Author: Ricardo Huarte Salazar

About Data Set:

You will find 2 .csv files attached to this task. 1 of the files consist of courier’s lifetime dependent features and other consist courier’s weekly variant features. Features are renamed for confidentiality purposes and data dictionary will NOT be provided. However, in 2 different .csv files, same courier ID represents same courier.

In [328]:

```
 as pd
as np
tlib.pyplot as plt
n as sns
gno as msno
ute import KNN
model_selection import train_test_split
ensemble import RandomForestClassifier
ats import uniform
linear_model import LogisticRegression
metrics import confusion_matrix, classification_report
model_selection import cross_val_score, KFold, StratifiedKFold
metrics import confusion_matrix, classification_report, auc, roc_curve, precision_recall_cur
model_selection import GridSearchCV, RandomizedSearchCV

metrics import roc_curve, precision_recall_curve, auc, make_scorer, recall_score, accuracy_s

```

In [2]:

```
%matplotlib inline
```

In [3]:

```
lifetime = pd.read_csv(filepath_or_buffer='Courier_lifetime_data.csv')
```

In [4]:

```
weekly = pd.read_csv(filepath_or_buffer='Courier_weekly_data.csv')
```

In [5]:

```
lifetime.dtypes
```

Out[5]:

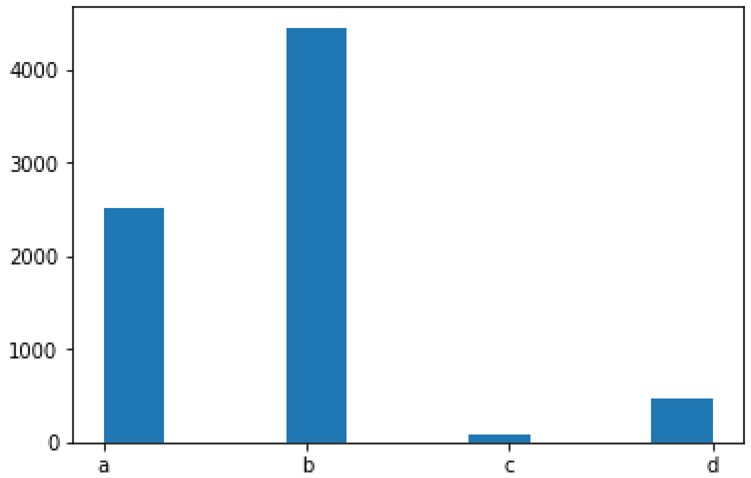
```
courier        int64
feature_1     object
feature_2    float64
dtype: object
```

In [6]:

```
1 plt.hist(lifetime['feature_1'])
```

Out[6]:

```
(array([2516.,    0.,    0., 4456.,    0.,    0.,   85.,    0.,    0.,
         467.]),
 array([0. , 0.3, 0.6, 0.9, 1.2, 1.5, 1.8, 2.1, 2.4, 2.7, 3. ]),
 <a list of 10 Patch objects>)
```

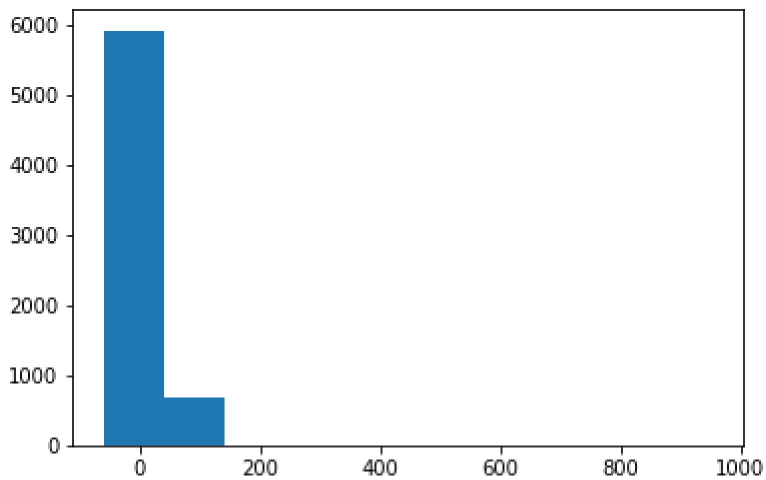


In [7]:

```
1 plt.hist(lifetime['feature_2'][~np.isnan(lifetime['feature_2'])])
```

