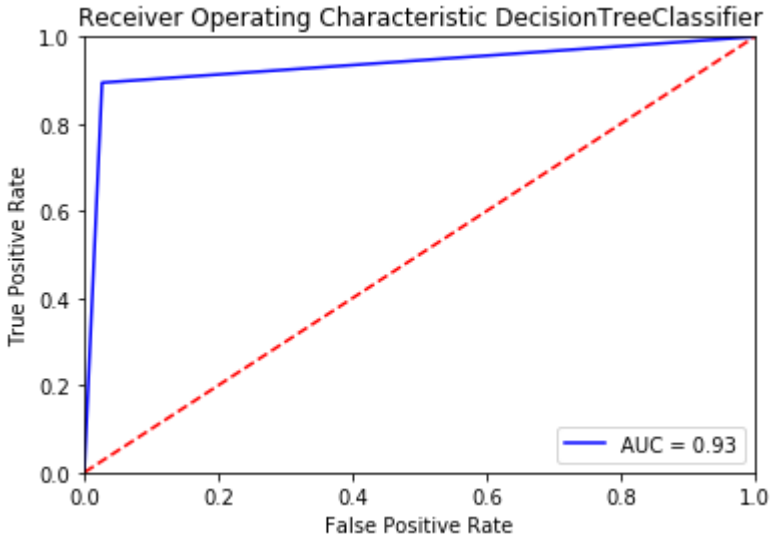

```
Score:  0.864
Accuracy score:  0.864
F1 score:  0.306
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.8947368421052632
TP 119
TN 844
FP 23
FN 14
```



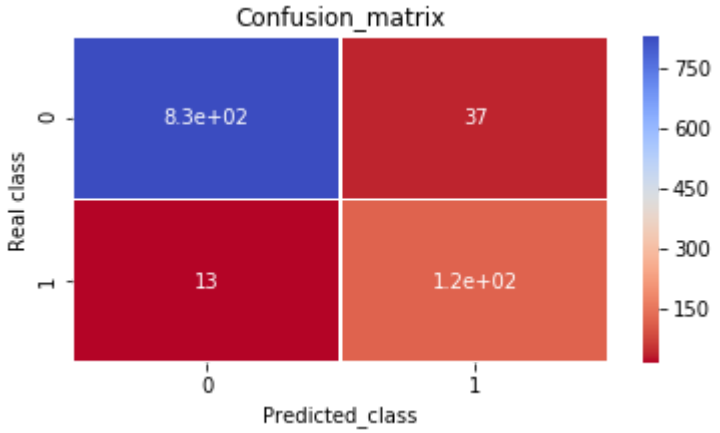

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.98      0.97      0.98       867
       True       0.84      0.89      0.87       133

avg / total       0.96      0.96      0.96      1000
```



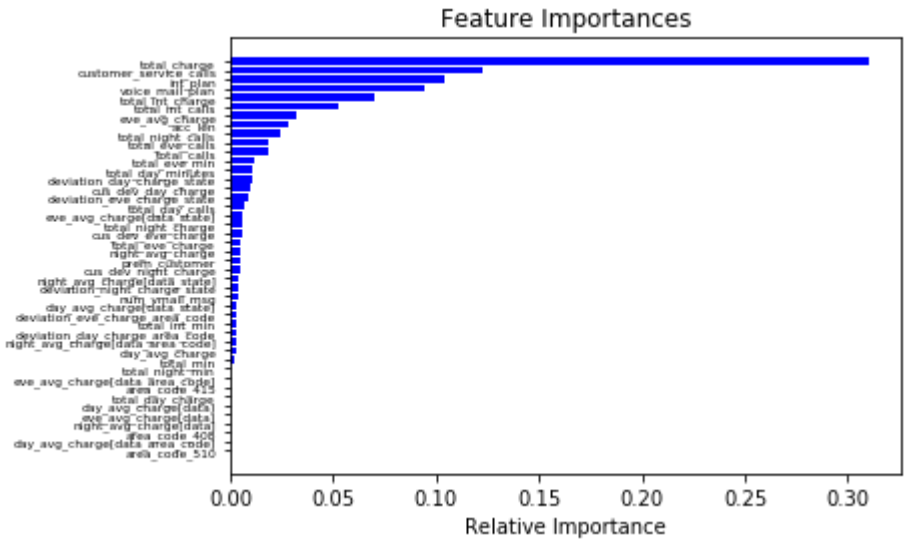

```
<Figure size 1440x2880 with 0 Axes>
```



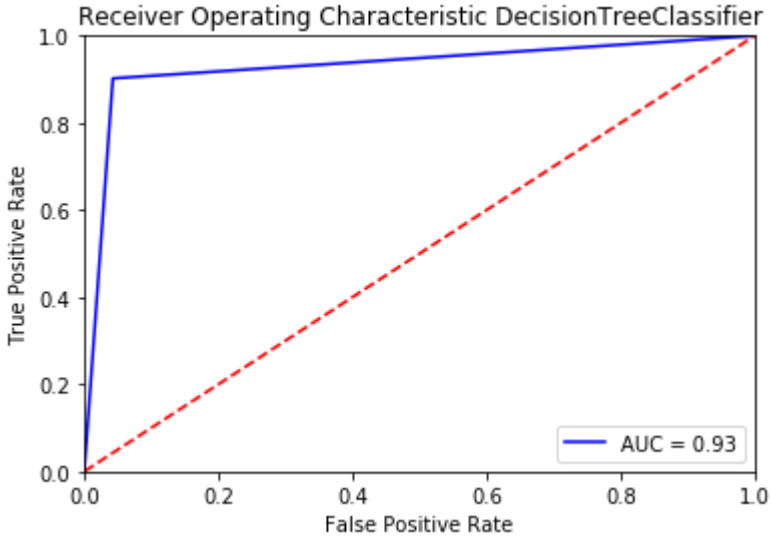

```
Score:  0.96
Accuracy score:  0.96
F1 score:  0.857
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.9022556390977443
TP 120
TN 830
FP 37
FN 13
```



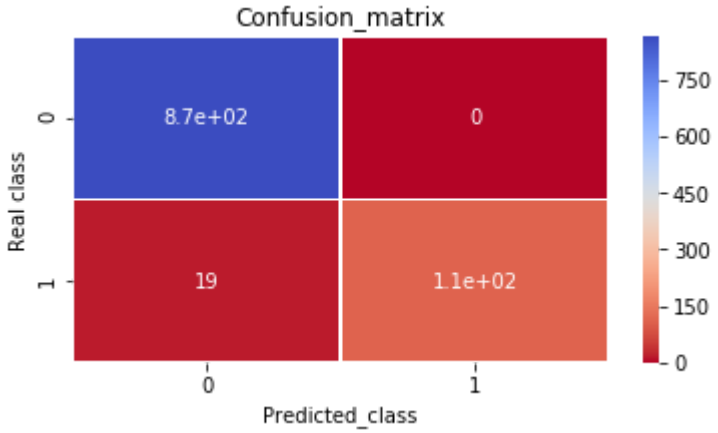
```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.98      0.96      0.97       867
       True       0.76      0.90      0.83       133

avg / total       0.96      0.95      0.95      1000
```



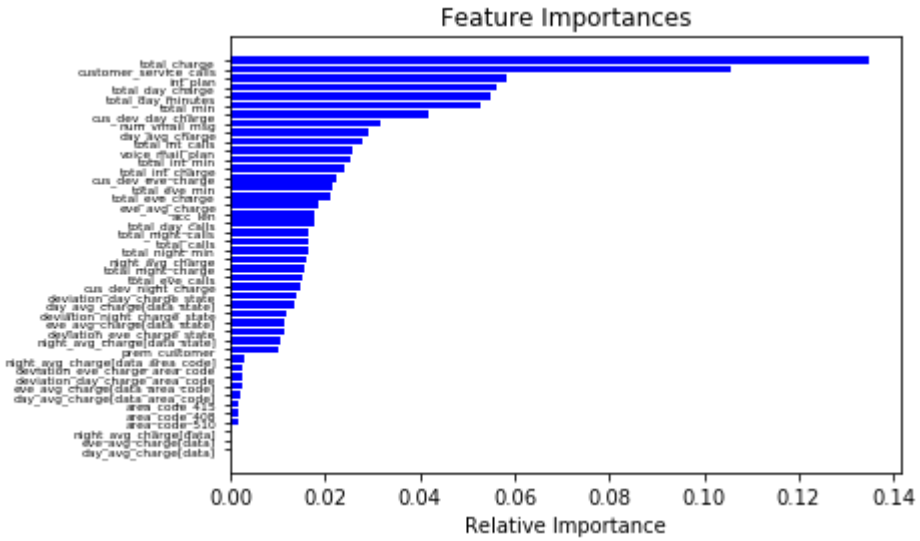

```
<Figure size 1440x2880 with 0 Axes>
```



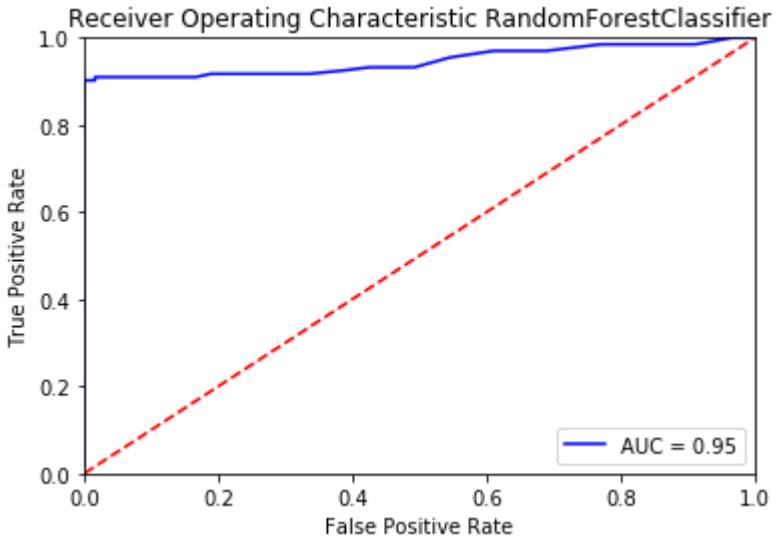
```
Score:  0.957
Accuracy score:  0.957
F1 score:  0.848
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=120, n_jobs=20,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
the recall for this model is : 0.8571428571428571
TP 114
TN 867
FP 0
FN 19
```



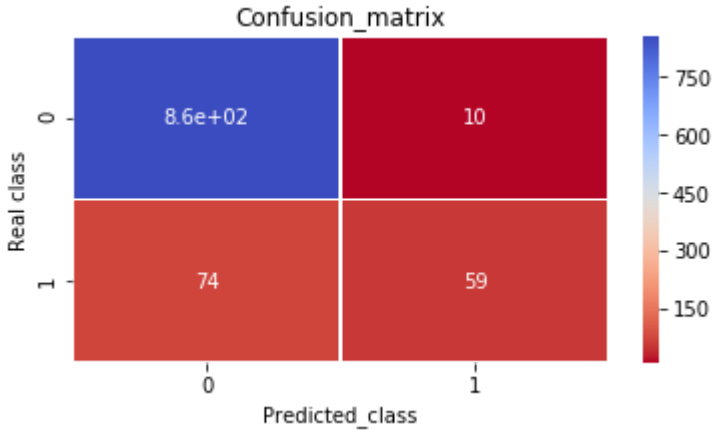

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.98      1.00      0.99       867
       True       1.00      0.86      0.92       133

avg / total       0.98      0.98      0.98      1000
```



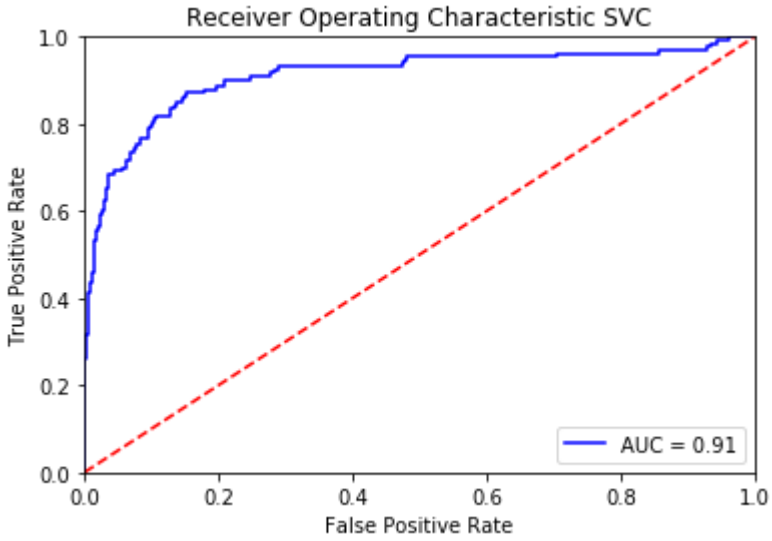
```
<Figure size 1440x2880 with 0 Axes>
```



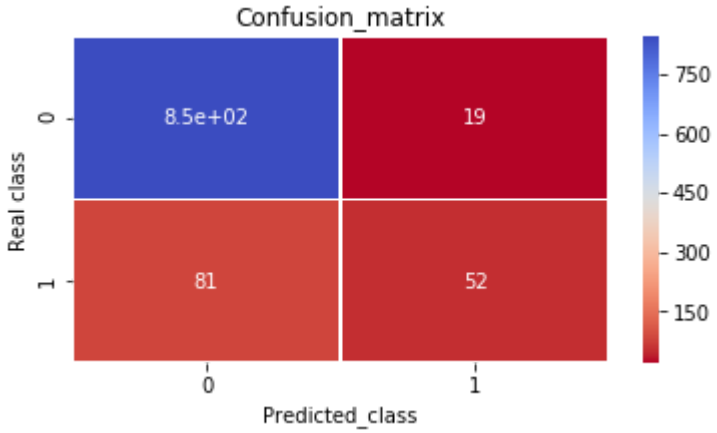

```
Score:  0.981
Accuracy score:  0.981
F1 score:  0.923
- - - - -

SVC
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=123, shrinking=True,
  tol=0.001, verbose=False)
the recall for this model is : 0.44360902255639095
TP 59
TN 857
FP 10
FN 74
```



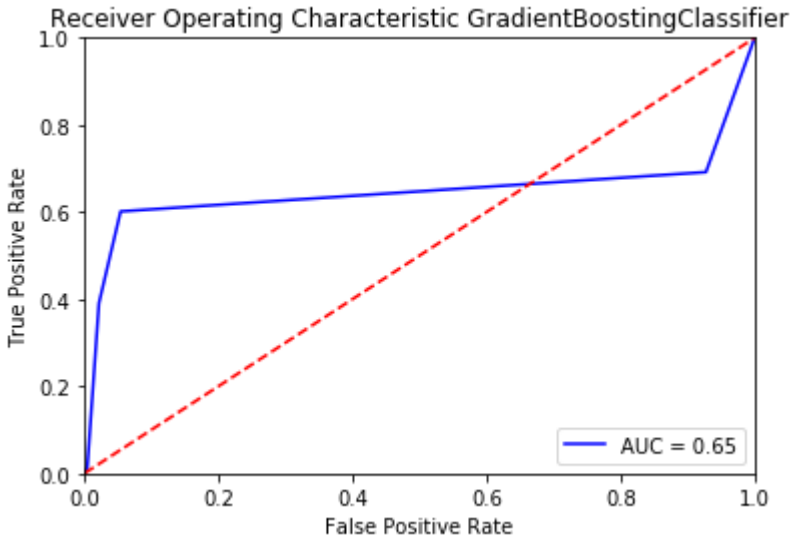

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.92      0.99      0.95       867
       True       0.86      0.44      0.58       133

avg / total       0.91      0.92      0.90      1000
```




```
Score:  0.916
Accuracy score:  0.916
F1 score:  0.584
- - - - -

GradientBoostingClassifier
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1.0, loss='deviance', max_depth=1,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=123, subsample=1.0, verbose=0,
              warm_start=False)
the recall for this model is : 0.39097744360902253
TP 52
TN 848
FP 19
FN 81
```



```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.91      0.98      0.94       867
       True       0.73      0.39      0.51       133

avg / total       0.89      0.90      0.89      1000
```




```
Score:  0.9
Accuracy score:  0.9
F1 score:  0.51
- - - - -
```

In [47]:
```
#Hyper Parameter Tuning with Grid Search and classifier as Random Forest
from sklearn.model_selection import GridSearchCV
model = RandomForestClassifier(random_state=42)

param_grid = [{'n_estimators': [3, 10, 30, 100, 500,1000], 'max_features': ['a
uto', 'sqrt', 'log2', 2, 4, 6, 8],'max_leaf_nodes': [2,4,6,8,16]},
{'bootstrap': [False], 'n_estimators': [3, 10, 20],
 'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']},]

clf = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
clf.fit(train_x, train_y)
# view accuracy scores for all the models
print("Grid scores for all the models based on CV:\n")
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.5f (+/-%0.05f) for %r" % (mean, std * 2, params))
# check out best model performance
print("\nBest parameters set found on development set:", clf.best_params_)
print("Best model validation accuracy:", clf.best_score_)
```