Out[7]:

```
(array([5.915e+03, 6.700e+02, 0.000e+00, 0.000e+00, 0.000e+00, 0.000e+00,
        0.000e+00, 0.000e+00, 0.000e+00, 3.000e+00]),
 array([-61. ,  40.5, 142. , 243.5, 345. , 446.5, 548. , 649.5, 751. ,
        852.5, 954. ]),
 <a list of 10 Patch objects>)
```



In [367]:

```
1 lifetime[(lifetime.feature_2.isnull())].head()
```

Out[367]:

	courier	feature_1	feature_2
2	225	c	NaN
4	242	c	NaN
5	350	a	NaN
6	645	a	NaN
7	1210	a	NaN

In [9]:

```
1 lifetime.describe()
```

Out[9]:

	courier	feature_2
count	7524.000000	6588.000000
mean	518864.440324	26.373862
std	286880.574472	22.703621
min	208.000000	-61.000000
25%	275875.750000	20.000000
50%	529366.500000	25.000000
75%	803120.500000	32.000000
max	964240.000000	954.000000

In [11]:

```
1 lifetime_encoded= pd.get_dummies(lifetime, columns=['feature_1'])
```

In [368]:

```
1 lifetime_encoded.head()
```

Out[368]:

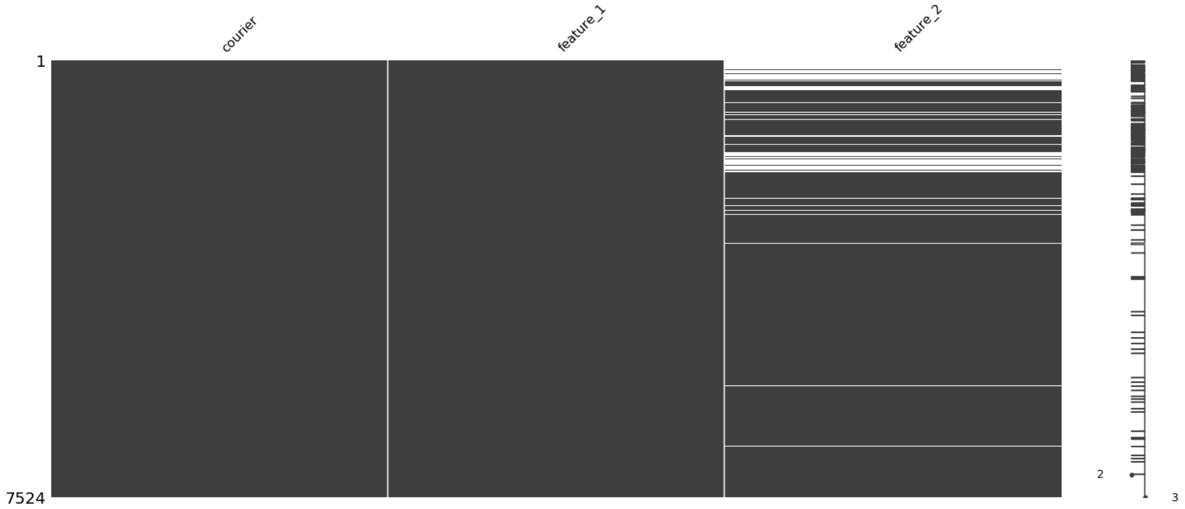
	feature_2	feature_1_a	feature_1_b	feature_1_c	feature_1_d
courier					
208	25.0	1	0	0	0
218	0.0	0	0	1	0
225	NaN	0	0	1	0
231	0.0	0	0	1	0
242	NaN	0	0	1	0

In [13]:

```
1 msno.matrix(lifetime)
```

Out[13]:

<matplotlib.axes._subplots.AxesSubplot at 0x1dfa1b5fa90>

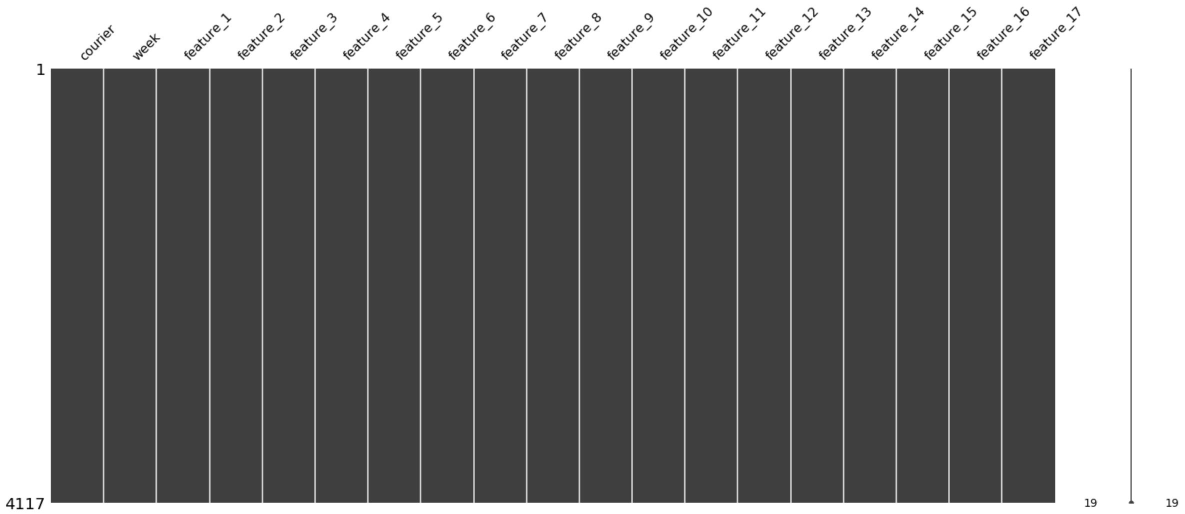


In [18]:

```
1 msno.matrix(weekly)
```

Out[18]:

<matplotlib.axes._subplots.AxesSubplot at 0x1dfa1c3f3c8>



In [19]:

```
1 weekly['courier'].drop_duplicates().count()
```

Out[19]:

759

In [20]:

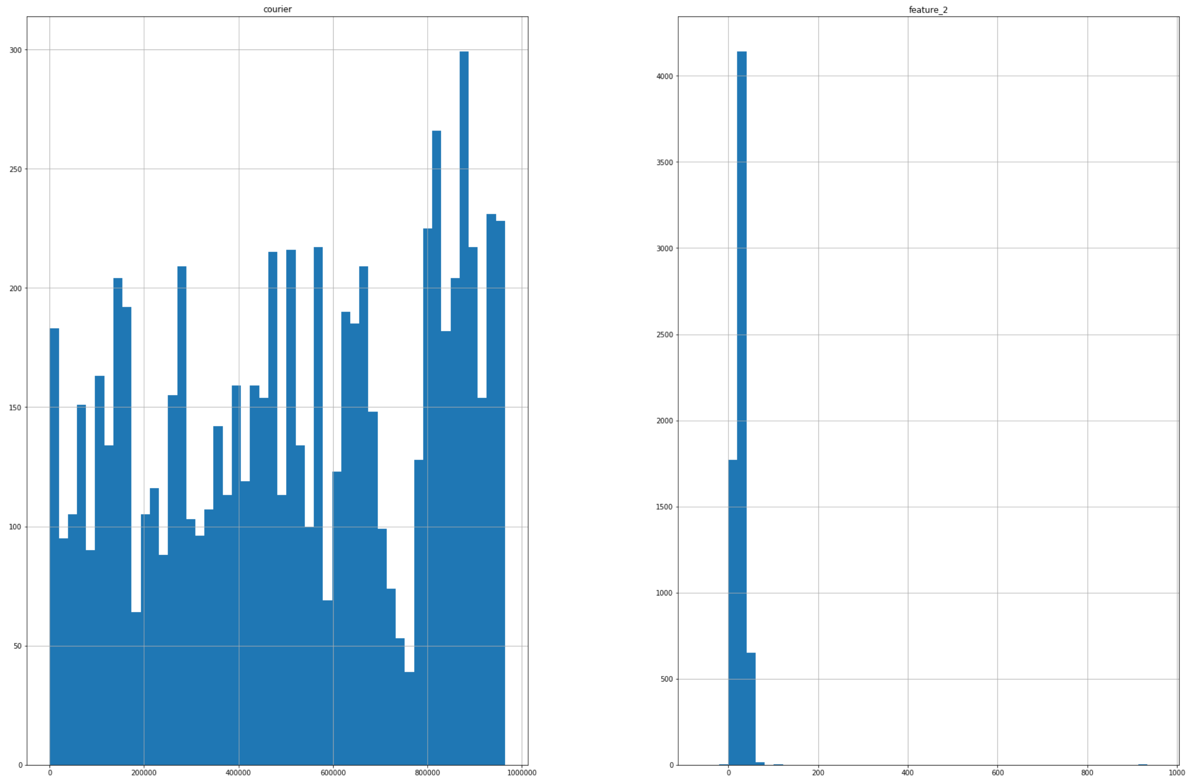
```
1 weekly[(weekly.week==11) | (weekly.week==10) | (weekly.week==9)]['courier'].drop_dupli
```