```
Grid scores for all the models based on CV:

0.84869 (+/-0.00461) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.84998 (+/-0.00032) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.84998 (+/-0.00032) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.84998 (+/-0.00032) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.84998 (+/-0.00032) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.84998 (+/-0.00032) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.86069 (+/-0.01686) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.85898 (+/-0.01318) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.87184 (+/-0.01173) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.87784 (+/-0.01712) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.87913 (+/-0.01563) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.88041 (+/-0.01458) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.87098 (+/-0.03695) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.87012 (+/-0.01384) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.88084 (+/-0.01413) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88727 (+/-0.01767) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88856 (+/-0.02332) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.88984 (+/-0.02292) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.88341 (+/-0.04115) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.88598 (+/-0.02055) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.88984 (+/-0.01656) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.89713 (+/-0.01689) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.89970 (+/-0.02491) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.90056 (+/-0.02178) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.89241 (+/-0.04381) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.90184 (+/-0.03460) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.90056 (+/-0.02211) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.90613 (+/-0.02292) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 100}
0.90484 (+/-0.02247) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.90527 (+/-0.01980) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.84869 (+/-0.00461) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.84998 (+/-0.00032) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.84998 (+/-0.00032) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.84998 (+/-0.00032) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.84998 (+/-0.00032) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.84998 (+/-0.00032) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.86069 (+/-0.01686) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.85898 (+/-0.01318) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.87184 (+/-0.01173) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.87784 (+/-0.01712) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.87913 (+/-0.01563) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.88041 (+/-0.01458) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.87098 (+/-0.03695) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.87012 (+/-0.01384) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.88084 (+/-0.01413) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88727 (+/-0.01767) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88856 (+/-0.02332) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.88984 (+/-0.02292) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.88341 (+/-0.04115) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.88598 (+/-0.02055) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.88984 (+/-0.01656) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.89713 (+/-0.01689) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.89970 (+/-0.02491) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.90056 (+/-0.02178) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.89241 (+/-0.04381) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.90184 (+/-0.03460) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 10}
```

```
0.90056 (+/-0.02211) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.90613 (+/-0.02292) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.90484 (+/-0.02247) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.90527 (+/-0.01980) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.84869 (+/-0.00461) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.84998 (+/-0.00032) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.84998 (+/-0.00032) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.84998 (+/-0.00032) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.84998 (+/-0.00032) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.84998 (+/-0.00032) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.87998 (+/-0.01577) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.85769 (+/-0.01006) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.87098 (+/-0.00796) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.86970 (+/-0.01204) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.87312 (+/-0.01402) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.87184 (+/-0.01348) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.90999 (+/-0.03863) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.88856 (+/-0.02095) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.88470 (+/-0.01849) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88127 (+/-0.01539) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88298 (+/-0.01655) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.88298 (+/-0.01489) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.91427 (+/-0.03918) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.89456 (+/-0.02219) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.89070 (+/-0.02384) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.89113 (+/-0.01843) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.89070 (+/-0.02185) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.88898 (+/-0.01843) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.91513 (+/-0.03092) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 3}
0.90956 (+/-0.01605) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.90227 (+/-0.02541) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.90484 (+/-0.01680) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.90313 (+/-0.01980) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.90227 (+/-0.01926) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.85169 (+/-0.00507) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.85041 (+/-0.00186) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.85855 (+/-0.01645) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.85255 (+/-0.00734) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.85555 (+/-0.00440) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.85255 (+/-0.00345) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.86284 (+/-0.01965) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.85298 (+/-0.00956) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.86198 (+/-0.01143) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.85984 (+/-0.00720) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.85727 (+/-0.00649) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
```

```
0.85255 (+/-0.00155) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.87055 (+/-0.02642) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.86584 (+/-0.01327) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.87184 (+/-0.01924) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.87098 (+/-0.01540) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.87098 (+/-0.00924) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.87098 (+/-0.00924) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.84955 (+/-0.00529) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.84998 (+/-0.00032) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.84998 (+/-0.00032) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.84998 (+/-0.00032) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.84998 (+/-0.00032) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.86541 (+/-0.01109) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.85255 (+/-0.00626) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.86712 (+/-0.01249) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.86627 (+/-0.00332) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.86970 (+/-0.00962) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.86884 (+/-0.00748) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.87827 (+/-0.02063) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.86069 (+/-0.01066) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.87698 (+/-0.00755) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.87441 (+/-0.01459) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.87827 (+/-0.01774) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.87698 (+/-0.01693) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.89113 (+/-0.01666) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.88170 (+/-0.01196) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.88170 (+/-0.01484) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.88384 (+/-0.01008) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 100}
0.88341 (+/-0.01303) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.88341 (+/-0.01385) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.89413 (+/-0.01605) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.89584 (+/-0.01567) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.89413 (+/-0.01982) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.89456 (+/-0.01604) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.89627 (+/-0.02125) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.89627 (+/-0.01828) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.84869 (+/-0.00461) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.84998 (+/-0.00032) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.84998 (+/-0.00032) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.84998 (+/-0.00032) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.84998 (+/-0.00032) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.86069 (+/-0.01686) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.85898 (+/-0.01318) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.87184 (+/-0.01173) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.87784 (+/-0.01712) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.87913 (+/-0.01563) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.88041 (+/-0.01458) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.87098 (+/-0.03695) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.87012 (+/-0.01384) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.88084 (+/-0.01413) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.88727 (+/-0.01767) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.88856 (+/-0.02332) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.88984 (+/-0.02292) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.88341 (+/-0.04115) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.88598 (+/-0.02055) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
```

```
0.88984 (+/-0.01656) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.89713 (+/-0.01689) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.89970 (+/-0.02491) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.90056 (+/-0.02178) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.89241 (+/-0.04381) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.90184 (+/-0.03460) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.90056 (+/-0.02211) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.90613 (+/-0.02292) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.90484 (+/-0.02247) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.90527 (+/-0.01980) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.85298 (+/-0.00827) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.84998 (+/-0.00032) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.84955 (+/-0.00193) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.84955 (+/-0.00442) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.84955 (+/-0.00193) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.84998 (+/-0.00032) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.88556 (+/-0.03544) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.87913 (+/-0.01566) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.87998 (+/-0.01897) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.88556 (+/-0.01865) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.88513 (+/-0.01516) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.88470 (+/-0.01778) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.90056 (+/-0.02328) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.88770 (+/-0.02198) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.89027 (+/-0.02293) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.89970 (+/-0.02337) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.90099 (+/-0.02270) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.89970 (+/-0.02270) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.91342 (+/-0.06065) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 3}
0.90227 (+/-0.05183) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.89841 (+/-0.02710) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.90527 (+/-0.01902) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.90527 (+/-0.01759) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.90527 (+/-0.01759) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.92113 (+/-0.06579) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.93356 (+/-0.04852) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.91642 (+/-0.04819) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.92842 (+/-0.03829) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.93013 (+/-0.03129) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.92585 (+/-0.02660) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.89456 (+/-0.02821) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 3}
0.89884 (+/-0.02326) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 10}
0.88941 (+/-0.02199) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 20}
0.90013 (+/-0.02280) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 3}
0.92027 (+/-0.01717) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 10}
0.90699 (+/-0.01212) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 20}
0.90227 (+/-0.02051) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 3}
0.91942 (+/-0.02248) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 10}
0.91470 (+/-0.02282) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 20}
0.90270 (+/-0.01583) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 3}
0.92113 (+/-0.03167) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 10}
0.92327 (+/-0.03412) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 20}
0.91084 (+/-0.03262) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 3}
0.93871 (+/-0.02042) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 10}
0.93999 (+/-0.01823) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 20}
0.88598 (+/-0.02855) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 3}
0.90013 (+/-0.02315) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 10}
```

```
0.89413 (+/-0.02197) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 20}
0.89456 (+/-0.01940) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 3}
0.91427 (+/-0.02832) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 10}
0.90613 (+/-0.01901) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 20}
0.90999 (+/-0.00910) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 3}
0.93185 (+/-0.01864) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 10}
0.92885 (+/-0.01662) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 20}
0.91513 (+/-0.02693) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 3}
0.93742 (+/-0.02132) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 10}
0.93270 (+/-0.02272) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 20}
0.91770 (+/-0.03115) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 3}
0.94471 (+/-0.02478) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 10}
0.94128 (+/-0.02344) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 20}

Best parameters set found on development set: {'bootstrap': False, 'criterio
n': 'entropy', 'max_depth': 8, 'n_estimators': 10}
Best model validation accuracy: 0.944706386626661
```

we are not observing great accuracies with default parameters; this can be a possible implication of class imbalance problem, we need to explore better ways to handle this. I hope to have this issue tackled by cross validation. We need to create more observations to help the model learn the pattern for the class which is in minority.

This can be dealt in two ways:

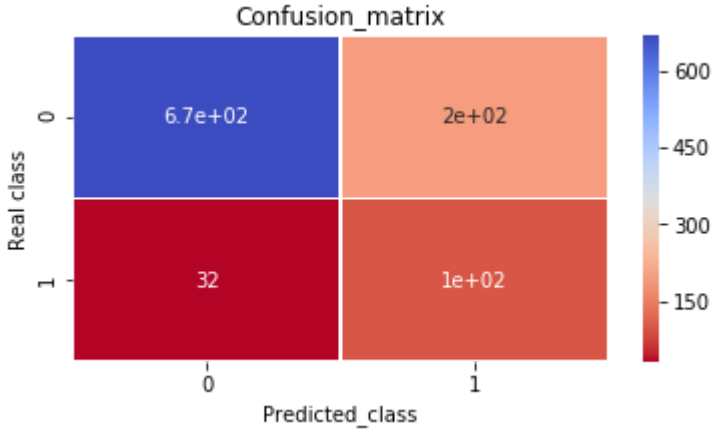
```
- either decrease the samples for majority class
    Issue: as there are less samples for minority class, decreasing the samples for
 majority class would do less good.

- Synthetically increase/generate new samples for minority class
```

In [48]:
```
## Class Imbalance Issue - synthetically generating samples for minority class
 observations
# smote uses k nearest neighbour technique to generate samples for minority cl
ass samples

! pip install imblearn
from imblearn.over_sampling import SMOTE
sm = SMOTE(random_state=123, ratio = 1)

train_x, train_y = sm.fit_sample(train[num_data], train[target])

train_x = pd.DataFrame(train_x, columns=num_data)
train_y = pd.DataFrame(train_y, columns=target)
```

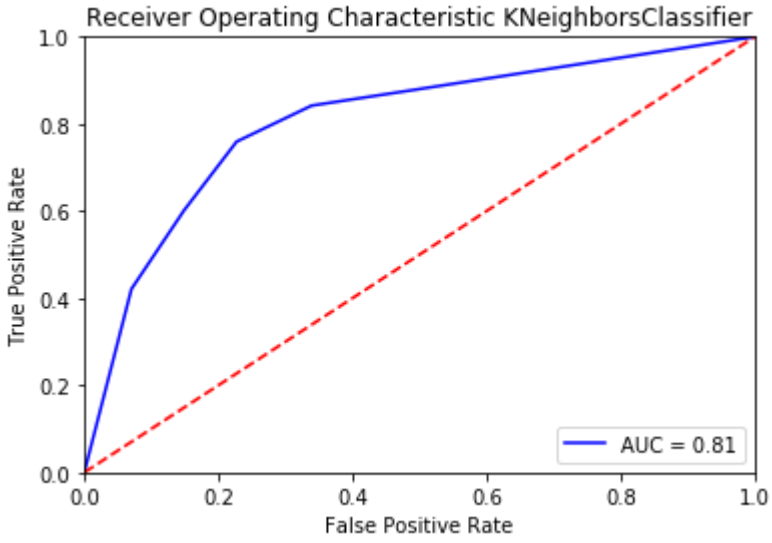
```
Requirement already satisfied: imblearn in /local/home/arorjati/.pyenv/versio
ns/miniconda3-latest/lib/python3.7/site-packages (0.0)
Requirement already satisfied: imbalanced-learn in /local/home/arorjati/.pyen
v/versions/miniconda3-latest/lib/python3.7/site-packages (from imblearn) (0.
3.3)
Requirement already satisfied: numpy in /local/home/arorjati/.pyenv/versions/
miniconda3-latest/lib/python3.7/site-packages (from imbalanced-learn->imblear
n) (1.15.0)
Requirement already satisfied: scikit-learn in /local/home/arorjati/.pyenv/ve
rsions/miniconda3-latest/lib/python3.7/site-packages (from imbalanced-learn->
imblearn) (0.19.2)
Requirement already satisfied: scipy in /local/home/arorjati/.pyenv/versions/
miniconda3-latest/lib/python3.7/site-packages (from imbalanced-learn->imblear
n) (1.1.0)
```

In [49]:
```
# iterate over classifiers
for item in classifiers:
    classifier_name = ((str(item)[:(str(item).find("("))]))
    print (classifier_name)

    # Create classifier, train it and test it.
    clf = item
    print(item)

    clf.fit(train_x, train_y)
    pred = clf.predict(test_x)
    score = clf.score(test_x, test_y)
    report_model(clf, train_x, test_x, train_y,test_y)

    if classifier_name in ["RandomForestClassifier", 'DecisionTreeClassifier'
]:
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        features = train_x.columns

        importances = clf.feature_importances_
        indices = np.argsort(importances)
        plt.title('Feature Importances')
        plt.barh(range(len(indices)), importances[indices], color='b', align=
'center')
        plt.yticks(range(len(indices)), [features[i] for i in indices], fontsi
ze = 6)
        plt.xlabel('Relative Importance')
        plt.figure(figsize=(20,40))
        plt.savefig(classifier_name+ "_feature_imp.png")
        plt.show()


    # calculate the fpr and tpr for all thresholds of the classification
    probs = clf.predict_proba(test_x)
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(test_y, preds)
    roc_auc = auc(fpr, tpr)

    plt.title('Receiver Operating Characteristic '+classifier_name)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig(classifier_name+ "_roc_auc.png")
    plt.show()

    print ("Score: ", round(score,3),"\nAccuracy score: ", round(accuracy_scor
e(test_y, pred), 3),"\nF1 score: ", round(f1_score(test_y, pred), 3), "\n- - -
 - - ", "\n")
```

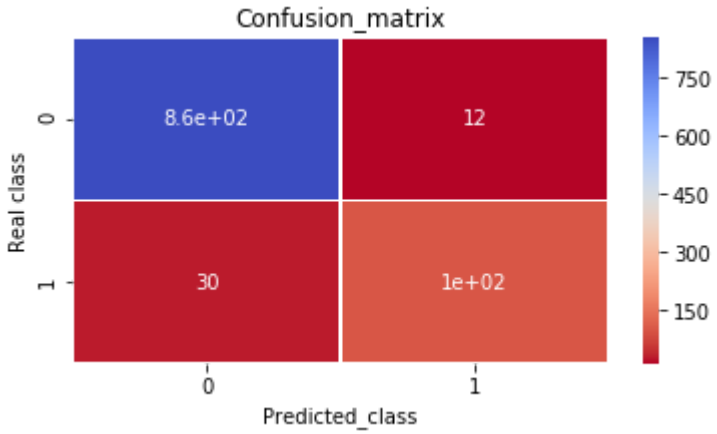
```
KNeighborsClassifier
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=20, n_neighbors=5, p=2,
           weights='uniform')
the recall for this model is : 0.7593984962406015
TP 101
TN 670
FP 197
FN 32
```



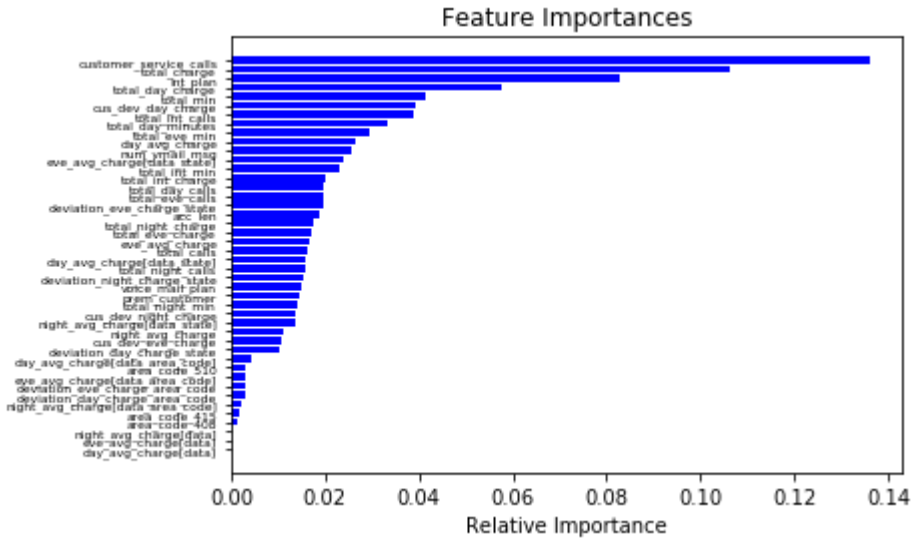

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.95      0.77      0.85       867
       True       0.34      0.76      0.47       133

avg / total       0.87      0.77      0.80      1000
```



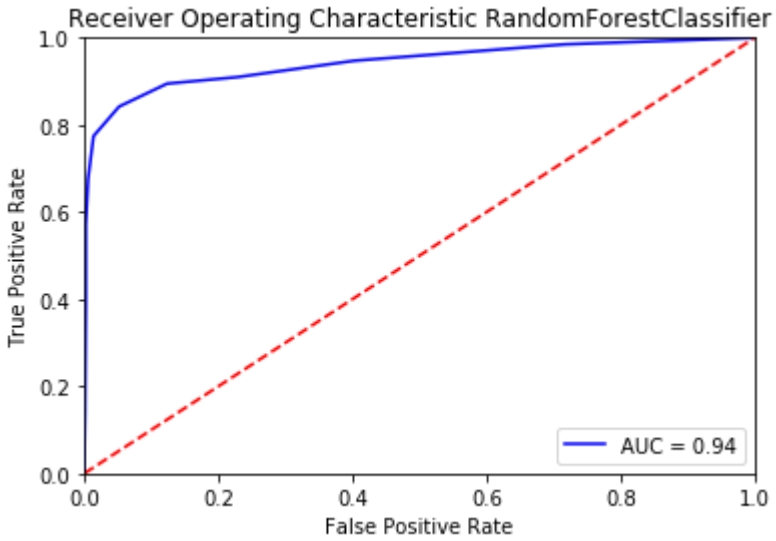
```
Score:  0.771
Accuracy score:  0.771
F1 score:  0.469
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=20,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
the recall for this model is : 0.7744360902255639
TP 103
TN 855
FP 12
FN 30
```




```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.97      0.99      0.98       867
       True       0.90      0.77      0.83       133

avg / total       0.96      0.96      0.96      1000
```

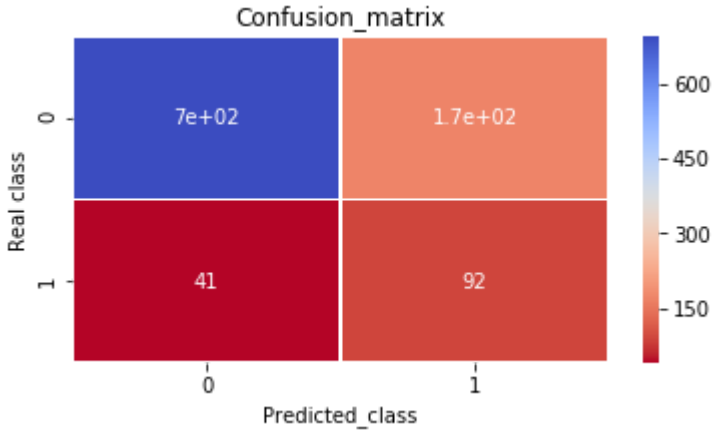


<Figure size 1440x2880 with 0 Axes>

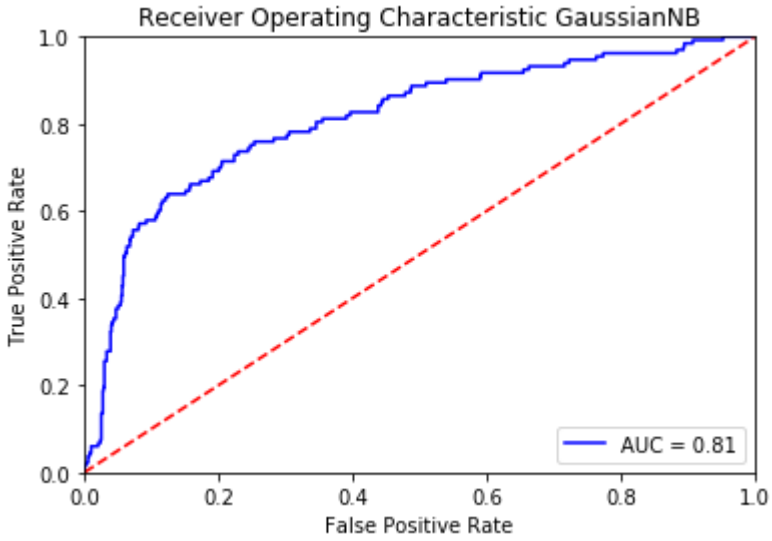


Score: 0.964
Accuracy score: 0.964
F1 score: 0.862
- - - - -

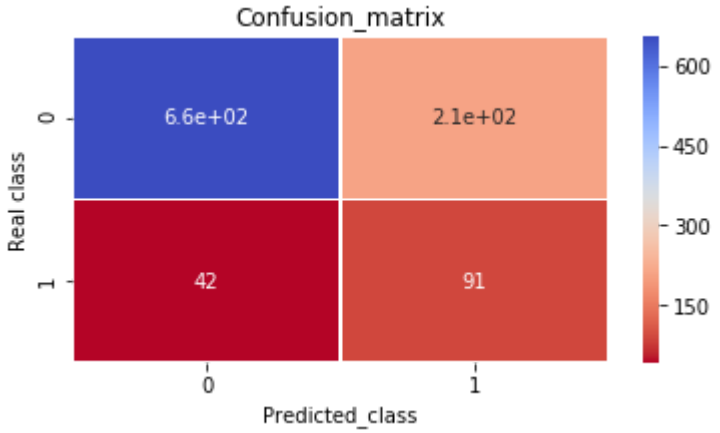
GaussianNB
GaussianNB(priors=None)
the recall for this model is : 0.6917293233082706
TP 92
TN 695
FP 172
FN 41



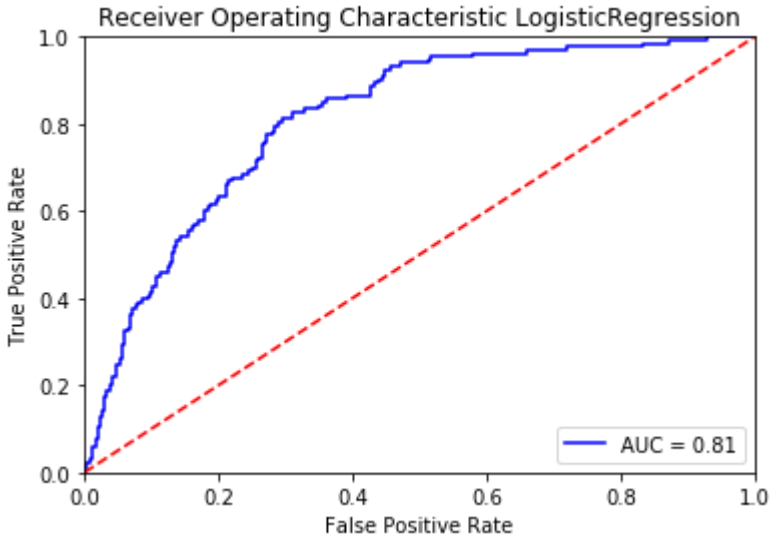

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.94      0.80      0.87       867
       True       0.35      0.69      0.46       133

avg / total       0.87      0.79      0.81      1000
```



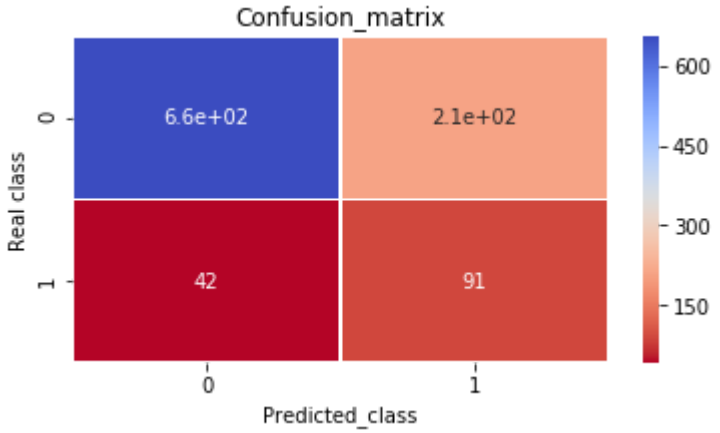

```
Score:  0.787
Accuracy score:  0.787
F1 score:  0.463
- - - - -

LogisticRegression
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=5,
          penalty='l2', random_state=123, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
the recall for this model is : 0.6842105263157895
TP 91
TN 656
FP 211
FN 42
```



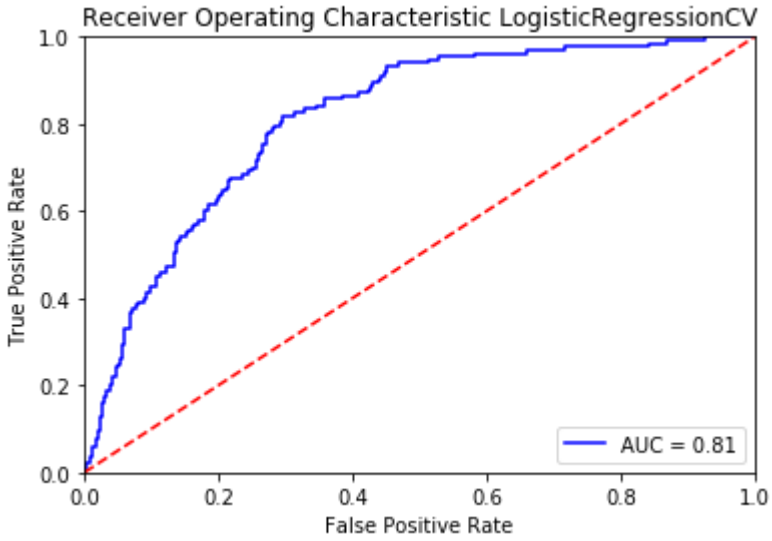

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.94      0.76      0.84       867
       True       0.30      0.68      0.42       133

avg / total       0.85      0.75      0.78      1000
```



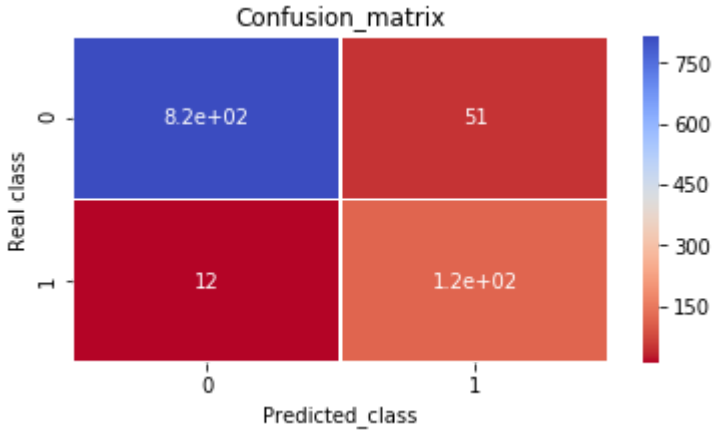

```
Score:  0.747
Accuracy score:  0.747
F1 score:  0.418
- - - - -

LogisticRegressionCV
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=50, penalty='l2', random_state=111,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
the recall for this model is : 0.6842105263157895
TP 91
TN 656
FP 211
FN 42
```



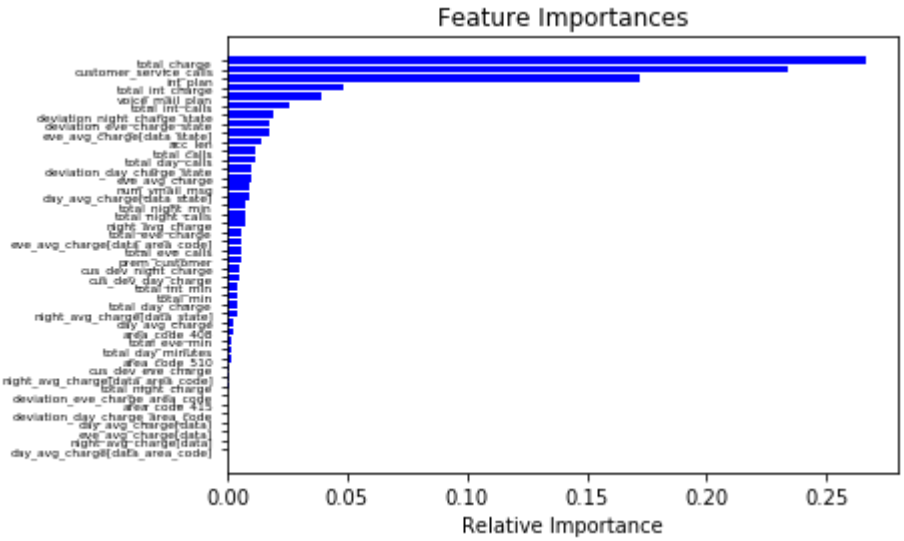

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.94      0.76      0.84       867
       True       0.30      0.68      0.42       133

avg / total       0.85      0.75      0.78      1000
```



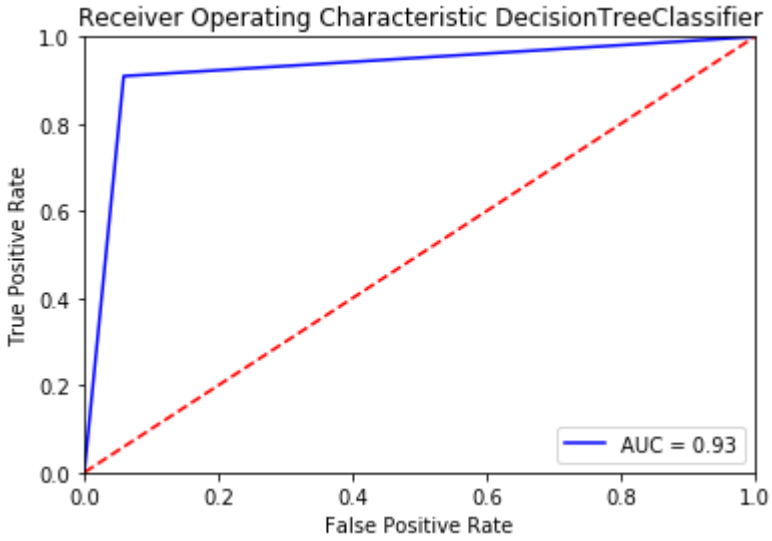

```
Score:  0.747
Accuracy score:  0.747
F1 score:  0.418
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.9097744360902256
TP 121
TN 816
FP 51
FN 12
```



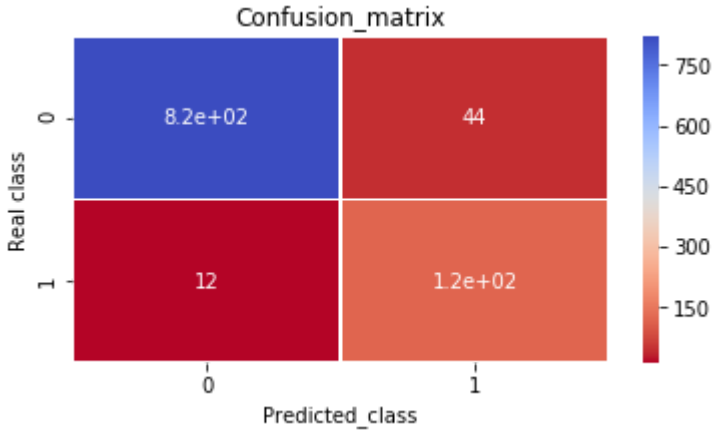

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.99      0.94      0.96       867
       True       0.70      0.91      0.79       133

avg / total       0.95      0.94      0.94      1000
```



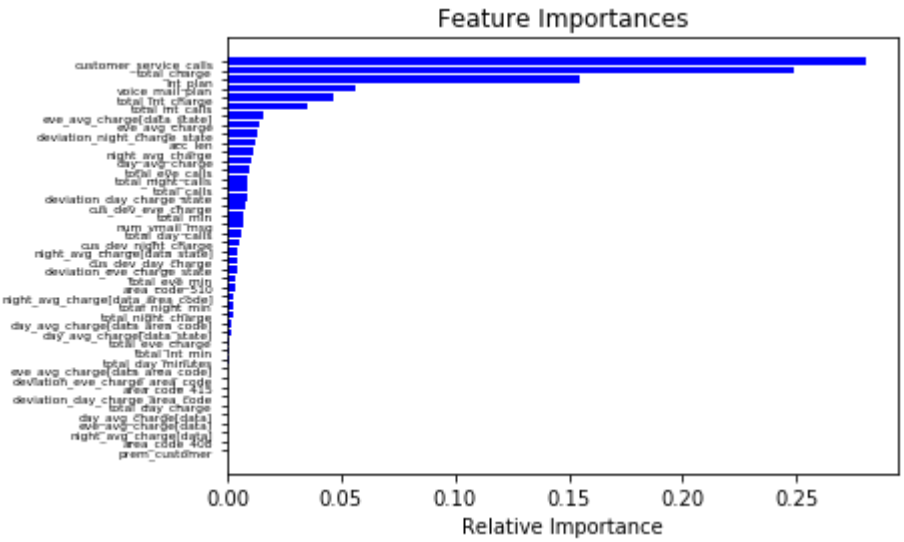

```
<Figure size 1440x2880 with 0 Axes>
```



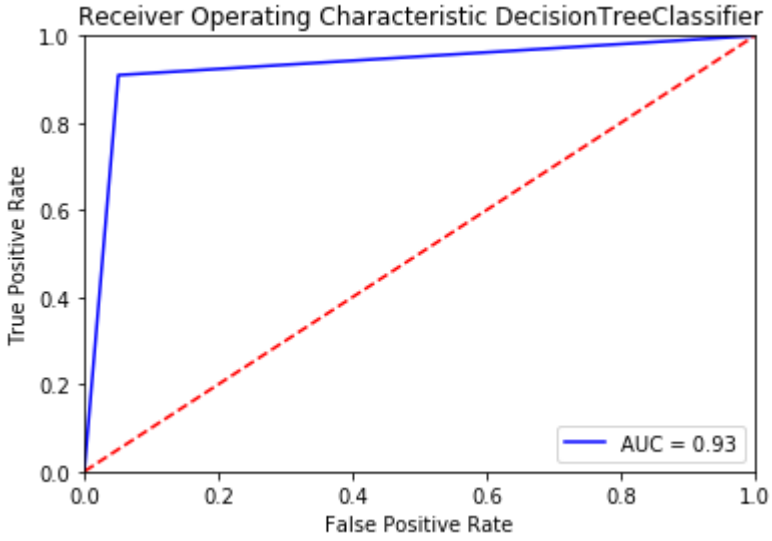

```
Score:  0.936
Accuracy score:  0.936
F1 score:  0.789
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.9097744360902256
TP 121
TN 823
FP 44
FN 12
```



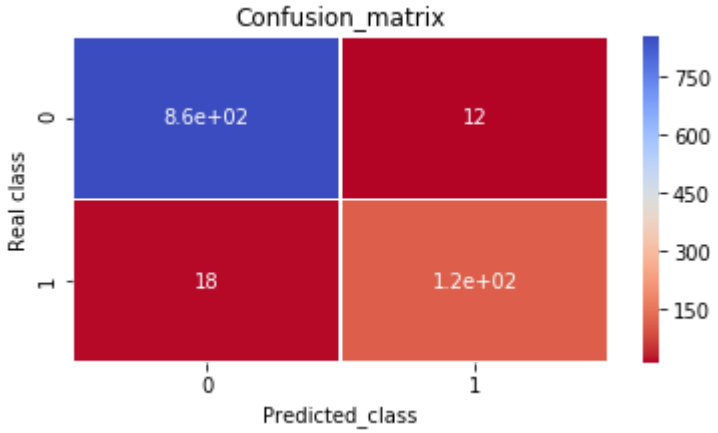
```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.99      0.95      0.97       867
       True       0.73      0.91      0.81       133

avg / total       0.95      0.94      0.95      1000
```



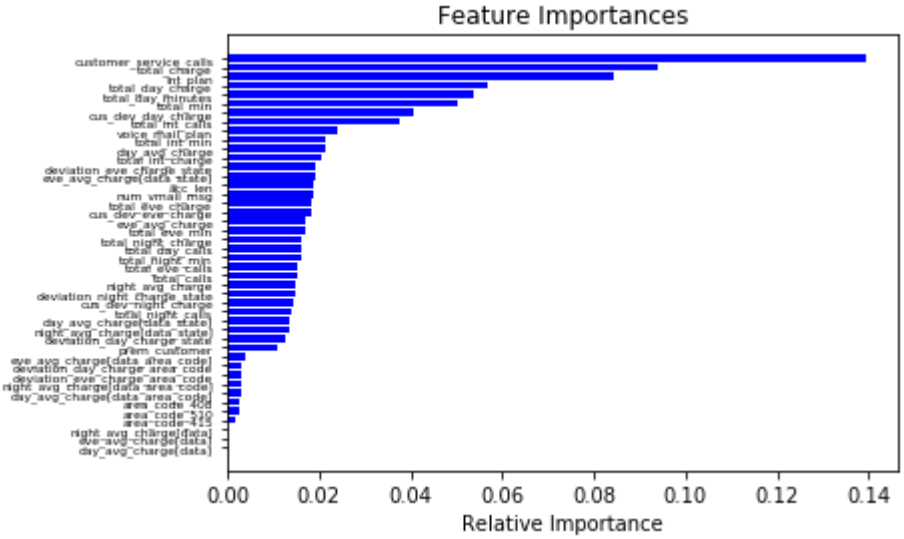
<Figure size 1440x2880 with 0 Axes>



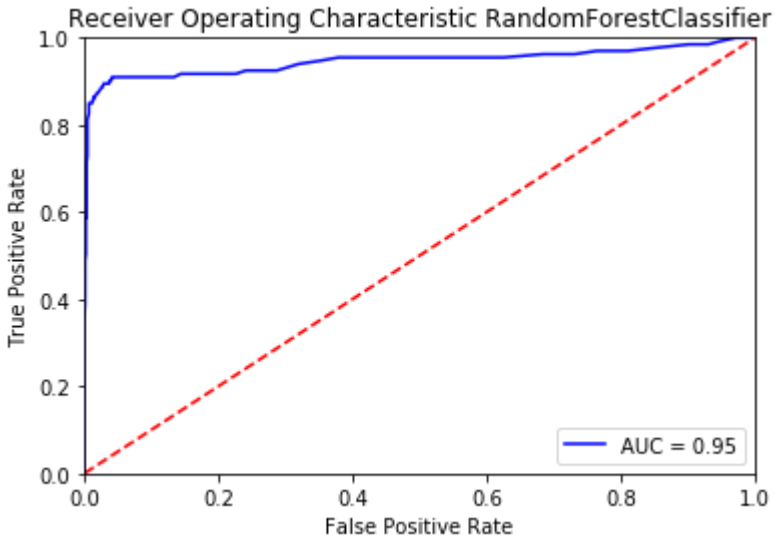
```
Score:  0.942
Accuracy score:  0.942
F1 score:  0.808
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=120, n_jobs=20,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
the recall for this model is : 0.8646616541353384
TP 115
TN 855
FP 12
FN 18
```



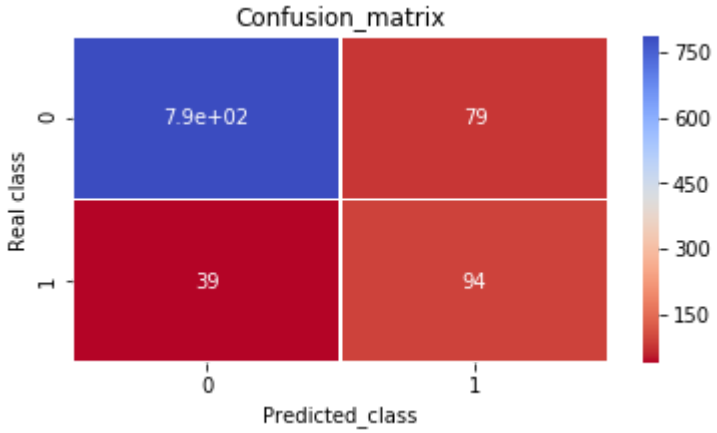

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.98      0.99      0.98       867
       True       0.91      0.86      0.88       133

avg / total       0.97      0.97      0.97      1000
```



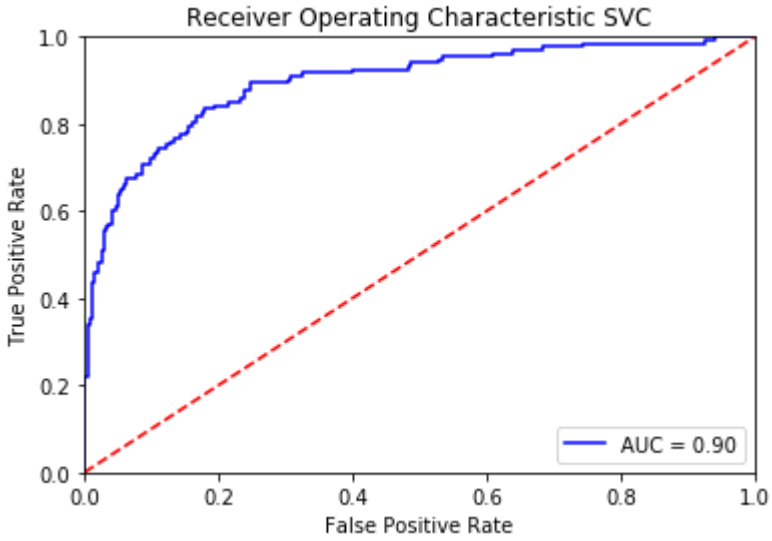
```
<Figure size 1440x2880 with 0 Axes>
```



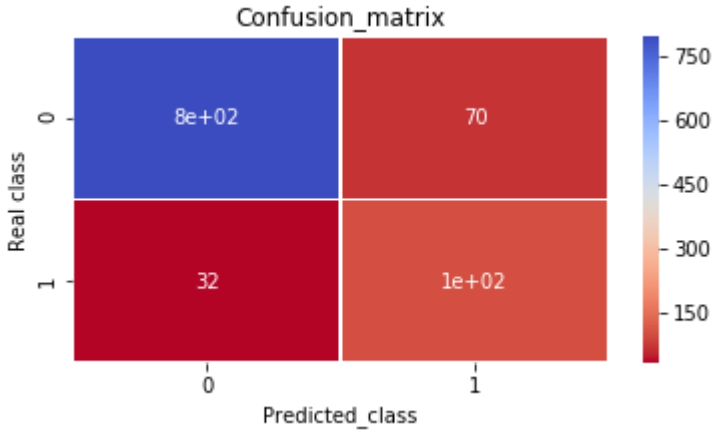

```
Score:  0.97
Accuracy score:  0.97
F1 score:  0.885
- - - - -

SVC
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=123, shrinking=True,
  tol=0.001, verbose=False)
the recall for this model is : 0.706766917293233
TP 94
TN 788
FP 79
FN 39
```



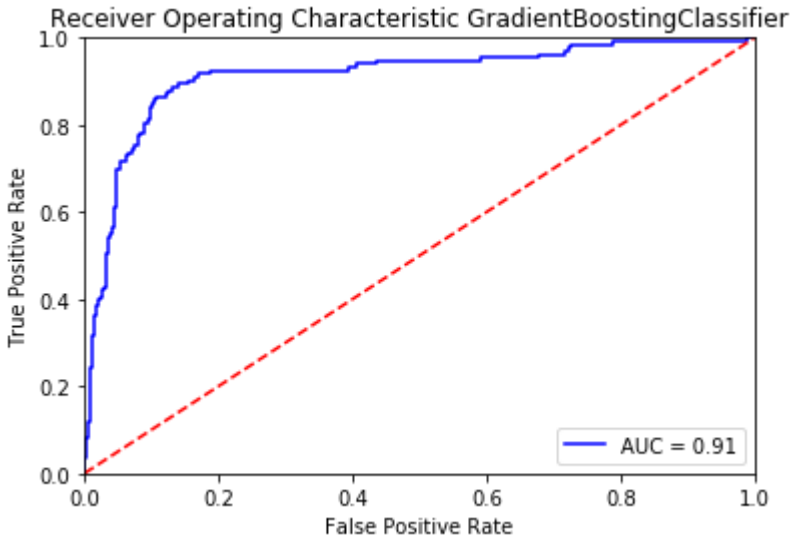

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.95      0.91      0.93       867
       True       0.54      0.71      0.61       133

avg / total       0.90      0.88      0.89      1000
```




```
Score:  0.882
Accuracy score:  0.882
F1 score:  0.614
- - - - -

GradientBoostingClassifier
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1.0, loss='deviance', max_depth=1,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=123, subsample=1.0, verbose=0,
              warm_start=False)
the recall for this model is : 0.7593984962406015
TP 101
TN 797
FP 70
FN 32
```



```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.96      0.92      0.94       867
       True       0.59      0.76      0.66       133

avg / total       0.91      0.90      0.90      1000
```




```
Score:  0.898
Accuracy score:  0.898
F1 score:  0.664
- - - - -
```

In [50]:
```
#Hyper Parameter Tuning with Grid Search and classifier as Random Forest
from sklearn.model_selection import GridSearchCV
model = RandomForestClassifier(random_state=42)

param_grid = [{'n_estimators': [3, 10, 30, 100, 500,1000], 'max_features': ['a
uto', 'sqrt', 'log2', 2, 4, 6, 8],'max_leaf_nodes': [2,4,6,8,16]},
{'bootstrap': [False], 'n_estimators': [3, 10, 20],
 'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']},]

clf = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
clf.fit(train_x, train_y)
# view accuracy scores for all the models
print("Grid scores for all the models based on CV:\n")
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.5f (+/-%0.05f) for %r" % (mean, std * 2, params))
# check out best model performance
print("\nBest parameters set found on development set:", clf.best_params_)
print("Best model validation accuracy:", clf.best_score_)
```

```
Grid scores for all the models based on CV:

0.73323 (+/-0.01464) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.68356 (+/-0.01478) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.66919 (+/-0.01061) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.67726 (+/-0.01071) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.67801 (+/-0.01237) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.68129 (+/-0.00922) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.75088 (+/-0.01894) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.76702 (+/-0.01877) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.75038 (+/-0.03907) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.79627 (+/-0.04720) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.81417 (+/-0.04505) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.81543 (+/-0.04266) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.76904 (+/-0.02525) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.81039 (+/-0.03765) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.82552 (+/-0.04337) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.86258 (+/-0.02166) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.86309 (+/-0.02879) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.87115 (+/-0.02858) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.82879 (+/-0.05253) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.85350 (+/-0.02305) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.86510 (+/-0.02486) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.88149 (+/-0.01724) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.88376 (+/-0.01257) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.88603 (+/-0.01017) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.85174 (+/-0.04441) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.88225 (+/-0.02282) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.88527 (+/-0.02613) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.89284 (+/-0.01600) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 100}
0.89309 (+/-0.01394) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.89309 (+/-0.01293) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.73323 (+/-0.01464) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.68356 (+/-0.01478) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.66919 (+/-0.01061) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.67726 (+/-0.01071) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.67801 (+/-0.01237) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.68129 (+/-0.00922) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.75088 (+/-0.01894) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.76702 (+/-0.01877) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.75038 (+/-0.03907) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.79627 (+/-0.04720) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.81417 (+/-0.04505) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.81543 (+/-0.04266) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.76904 (+/-0.02525) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.81039 (+/-0.03765) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.82552 (+/-0.04337) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.86258 (+/-0.02166) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.86309 (+/-0.02879) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.87115 (+/-0.02858) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.82879 (+/-0.05253) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.85350 (+/-0.02305) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.86510 (+/-0.02486) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.88149 (+/-0.01724) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.88376 (+/-0.01257) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.88603 (+/-0.01017) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.85174 (+/-0.04441) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.88225 (+/-0.02282) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 10}
```

```
0.88527 (+/-0.02613) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.89284 (+/-0.01600) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.89309 (+/-0.01394) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.89309 (+/-0.01293) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.73323 (+/-0.01464) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.72340 (+/-0.01565) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.68053 (+/-0.01907) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.66969 (+/-0.01642) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.68053 (+/-0.01240) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.68331 (+/-0.01036) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.85880 (+/-0.01315) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.82375 (+/-0.02168) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.78921 (+/-0.03945) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.77509 (+/-0.04385) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.79123 (+/-0.03097) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.79778 (+/-0.04245) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.88099 (+/-0.03123) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.87670 (+/-0.03028) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.87368 (+/-0.01973) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.86460 (+/-0.03394) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.86662 (+/-0.02625) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.87191 (+/-0.02785) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.88200 (+/-0.02118) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.88855 (+/-0.01153) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.88628 (+/-0.01504) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.87821 (+/-0.02005) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.88023 (+/-0.01724) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.88174 (+/-0.01378) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.88099 (+/-0.01456) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 3}
0.89536 (+/-0.01093) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.89536 (+/-0.01081) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.88729 (+/-0.01397) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.88906 (+/-0.01347) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.89032 (+/-0.01162) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.73248 (+/-0.01436) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.71533 (+/-0.03018) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.67877 (+/-0.02256) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.70298 (+/-0.03145) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.73500 (+/-0.02866) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.73323 (+/-0.02502) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.74181 (+/-0.01485) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.78341 (+/-0.04099) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.74281 (+/-0.03275) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.75189 (+/-0.04410) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.77862 (+/-0.02620) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.78038 (+/-0.03075) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.76349 (+/-0.02217) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.77660 (+/-0.02952) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.79274 (+/-0.04125) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.81367 (+/-0.04819) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.82073 (+/-0.04303) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.81694 (+/-0.04408) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.76425 (+/-0.04135) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.82703 (+/-0.02315) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.82073 (+/-0.03079) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.84014 (+/-0.04180) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.84165 (+/-0.03960) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
```

```
0.83938 (+/-0.04625) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.80459 (+/-0.05242) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.84544 (+/-0.03095) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.85653 (+/-0.02597) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.86586 (+/-0.03841) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.86989 (+/-0.03862) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.87090 (+/-0.04063) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.73323 (+/-0.01464) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.72315 (+/-0.03187) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.68230 (+/-0.02058) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.67499 (+/-0.01453) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.68028 (+/-0.01339) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.68432 (+/-0.01167) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.74231 (+/-0.02296) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.76273 (+/-0.01558) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.75517 (+/-0.02920) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.79425 (+/-0.04524) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.77181 (+/-0.02394) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.76147 (+/-0.02232) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.76072 (+/-0.01543) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.81745 (+/-0.02577) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.83964 (+/-0.03069) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.85477 (+/-0.03818) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.86334 (+/-0.03256) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.85603 (+/-0.03935) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.76147 (+/-0.03092) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.84619 (+/-0.02147) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.86334 (+/-0.02220) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.87015 (+/-0.04246) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 100}
0.87368 (+/-0.03112) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.87141 (+/-0.03208) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.78694 (+/-0.04131) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.86788 (+/-0.02461) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.88048 (+/-0.02312) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.88578 (+/-0.02684) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.88654 (+/-0.01731) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.88603 (+/-0.02036) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.73323 (+/-0.01464) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.68356 (+/-0.01478) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.66919 (+/-0.01061) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.67726 (+/-0.01071) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.67801 (+/-0.01237) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.68129 (+/-0.00922) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.75088 (+/-0.01894) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.76702 (+/-0.01877) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.75038 (+/-0.03907) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.79627 (+/-0.04720) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.81417 (+/-0.04505) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.81543 (+/-0.04266) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.76904 (+/-0.02525) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.81039 (+/-0.03765) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.82552 (+/-0.04337) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.86258 (+/-0.02166) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.86309 (+/-0.02879) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.87115 (+/-0.02858) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.82879 (+/-0.05253) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.85350 (+/-0.02305) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
```

```
0.86510 (+/-0.02486) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.88149 (+/-0.01724) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.88376 (+/-0.01257) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.88603 (+/-0.01017) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.85174 (+/-0.04441) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.88225 (+/-0.02282) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.88527 (+/-0.02613) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.89284 (+/-0.01600) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.89309 (+/-0.01394) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.89309 (+/-0.01293) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.75164 (+/-0.01576) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.68457 (+/-0.02211) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.68533 (+/-0.02790) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.67574 (+/-0.01291) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.67650 (+/-0.01374) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.67700 (+/-0.01258) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.77105 (+/-0.00970) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.76223 (+/-0.04135) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.75088 (+/-0.07422) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.83106 (+/-0.04515) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.83661 (+/-0.04109) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.84771 (+/-0.04364) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.86132 (+/-0.08090) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.88048 (+/-0.00766) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.87569 (+/-0.01795) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.87872 (+/-0.01132) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.88275 (+/-0.01138) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.88477 (+/-0.00856) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.86460 (+/-0.08251) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 3}
0.88527 (+/-0.00772) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.88225 (+/-0.01928) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.88351 (+/-0.01176) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.88628 (+/-0.01365) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.88855 (+/-0.01282) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.88099 (+/-0.02483) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.89057 (+/-0.01474) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.89107 (+/-0.01256) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.89259 (+/-0.01394) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.89410 (+/-0.01503) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.89486 (+/-0.01511) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.77837 (+/-0.02273) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 3}
0.88275 (+/-0.02036) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 10}
0.88200 (+/-0.01282) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 20}
0.85149 (+/-0.05725) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 3}
0.88931 (+/-0.01901) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 10}
0.89334 (+/-0.01570) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 20}
0.85603 (+/-0.04626) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 3}
0.89939 (+/-0.01250) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 10}
0.90519 (+/-0.01087) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 20}
0.87821 (+/-0.02838) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 3}
0.91024 (+/-0.01750) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 10}
0.90973 (+/-0.01303) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 20}
0.87771 (+/-0.06408) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 3}
0.91604 (+/-0.01311) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 10}
0.91654 (+/-0.01357) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 20}
0.77534 (+/-0.01629) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 3}
0.87872 (+/-0.02553) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 10}
```

```
0.88452 (+/-0.01881) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 20}
0.86208 (+/-0.05731) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 3}
0.88931 (+/-0.01568) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 10}
0.89612 (+/-0.02041) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 20}
0.84796 (+/-0.02787) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 3}
0.89183 (+/-0.01830) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 10}
0.90318 (+/-0.01158) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 20}
0.87922 (+/-0.03558) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 3}
0.90772 (+/-0.01090) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 10}
0.91377 (+/-0.01180) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 20}
0.89082 (+/-0.04836) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 3}
0.91049 (+/-0.01425) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 10}
0.91452 (+/-0.01780) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 20}

Best parameters set found on development set: {'bootstrap': False, 'criterio
n': 'gini', 'max_depth': 8, 'n_estimators': 20}
Best model validation accuracy: 0.9165405950579929
```

In [51]:
```
#SMOTE 2

sm = SMOTE(random_state=123, ratio = 1)

Z_x, Z_y = sm.fit_sample(data[num_data], data[target])

Z_x = pd.DataFrame(Z_x, columns=num_data)
Z_y = pd.DataFrame(Z_y, columns=target)
```

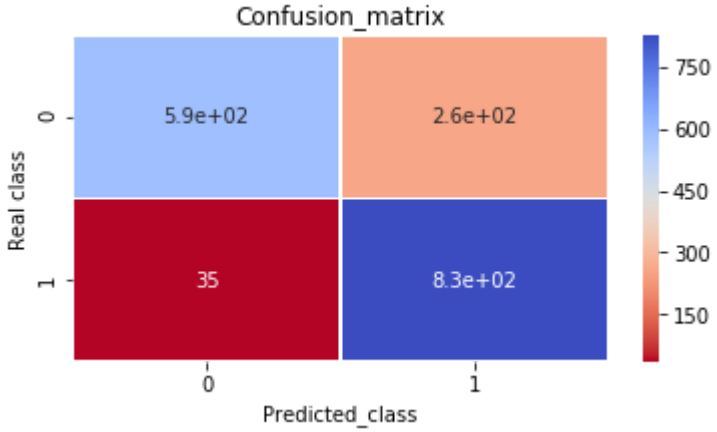
In [52]:
```
X = pd.concat([Z_x,Z_y],axis=1)
```

In [53]:
```
#Data Split 70:30
from sklearn.cross_validation import train_test_split

train, test = sklearn.cross_validation.train_test_split(X, train_size = 0.7, r
andom_state=123)
```

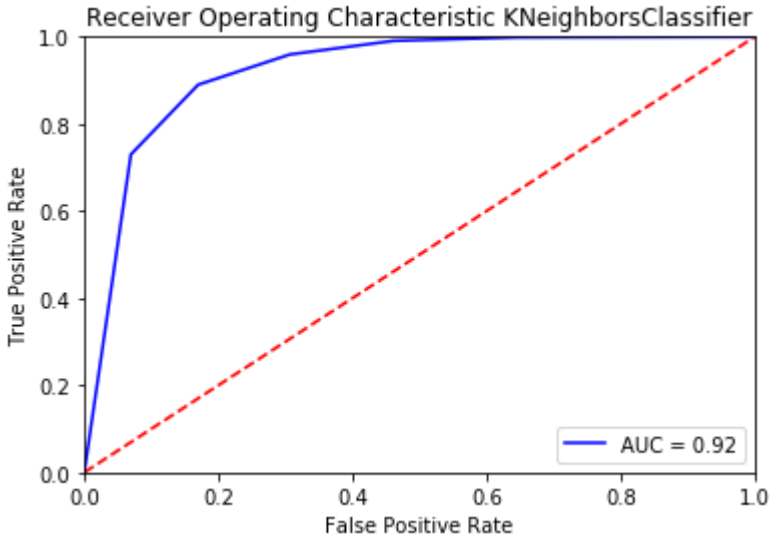
In [54]:
```
train_x = train[num_data]
test_x = test[num_data]

train_y =train[target]
test_y =test[target]
```

In [55]:
```
# iterate over classifiers
for item in classifiers:
    classifier_name = ((str(item)[:(str(item).find("("))]))
    print (classifier_name)

    # Create classifier, train it and test it.
    clf = item
    print(item)

    clf.fit(train_x, train_y)
    pred = clf.predict(test_x)
    score = clf.score(test_x, test_y)
    report_model(clf, train_x, test_x, train_y,test_y)

    if classifier_name in ["RandomForestClassifier", 'DecisionTreeClassifier'
]:
        %matplotlib inline
        import matplotlib.pyplot as plt
        import numpy as np
        features = train_x.columns

        importances = clf.feature_importances_
        indices = np.argsort(importances)
        plt.title('Feature Importances')
        plt.barh(range(len(indices)), importances[indices], color='b', align=
'center')
        plt.yticks(range(len(indices)), [features[i] for i in indices], fontsi
ze = 6)
        plt.xlabel('Relative Importance')
        plt.figure(figsize=(20,40))
        plt.savefig(classifier_name+ "_feature_imp.png")
        plt.show()


    # calculate the fpr and tpr for all thresholds of the classification
    probs = clf.predict_proba(test_x)
    preds = probs[:,1]
    fpr, tpr, threshold = roc_curve(test_y, preds)
    roc_auc = auc(fpr, tpr)

    plt.title('Receiver Operating Characteristic '+classifier_name)
    plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
    plt.legend(loc = 'lower right')
    plt.plot([0, 1], [0, 1],'r--')
    plt.xlim([0, 1])
    plt.ylim([0, 1])
    plt.ylabel('True Positive Rate')
    plt.xlabel('False Positive Rate')
    plt.savefig(classifier_name+ "_roc_auc.png")
    plt.show()

    print ("Score: ", round(score,3),"\nAccuracy score: ", round(accuracy_scor
e(test_y, pred), 3),"\nF1 score: ", round(f1_score(test_y, pred), 3), "\n- - -
 - - ", "\n")
```

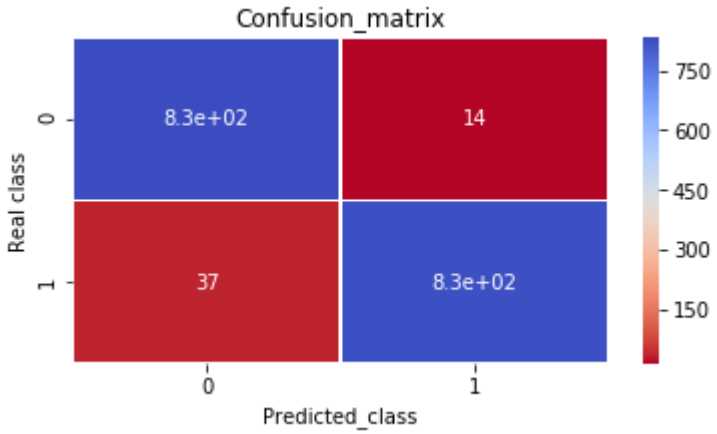
```
KNeighborsClassifier
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=20, n_neighbors=5, p=2,
           weights='uniform')
the recall for this model is : 0.9594438006952491
TP 828
TN 588
FP 260
FN 35
```



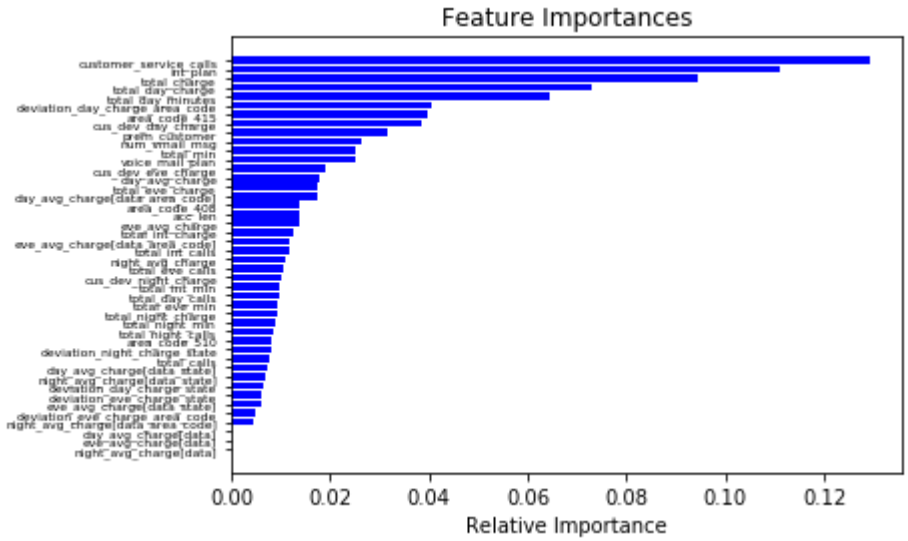

```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.94      0.69      0.80       848
       True       0.76      0.96      0.85       863

avg / total       0.85      0.83      0.82      1711
```



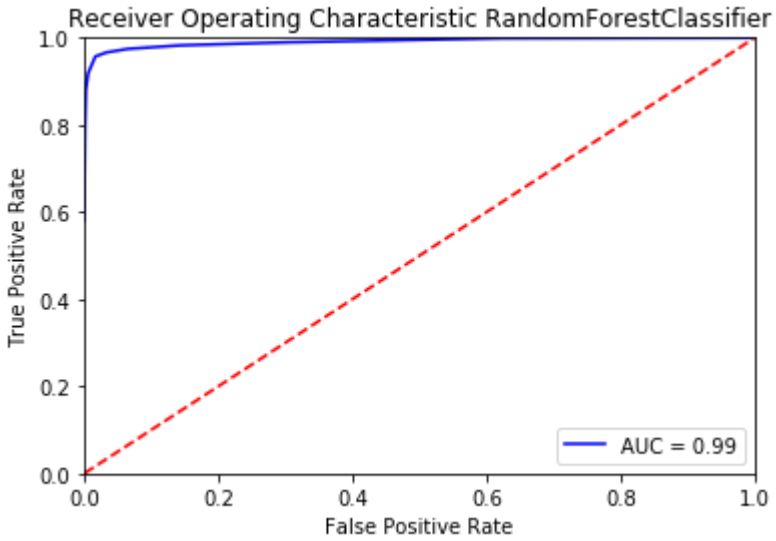
```
Score:  0.828
Accuracy score:  0.828
F1 score:  0.849
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=20,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
the recall for this model is : 0.9571263035921205
TP 826
TN 834
FP 14
FN 37
```




```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.96      0.98      0.97       848
       True       0.98      0.96      0.97       863

avg / total       0.97      0.97      0.97      1711
```

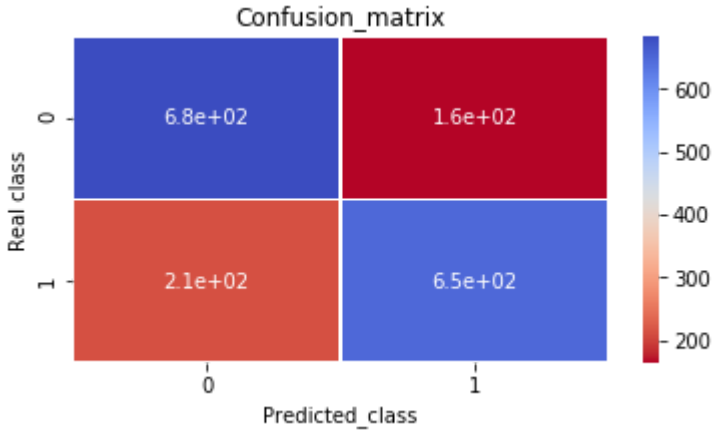


<Figure size 1440x2880 with 0 Axes>

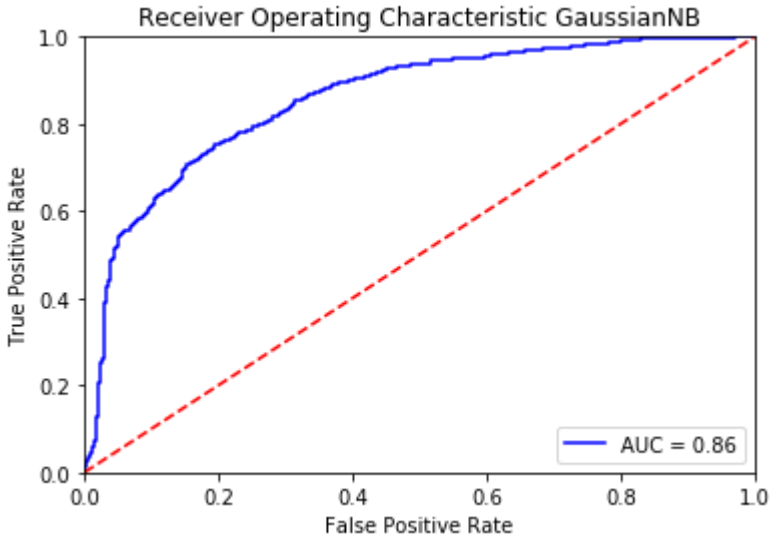


Score: 0.97
Accuracy score: 0.97
F1 score: 0.97
- - - - -

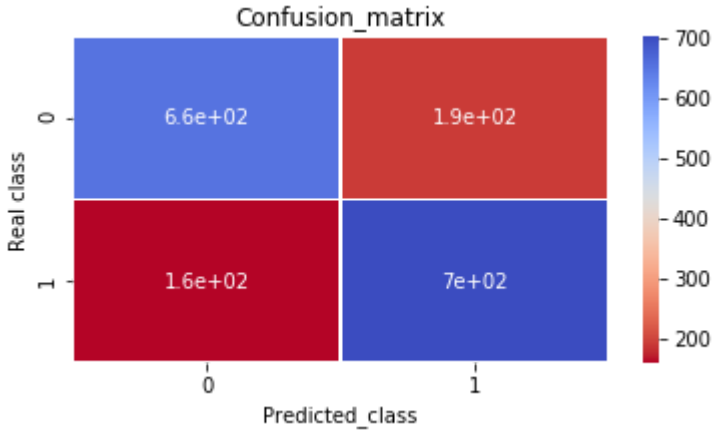
GaussianNB
GaussianNB(priors=None)
the recall for this model is : 0.7531865585168018
TP 650
TN 684
FP 164
FN 213



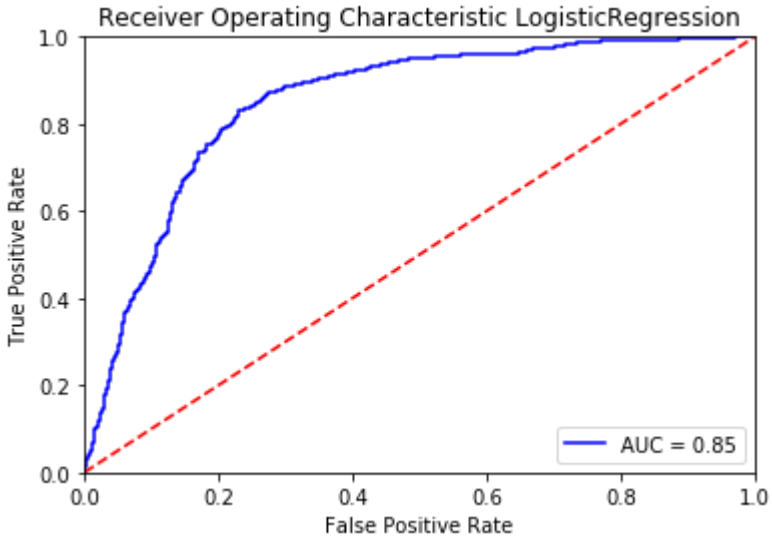

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.76      0.81      0.78       848
       True       0.80      0.75      0.78       863

avg / total       0.78      0.78      0.78      1711
```



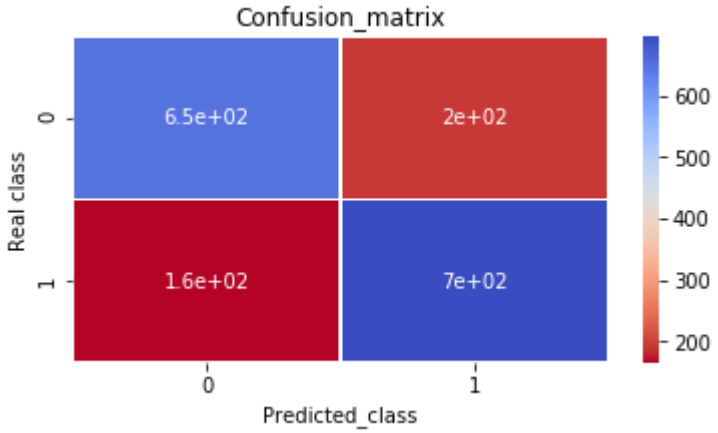

```
Score:  0.78
Accuracy score:  0.78
F1 score:  0.775
- - - - -

LogisticRegression
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=5,
          penalty='l2', random_state=123, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
the recall for this model is : 0.8157589803012746
TP 704
TN 655
FP 193
FN 159
```



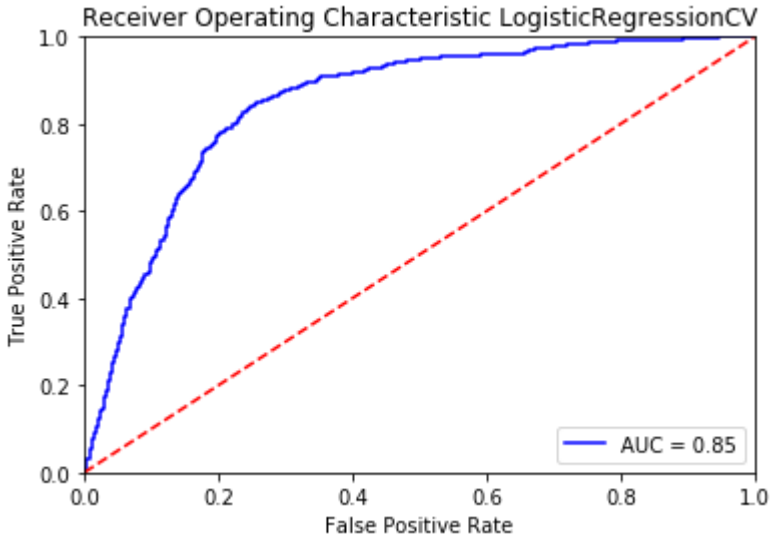

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.80      0.77      0.79       848
       True       0.78      0.82      0.80       863

avg / total       0.79      0.79      0.79      1711
```



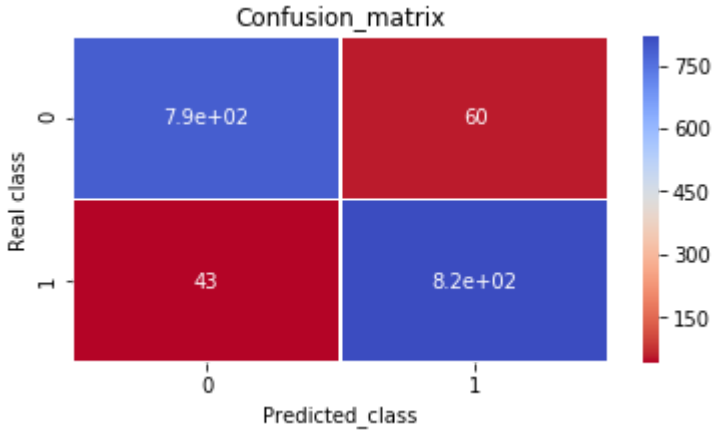

```
Score:  0.794
Accuracy score:  0.794
F1 score:  0.8
- - - - -

LogisticRegressionCV
LogisticRegressionCV(Cs=10, class_weight=None, cv=None, dual=False,
           fit_intercept=True, intercept_scaling=1.0, max_iter=100,
           multi_class='ovr', n_jobs=50, penalty='l2', random_state=111,
           refit=True, scoring=None, solver='lbfgs', tol=0.0001, verbose=0)
the recall for this model is : 0.8088064889918888
TP 698
TN 652
FP 196
FN 165
```



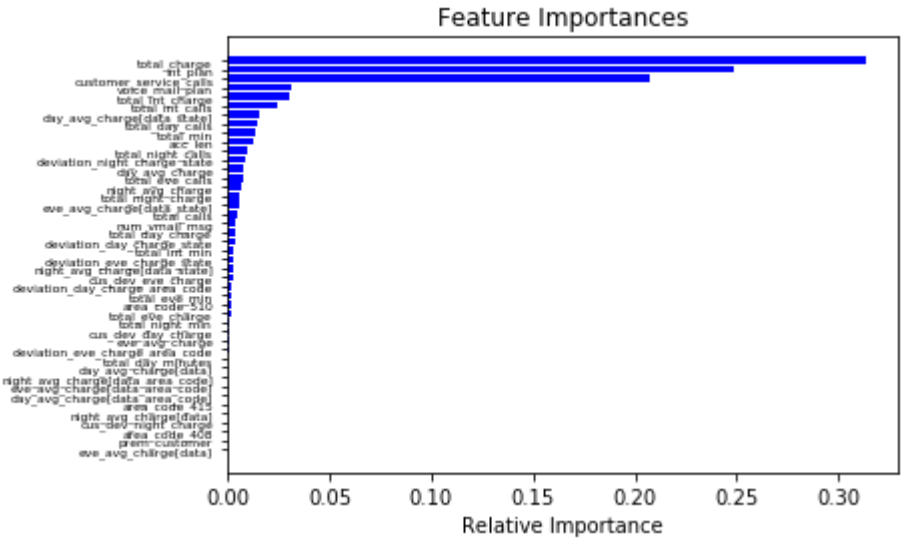

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.80      0.77      0.78       848
       True       0.78      0.81      0.79       863

avg / total       0.79      0.79      0.79      1711
```



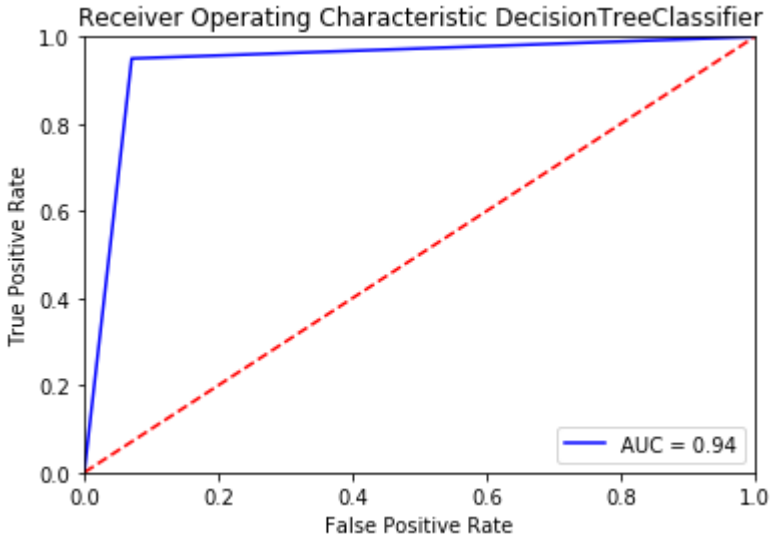

```
Score:  0.789
Accuracy score:  0.789
F1 score:  0.795
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.9501738122827347
TP 820
TN 788
FP 60
FN 43
```



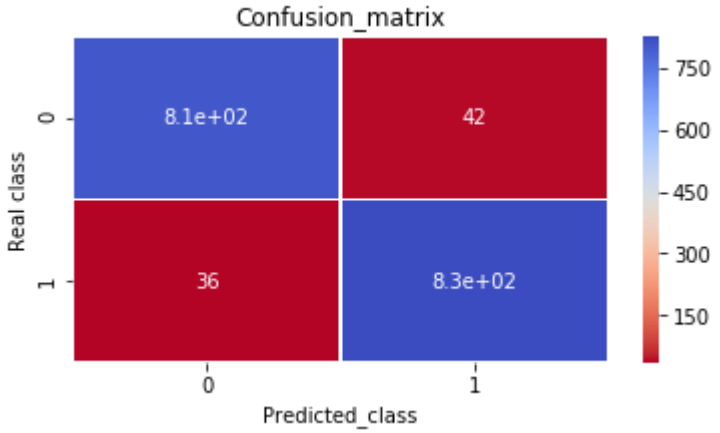

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.95      0.93      0.94       848
       True       0.93      0.95      0.94       863

avg / total       0.94      0.94      0.94      1711
```



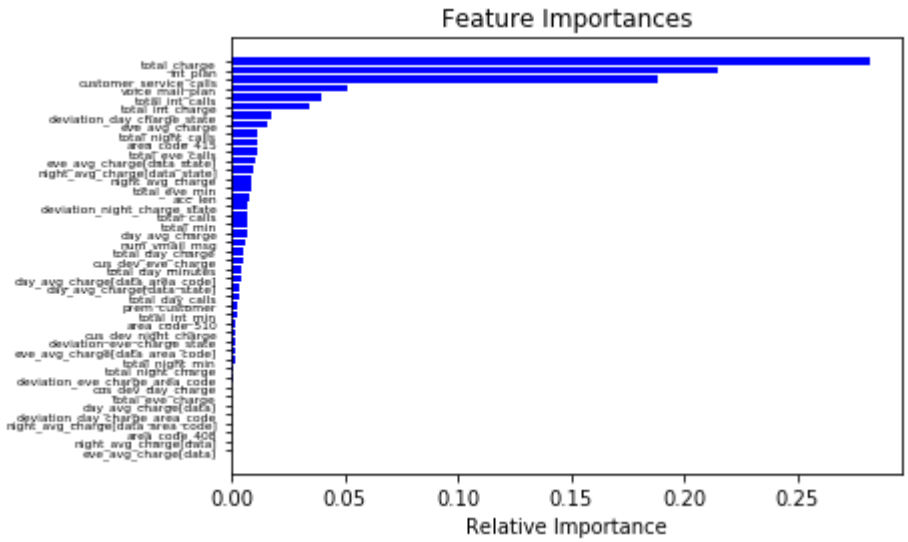

```
<Figure size 1440x2880 with 0 Axes>
```



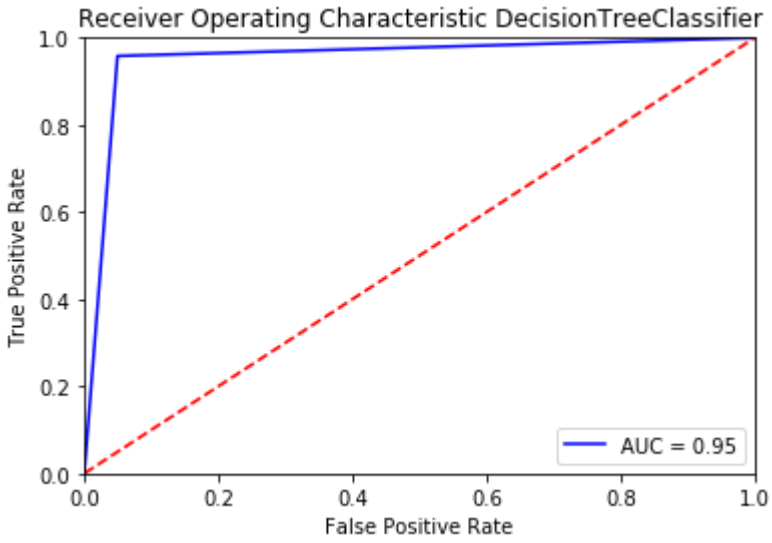

```
Score:  0.943
Accuracy score:  0.943
F1 score:  0.944
- - - - -

DecisionTreeClassifier
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=Non
e,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
the recall for this model is : 0.9582850521436849
TP 827
TN 806
FP 42
FN 36
```



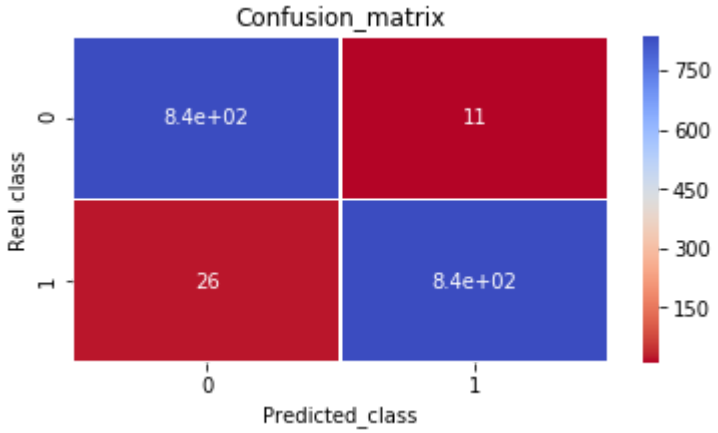
```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.96      0.95      0.95       848
       True       0.95      0.96      0.95       863

avg / total       0.95      0.95      0.95      1711
```



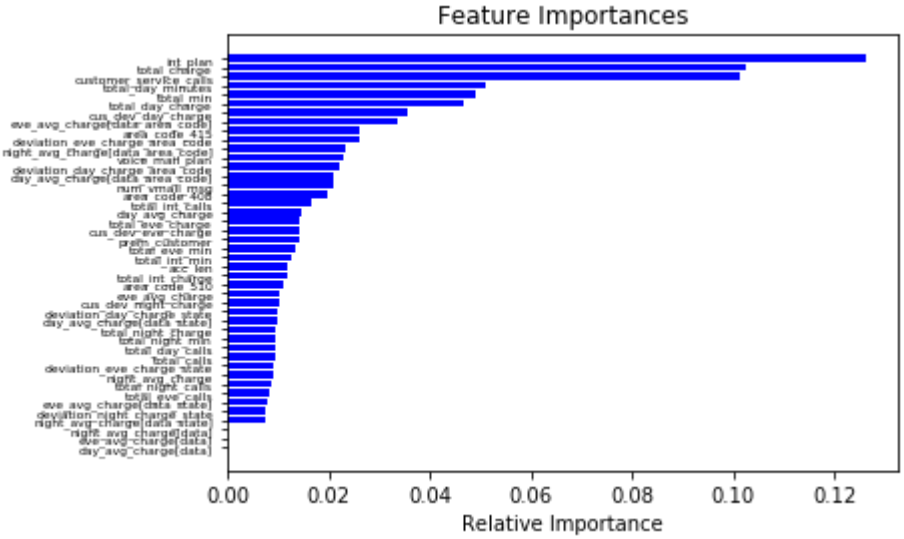

```
<Figure size 1440x2880 with 0 Axes>
```



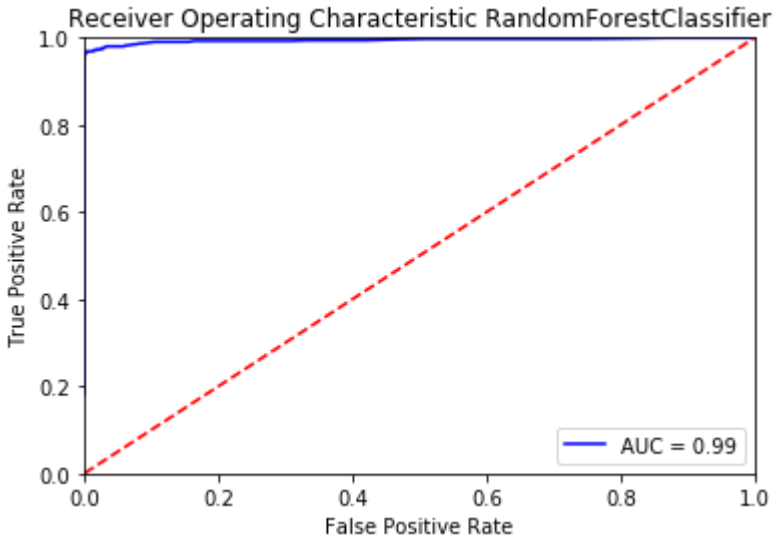
```
Score:  0.956
Accuracy score:  0.956
F1 score:  0.957
- - - - -

RandomForestClassifier
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=100, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=120, n_jobs=20,
            oob_score=False, random_state=123, verbose=0, warm_start=False)
the recall for this model is : 0.9698725376593279
TP 837
TN 837
FP 11
FN 26
```



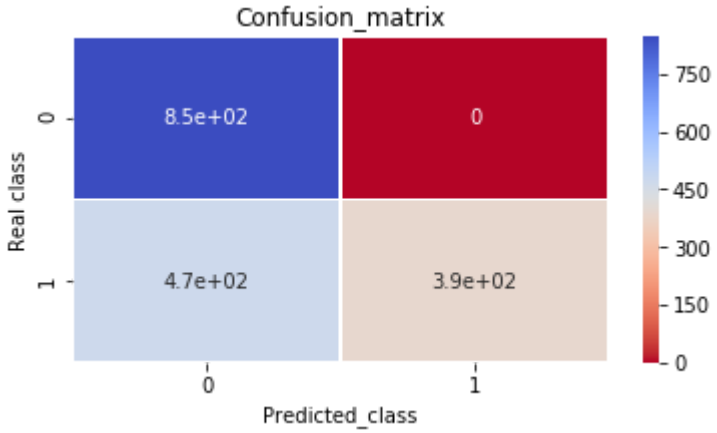

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.97      0.99      0.98       848
       True       0.99      0.97      0.98       863

avg / total       0.98      0.98      0.98      1711
```



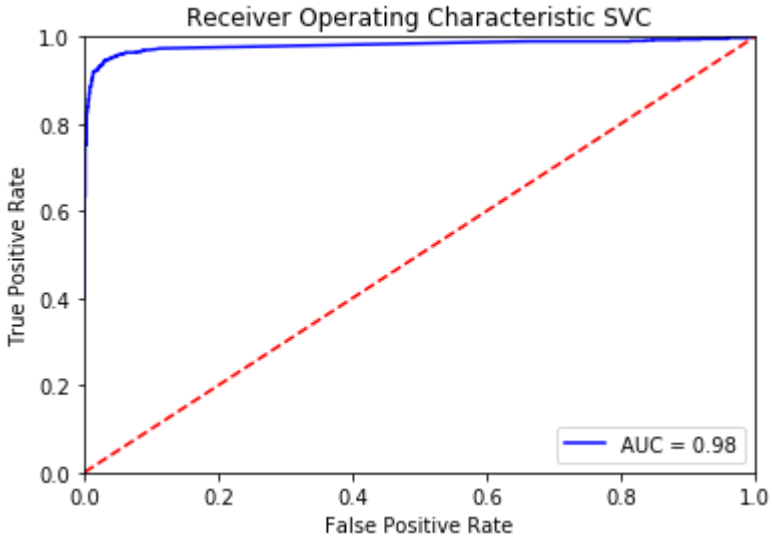
```
<Figure size 1440x2880 with 0 Axes>
```



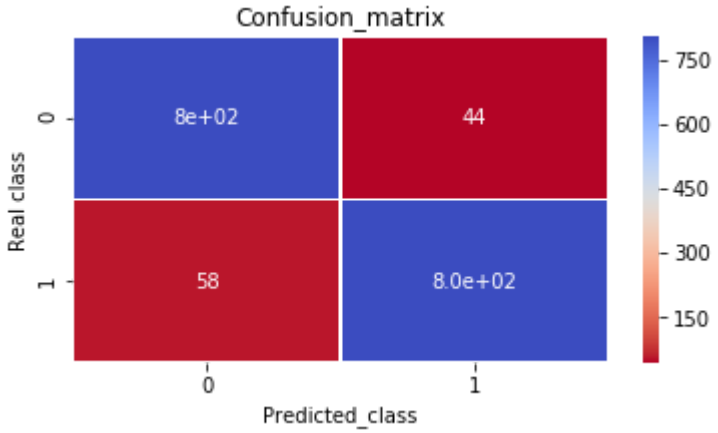

```
Score:  0.978
Accuracy score:  0.978
F1 score:  0.978
- - - - -

SVC
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=True, random_state=123, shrinking=True,
  tol=0.001, verbose=False)
the recall for this model is : 0.4519119351100811
TP 390
TN 848
FP 0
FN 473
```



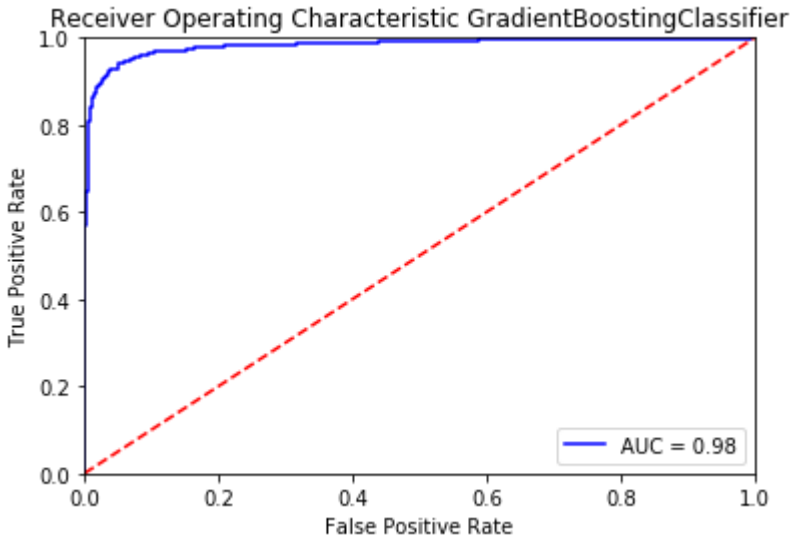

```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.64      1.00      0.78       848
       True       1.00      0.45      0.62       863

avg / total       0.82      0.72      0.70      1711
```




```
Score:  0.724
Accuracy score:  0.724
F1 score:  0.623
- - - - -

GradientBoostingClassifier
GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=1.0, loss='deviance', max_depth=1,
              max_features=None, max_leaf_nodes=None,
              min_impurity_decrease=0.0, min_impurity_split=None,
              min_samples_leaf=1, min_samples_split=2,
              min_weight_fraction_leaf=0.0, n_estimators=100,
              presort='auto', random_state=123, subsample=1.0, verbose=0,
              warm_start=False)
the recall for this model is : 0.93279258400927
TP 805
TN 804
FP 44
FN 58
```



```
----------Classification Report--------------------------------------
             precision    recall  f1-score   support

      False       0.93      0.95      0.94       848
       True       0.95      0.93      0.94       863

avg / total       0.94      0.94      0.94      1711
```




```
Score:  0.94
Accuracy score:  0.94
F1 score:  0.94
- - - - -
```

In [56]:
```
#Hyper Parameter Tuning with Grid Search and classifier as Random Forest
model = RandomForestClassifier(random_state=42)

param_grid = [{'n_estimators': [3, 10, 30, 100, 500,1000], 'max_features': ['a
uto', 'sqrt', 'log2', 2, 4, 6, 8],'max_leaf_nodes': [2,4,6,8,16]},
{'bootstrap': [False], 'n_estimators': [3, 10, 20],
 'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']},]

clf = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
clf.fit(train_x, train_y)
# view accuracy scores for all the models
print("Grid scores for all the models based on CV:\n")
means = clf.cv_results_['mean_test_score']
stds = clf.cv_results_['std_test_score']
for mean, std, params in zip(means, stds, clf.cv_results_['params']):
    print("%0.5f (+/-%0.05f) for %r" % (mean, std * 2, params))
# check out best model performance
print("\nBest parameters set found on development set:", clf.best_params_)
print("Best model validation accuracy:", clf.best_score_)
```

```
Grid scores for all the models based on CV:

0.77839 (+/-0.02392) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.76636 (+/-0.02293) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.76886 (+/-0.02125) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.78215 (+/-0.01738) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.77814 (+/-0.02854) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.78716 (+/-0.02576) for {'max_features': 'auto', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.80998 (+/-0.05849) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.84357 (+/-0.00819) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.84758 (+/-0.03152) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.86062 (+/-0.01624) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.86538 (+/-0.01789) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.86513 (+/-0.02043) for {'max_features': 'auto', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.81223 (+/-0.06029) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.86212 (+/-0.00872) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.86713 (+/-0.03350) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88393 (+/-0.01930) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88945 (+/-0.02232) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.89120 (+/-0.02210) for {'max_features': 'auto', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.82978 (+/-0.03968) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.88142 (+/-0.02454) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.88518 (+/-0.01897) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.89596 (+/-0.02763) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.90674 (+/-0.03772) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.90674 (+/-0.03455) for {'max_features': 'auto', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.86864 (+/-0.02573) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.90449 (+/-0.03642) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.91502 (+/-0.02057) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.93282 (+/-0.02103) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 100}
0.94209 (+/-0.02057) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.94209 (+/-0.02013) for {'max_features': 'auto', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.77839 (+/-0.02392) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.76636 (+/-0.02293) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.76886 (+/-0.02125) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.78215 (+/-0.01738) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.77814 (+/-0.02854) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.78716 (+/-0.02576) for {'max_features': 'sqrt', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.80998 (+/-0.05849) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.84357 (+/-0.00819) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.84758 (+/-0.03152) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.86062 (+/-0.01624) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.86538 (+/-0.01789) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.86513 (+/-0.02043) for {'max_features': 'sqrt', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.81223 (+/-0.06029) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.86212 (+/-0.00872) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.86713 (+/-0.03350) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88393 (+/-0.01930) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88945 (+/-0.02232) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.89120 (+/-0.02210) for {'max_features': 'sqrt', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.82978 (+/-0.03968) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.88142 (+/-0.02454) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.88518 (+/-0.01897) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.89596 (+/-0.02763) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.90674 (+/-0.03772) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.90674 (+/-0.03455) for {'max_features': 'sqrt', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.86864 (+/-0.02573) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 3}
0.90449 (+/-0.03642) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 10}
```

```
0.91502 (+/-0.02057) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.93282 (+/-0.02103) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.94209 (+/-0.02057) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.94209 (+/-0.02013) for {'max_features': 'sqrt', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.77839 (+/-0.02392) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 3}
0.76912 (+/-0.03332) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 10}
0.77137 (+/-0.03075) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 30}
0.78391 (+/-0.01690) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 100}
0.77914 (+/-0.02023) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 500}
0.78315 (+/-0.02584) for {'max_features': 'log2', 'max_leaf_nodes': 2, 'n_est
imators': 1000}
0.87992 (+/-0.02846) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 3}
0.86914 (+/-0.01803) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 10}
0.85159 (+/-0.02442) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 30}
0.86087 (+/-0.02204) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 100}
0.86187 (+/-0.01559) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 500}
0.86613 (+/-0.01890) for {'max_features': 'log2', 'max_leaf_nodes': 4, 'n_est
imators': 1000}
0.89697 (+/-0.02917) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 3}
0.87791 (+/-0.02468) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 10}
0.87616 (+/-0.02627) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 30}
0.88343 (+/-0.02093) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 100}
0.88343 (+/-0.02148) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 500}
0.88393 (+/-0.02260) for {'max_features': 'log2', 'max_leaf_nodes': 6, 'n_est
imators': 1000}
0.90875 (+/-0.01510) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 3}
0.89747 (+/-0.02216) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 10}
0.89747 (+/-0.02862) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 30}
0.88920 (+/-0.02641) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 100}
0.89270 (+/-0.02607) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 500}
0.89145 (+/-0.02489) for {'max_features': 'log2', 'max_leaf_nodes': 8, 'n_est
imators': 1000}
0.91802 (+/-0.01644) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
```

```
timators': 3}
0.91050 (+/-0.01393) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 10}
0.92279 (+/-0.01953) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 30}
0.92128 (+/-0.02551) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 100}
0.92855 (+/-0.02144) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 500}
0.93056 (+/-0.01905) for {'max_features': 'log2', 'max_leaf_nodes': 16, 'n_es
timators': 1000}
0.77839 (+/-0.02392) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.76134 (+/-0.01041) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.74254 (+/-0.03653) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.80722 (+/-0.02320) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.82101 (+/-0.01831) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.82552 (+/-0.02338) for {'max_features': 2, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.83956 (+/-0.03032) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.85435 (+/-0.02550) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.85285 (+/-0.02821) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.86237 (+/-0.02128) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.86262 (+/-0.02605) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.86388 (+/-0.02223) for {'max_features': 2, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.84257 (+/-0.03039) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.86739 (+/-0.02622) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.86037 (+/-0.01607) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.87867 (+/-0.02293) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.87591 (+/-0.02078) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.87516 (+/-0.02616) for {'max_features': 2, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.86137 (+/-0.03825) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.88669 (+/-0.03300) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.87992 (+/-0.02437) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.88343 (+/-0.02272) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.88443 (+/-0.02221) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
```

```
0.88368 (+/-0.02176) for {'max_features': 2, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.87691 (+/-0.04512) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.89797 (+/-0.03461) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.89697 (+/-0.02012) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.89822 (+/-0.02289) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.89521 (+/-0.02363) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.89446 (+/-0.02138) for {'max_features': 2, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.77839 (+/-0.02392) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.77413 (+/-0.02931) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.76385 (+/-0.01867) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.79243 (+/-0.01621) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.78967 (+/-0.01976) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.78867 (+/-0.01566) for {'max_features': 4, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.83805 (+/-0.01596) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.84908 (+/-0.02622) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.84257 (+/-0.01953) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.87491 (+/-0.02431) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.86789 (+/-0.02156) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.86789 (+/-0.02016) for {'max_features': 4, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.85184 (+/-0.04251) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.86964 (+/-0.03174) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.86312 (+/-0.02502) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.88393 (+/-0.02125) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.88067 (+/-0.02368) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.88193 (+/-0.02493) for {'max_features': 4, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.86337 (+/-0.03615) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.88744 (+/-0.02925) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.88142 (+/-0.02990) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.89421 (+/-0.01671) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 100}
0.88920 (+/-0.02591) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.89145 (+/-0.02020) for {'max_features': 4, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.89321 (+/-0.01380) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.90524 (+/-0.02883) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.90950 (+/-0.02841) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.92254 (+/-0.02134) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.91702 (+/-0.02396) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.91652 (+/-0.02326) for {'max_features': 4, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.77839 (+/-0.02392) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.76636 (+/-0.02293) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.76886 (+/-0.02125) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.78215 (+/-0.01738) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.77814 (+/-0.02854) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.78716 (+/-0.02576) for {'max_features': 6, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.80998 (+/-0.05849) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.84357 (+/-0.00819) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.84758 (+/-0.03152) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.86062 (+/-0.01624) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.86538 (+/-0.01789) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.86513 (+/-0.02043) for {'max_features': 6, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.81223 (+/-0.06029) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.86212 (+/-0.00872) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.86713 (+/-0.03350) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.88393 (+/-0.01930) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.88945 (+/-0.02232) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.89120 (+/-0.02210) for {'max_features': 6, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.82978 (+/-0.03968) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 3}
0.88142 (+/-0.02454) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
```

```
0.88518 (+/-0.01897) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.89596 (+/-0.02763) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.90674 (+/-0.03772) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.90674 (+/-0.03455) for {'max_features': 6, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.86864 (+/-0.02573) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.90449 (+/-0.03642) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.91502 (+/-0.02057) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.93282 (+/-0.02103) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.94209 (+/-0.02057) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.94209 (+/-0.02013) for {'max_features': 6, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.80045 (+/-0.02601) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 3}
0.80446 (+/-0.03446) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 10}
0.76836 (+/-0.05139) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 30}
0.74380 (+/-0.02383) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 100}
0.76260 (+/-0.05054) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 500}
0.78591 (+/-0.01396) for {'max_features': 8, 'max_leaf_nodes': 2, 'n_estimato
rs': 1000}
0.83329 (+/-0.02526) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 3}
0.84207 (+/-0.04071) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 10}
0.85159 (+/-0.02043) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 30}
0.85510 (+/-0.02583) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 100}
0.86538 (+/-0.02476) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 500}
0.86713 (+/-0.02625) for {'max_features': 8, 'max_leaf_nodes': 4, 'n_estimato
rs': 1000}
0.88092 (+/-0.03319) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 3}
0.90724 (+/-0.01774) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 10}
0.88493 (+/-0.02934) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 30}
0.88468 (+/-0.02801) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 100}
0.90424 (+/-0.02638) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 500}
0.90223 (+/-0.02957) for {'max_features': 8, 'max_leaf_nodes': 6, 'n_estimato
rs': 1000}
0.88117 (+/-0.00661) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
```

```
rs': 3}
0.90800 (+/-0.02062) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 10}
0.89747 (+/-0.03051) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 30}
0.91677 (+/-0.02530) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 100}
0.92454 (+/-0.02476) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 500}
0.92379 (+/-0.02414) for {'max_features': 8, 'max_leaf_nodes': 8, 'n_estimato
rs': 1000}
0.91025 (+/-0.01670) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 3}
0.93708 (+/-0.01712) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 10}
0.93983 (+/-0.02261) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 30}
0.93783 (+/-0.02512) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 100}
0.94610 (+/-0.02322) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 500}
0.94660 (+/-0.02276) for {'max_features': 8, 'max_leaf_nodes': 16, 'n_estimat
ors': 1000}
0.87917 (+/-0.01781) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 3}
0.88368 (+/-0.02628) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 10}
0.89647 (+/-0.02205) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 4, 'n_estimators': 20}
0.89446 (+/-0.04137) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 3}
0.91176 (+/-0.02636) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 10}
0.92655 (+/-0.01733) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 5, 'n_estimators': 20}
0.89421 (+/-0.03564) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 3}
0.93407 (+/-0.03089) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 10}
0.93507 (+/-0.03196) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 6, 'n_estimators': 20}
0.91702 (+/-0.03340) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 3}
0.93983 (+/-0.01973) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 10}
0.94410 (+/-0.02684) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 7, 'n_estimators': 20}
0.93507 (+/-0.01406) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 3}
0.95488 (+/-0.01885) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 10}
0.95914 (+/-0.01336) for {'bootstrap': False, 'criterion': 'gini', 'max_dept
h': 8, 'n_estimators': 20}
0.88193 (+/-0.05270) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 3}
0.89095 (+/-0.01998) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 10}
```

```
0.88894 (+/-0.02132) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 4, 'n_estimators': 20}
0.89120 (+/-0.03275) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 3}
0.92454 (+/-0.04521) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 10}
0.92981 (+/-0.03371) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 5, 'n_estimators': 20}
0.90800 (+/-0.03279) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 3}
0.93858 (+/-0.01967) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 10}
0.93983 (+/-0.01617) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 6, 'n_estimators': 20}
0.92229 (+/-0.02820) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 3}
0.94284 (+/-0.02119) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 10}
0.95337 (+/-0.01709) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 7, 'n_estimators': 20}
0.92479 (+/-0.02293) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 3}
0.95613 (+/-0.02032) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 10}
0.95839 (+/-0.01716) for {'bootstrap': False, 'criterion': 'entropy', 'max_de
pth': 8, 'n_estimators': 20}

Best parameters set found on development set: {'bootstrap': False, 'criterio
n': 'gini', 'max_depth': 8, 'n_estimators': 20}
Best model validation accuracy: 0.9591376284783154
```

Model Selection & Validation

Best parameters set found on development set: {'bootstrap': False, 'criterion': 'entropy', 'max_depth': 8, 'n_estimators': 10} Best model validation accuracy: 0.944706386626661

In [57]:
```
data = data.fillna(0)
final_data = data.copy()
final_data = final_data.fillna(0)
Z = final_data.copy()
Z.head()
```

Out[57]:

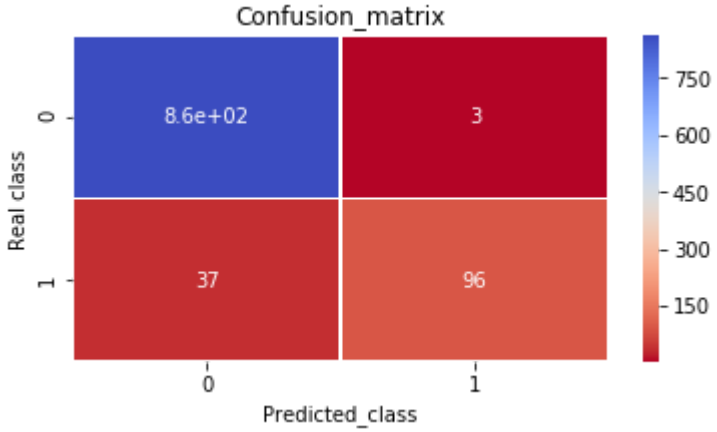
	id	state	acc_len	area_code	int_plan	voice_mail_plan	num_vmail_msg	total_da
0	CUST-1	KS	128	415	0	1	25	265.1
1	CUST-369	KS	132	415	0	0	0	83.4
2	CUST-380	KS	127	415	0	0	0	221.0
3	CUST-386	KS	137	415	0	0	0	230.2
4	CUST-620	KS	110	415	1	0	0	293.3

5 rows × 48 columns

In [58]:
```
#Data Split 70:30
from sklearn.cross_validation import train_test_split

train, test = sklearn.cross_validation.train_test_split(Z, train_size = 0.7, r
andom_state=123)

train_x = train[num_data]
test_x = test[num_data]

train_y =train[target]
test_y =test[target]
```

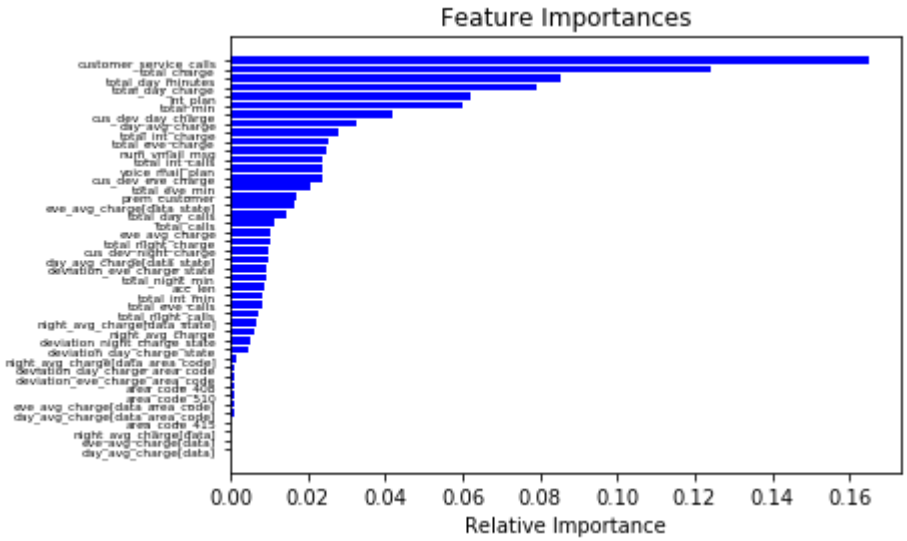
In [59]:
```
clf = RandomForestClassifier(random_state=42, max_depth=8, n_estimators=10, cr
iterion='entropy', bootstrap=False)

clf.fit(train_x, train_y)
pred = clf.predict(test_x)
score = clf.score(test_x, test_y)
report_model(clf, train_x, test_x, train_y,test_y)
```

```
the recall for this model is : 0.7218045112781954
TP 96
TN 864
FP 3
FN 37
```

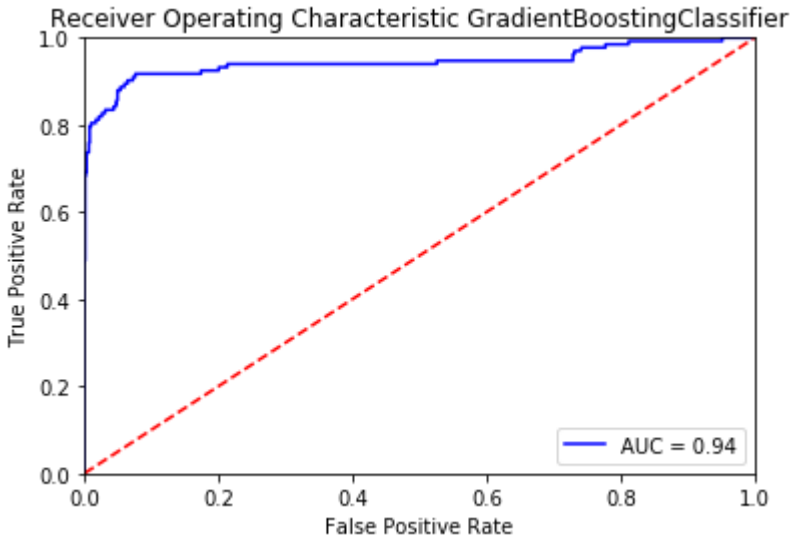



```
----------Classification Report------------------------------------
             precision    recall  f1-score   support

      False       0.96      1.00      0.98       867
       True       0.97      0.72      0.83       133

avg / total       0.96      0.96      0.96      1000
```

In [61]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
features = train_x.columns

importances = clf.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices], fontsize = 6)
plt.xlabel('Relative Importance')
plt.figure(figsize=(20,40))
plt.savefig(classifier_name+ "_feature_imp.png")
plt.show()


# calculate the fpr and tpr for all thresholds of the classification
probs = clf.predict_proba(test_x)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(test_y, preds)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic '+classifier_name)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig(classifier_name+ "_roc_auc.png")
plt.show()

print ("Score: ", round(score,3),"\nAccuracy score: ", round(accuracy_score(te
st_y, pred), 3),"\nF1 score: ", round(f1_score(test_y, pred), 3), "\n- - - - -
 ", "\n")
```




```
<Figure size 1440x2880 with 0 Axes>
```

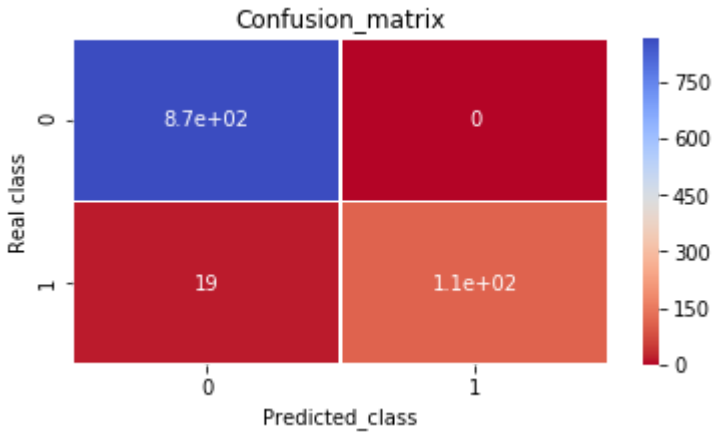



```
Score:  0.96
Accuracy score:  0.96
F1 score:  0.828
- - - - -
```

In [ ]:
```
#Best Model
```

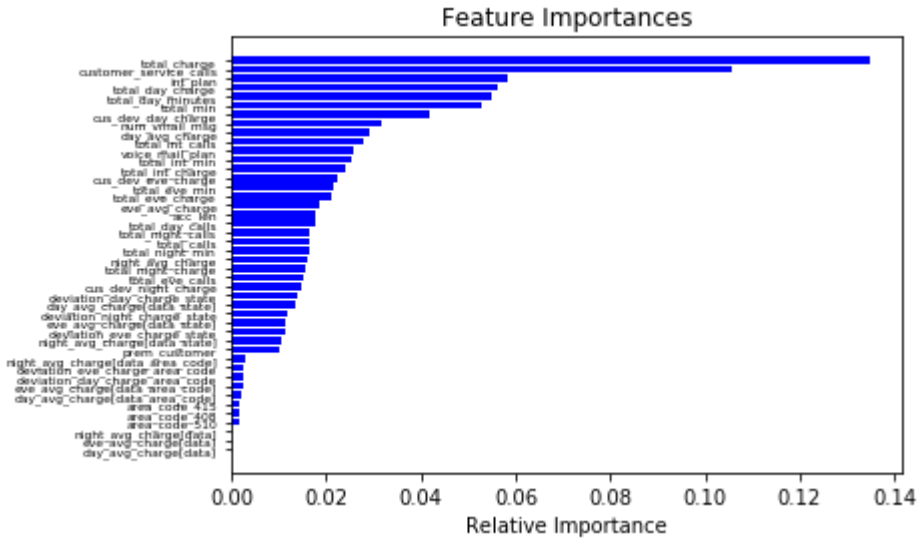
In [65]:
```
classifier_name = "RFC"
clf = RandomForestClassifier(max_depth=100, random_state=123, n_jobs=20, n_est
imators=120)

clf.fit(train_x, train_y)
pred = clf.predict(test_x)
score = clf.score(test_x, test_y)
report_model(clf, train_x, test_x, train_y,test_y)
```

```
the recall for this model is : 0.8571428571428571
TP 114
TN 867
FP 0
FN 19
```

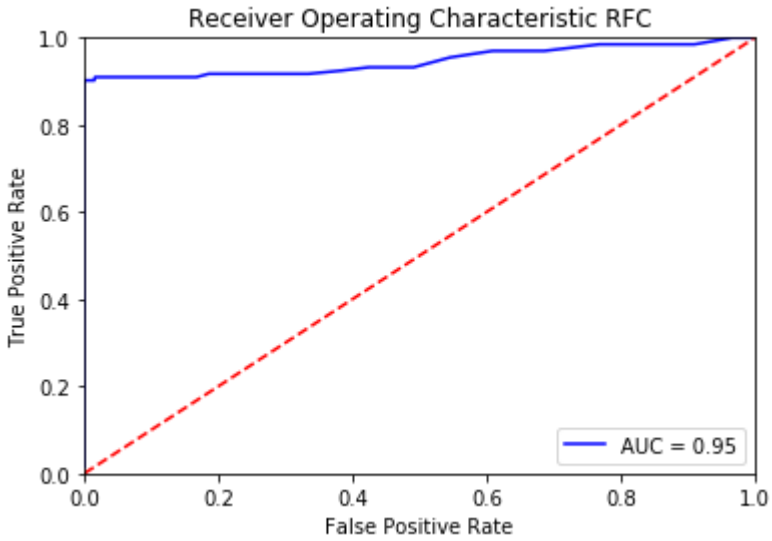



```
----------Classification Report-------------------------------------
             precision    recall  f1-score   support

      False       0.98      1.00      0.99       867
       True       1.00      0.86      0.92       133

avg / total       0.98      0.98      0.98      1000
```

In [67]:
```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
features = train_x.columns

importances = clf.feature_importances_
indices = np.argsort(importances)
plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices], fontsize = 6)
plt.xlabel('Relative Importance')
plt.figure(figsize=(20,40))
plt.savefig(classifier_name+ "_feature_imp.png")
plt.show()


# calculate the fpr and tpr for all thresholds of the classification
probs = clf.predict_proba(test_x)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(test_y, preds)
roc_auc = auc(fpr, tpr)

plt.title('Receiver Operating Characteristic '+classifier_name)
plt.plot(fpr, tpr, 'b', label = 'AUC = %0.2f' % roc_auc)
plt.legend(loc = 'lower right')
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0, 1])
plt.ylim([0, 1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.savefig(classifier_name+ "_roc_auc.png")
plt.show()

print ("Score: ", round(score,3),"\nAccuracy score: ", round(accuracy_score(te
st_y, pred), 3),"\nF1 score: ", round(f1_score(test_y, pred), 3), "\n- - - - -
 ", "\n")
```




```
<Figure size 1440x2880 with 0 Axes>
```




```
Score:  0.981
Accuracy score:  0.981
F1 score:  0.923
- - - - -
```