Out[20]:

387

In [21]:

```
lifetime.hist(bins=50, figsize=(30,20));
```



Task 1: Exploratory Analysis and Data Munging

In this task, you are being expected to clean data, treat missing values, find out related features and finally label the data. Every courier did not work every week. Thus, some of courier-week combinations' data are not provided. First, come up with a way to treat these missing values. Removing missing values are not suggested since provided data set is small and it will affect your predictive model's evaluation metric. Create a report / dashboard and correlation matrix, in addition to results of your univariate and bivariate analysis and explain your findings. Finally, label your data. If a specific courier's week 9, 10 and 11 data is not provided, we label this courier as "1" otherwise "0". After labeling, remove week 8(Yes including 8!), 9, 10 and 11 data to avoid bias in your next task. In addition, distribution of feature_3 is a hint how the data is generated.

In [23]:

```
1 weekly.describe().T
```

Out[23]:

	count	mean	std	min	25%	50%	
courier	4117.0	366530.934418	128603.611959	3767.000000	280239.000000	406936.000000	4
week	4117.0	4.910857	3.364852	0.000000	2.000000	5.000000	
feature_1	4117.0	-3.702453	17.407331	-138.000000	-12.000000	-2.000000	
feature_2	4117.0	44.232208	24.007116	1.000000	26.000000	41.000000	
feature_3	4117.0	55.691037	31.666550	1.000000	31.000000	51.000000	
feature_4	4117.0	0.068610	0.068999	0.000000	0.018500	0.054100	
feature_5	4117.0	0.931390	0.068999	0.000000	0.901200	0.945900	
feature_6	4117.0	104.331502	8.473348	92.857100	100.000000	100.465100	
feature_7	4117.0	0.059339	0.064646	0.000000	0.000000	0.043500	
feature_8	4117.0	3975.807328	1237.055134	1136.750000	2750.977800	4099.425000	
feature_9	4117.0	0.767527	0.136458	0.000000	0.693700	0.785700	
feature_10	4117.0	9.619359	1.827863	2.575000	8.424751	9.497961	
feature_11	4117.0	20.266942	12.460020	0.000000	11.000000	19.000000	
feature_12	4117.0	20.000994	3.205479	5.416667	18.168824	19.648810	
feature_13	4117.0	5.211435	0.961980	3.270000	4.570099	5.072500	
feature_14	4117.0	0.782381	0.164578	0.000000	0.739100	0.822200	
feature_15	4117.0	68.655642	18.828885	2.957809	57.839947	71.653595	
feature_16	4117.0	2.255526	1.542969	1.000000	1.000000	2.000000	
feature_17	4117.0	12.789410	11.691080	1.000000	5.000000	10.000000	

In [369]:

```
1 weekly.head()
```

Out[369]:

	courier	week	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	fe
0	3767	2	6	34	38	0.0789	0.9211	140.4737	0.1316	21
1	3767	4	-1	42	37	0.0000	1.0000	135.5946	0.0811	20
2	3767	5	24	41	43	0.0233	0.9767	131.0930	0.0233	20
3	3767	6	-22	65	66	0.0606	0.9394	120.1515	0.0000	21
4	6282	2	9	33	27	0.0741	0.9259	100.0000	0.0370	40

In [ ]:

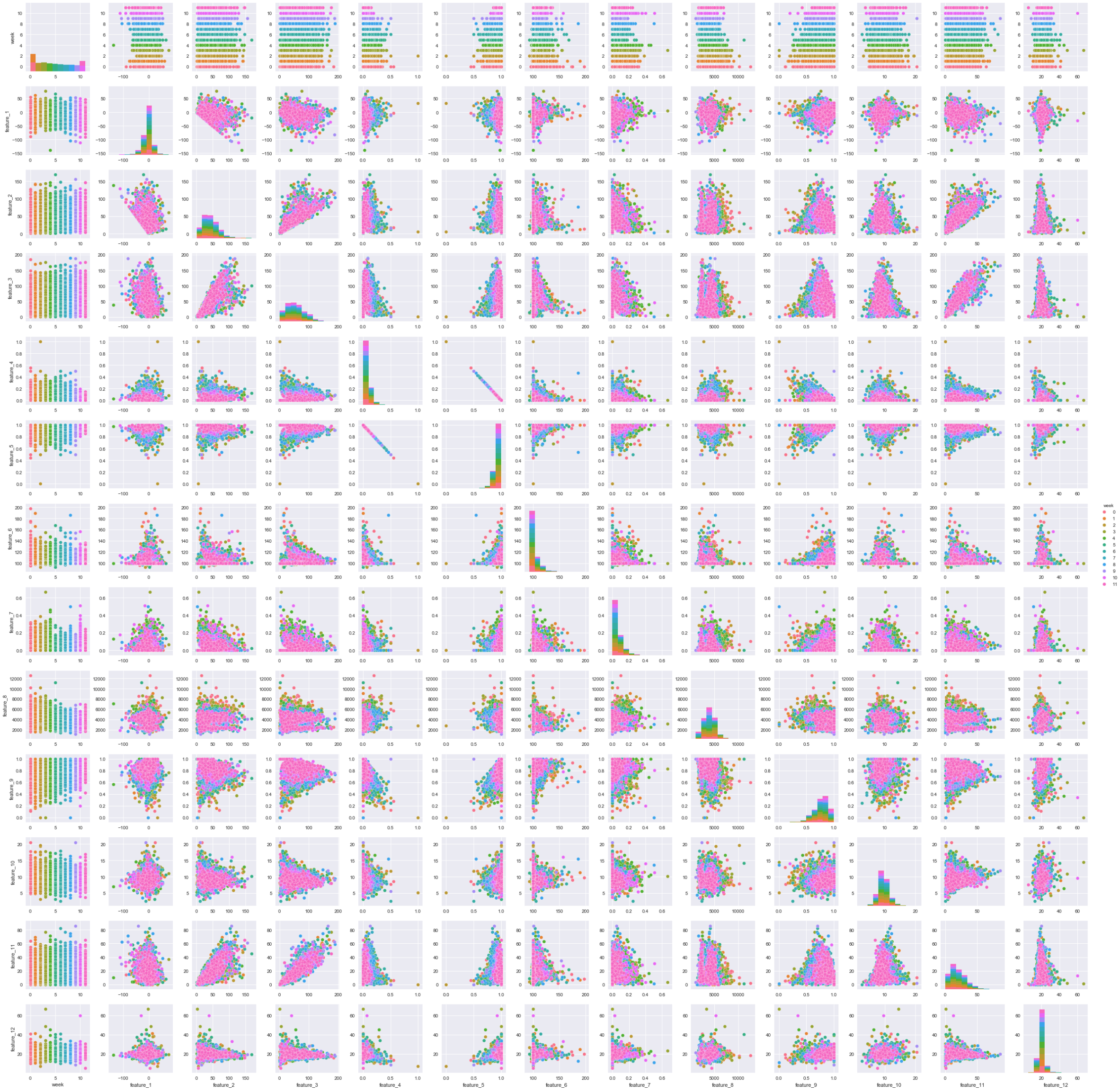
```
1 
```

In [25]:

```
1 sns.pairplot(weekly[['week','feature_1','feature_2','feature_3','feature_4','feature_5
```

Out[25]:

<seaborn.axisgrid.PairGrid at 0x1dfa4389a58>



In [26]:

```
1 analysis=weekly[['week','feature_1','feature_2','feature_3','feature_4','feature_5','f
2 corr_mt=analysis.corr()
```

In [27]:

```
corr_mt
```

Out[27]:

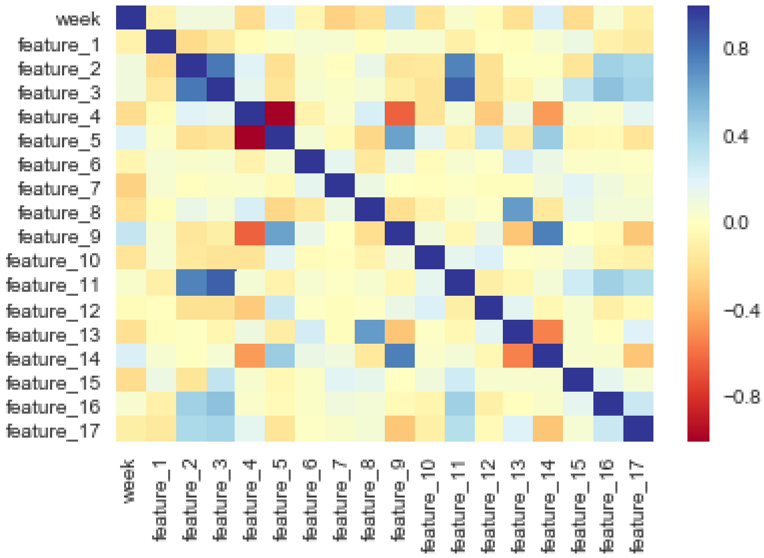
	week	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_
week	1.000000	-0.084761	0.098734	0.095960	-0.206059	0.206057	-0.059651	-0.25370
feature_1	-0.084761	1.000000	-0.212915	-0.132828	-0.029416	0.029420	0.063833	0.05642
feature_2	0.098734	-0.212915	1.000000	0.788146	0.188848	-0.188848	0.039640	-0.00313
feature_3	0.095960	-0.132828	0.788146	1.000000	0.162444	-0.162446	0.062238	0.02747
feature_4	-0.206059	-0.029416	0.188848	0.162444	1.000000	-1.000000	-0.073383	0.03801
feature_5	0.206057	0.029420	-0.188848	-0.162446	-1.000000	1.000000	0.073384	-0.03801
feature_6	-0.059651	0.063833	0.039640	0.062238	-0.073383	0.073384	1.000000	0.16207
feature_7	-0.253702	0.056426	-0.003138	0.027473	0.038013	-0.038012	0.162079	1.00000
feature_8	-0.192313	-0.008650	0.133437	0.068195	0.227298	-0.227299	-0.144915	0.11747
feature_9	0.302481	0.057826	-0.152309	-0.103423	-0.636237	0.636239	0.148237	0.00186
feature_10	-0.165839	0.062050	-0.146601	-0.173151	-0.175879	0.175880	-0.016608	-0.00251
feature_11	0.042760	-0.100593	0.749931	0.863150	0.073725	-0.073725	0.061017	0.01958
feature_12	-0.028613	-0.004104	-0.183353	-0.179758	-0.285753	0.285752	0.019124	-0.01385
feature_13	-0.189233	-0.013375	0.012738	-0.060513	0.115765	-0.115766	0.242932	-0.01201
feature_14	0.223040	0.062439	0.010894	0.063962	-0.454770	0.454773	0.129238	0.09668
feature_15	-0.205423	0.124982	-0.162611	0.316081	0.047795	-0.047795	0.030601	0.18724
feature_16	0.055816	-0.091912	0.430875	0.505898	0.032682	-0.032686	0.031249	0.09779
feature_17	-0.107284	-0.139890	0.396970	0.421079	0.171434	-0.171435	0.020535	0.04656

In [28]:

```
1 sns.heatmap(corr_mt, vmax=1., square=False,cmap="RdYlBu")
```

Out[28]:

<matplotlib.axes._subplots.AxesSubplot at 0x1dfa4bfa208>



In [ ]:

```
1
```

In [29]:

```
1 weekly.head()
```

Out[29]:

	courier	week	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	fe
0	3767	2	6	34	38	0.0789	0.9211	140.4737	0.1316	21
1	3767	4	-1	42	37	0.0000	1.0000	135.5946	0.0811	20
2	3767	5	24	41	43	0.0233	0.9767	131.0930	0.0233	20
3	3767	6	-22	65	66	0.0606	0.9394	120.1515	0.0000	21
4	6282	2	9	33	27	0.0741	0.9259	100.0000	0.0370	40

In [ ]:

```
1
```

In [30]:

```
1 week=weekly.copy()
```

In [31]:

```
def week_label(row):
    courier_set=weekly[(weekly.courier==row['courier']) & ((weekly.week==9) | (weekly.w
    if courier_set['courier'].count() == 0:
        label=1
    else:
        label=0
    return label
```

In [32]:

```
week['label']=week.apply(week_label, axis=1)
```

In [370]:

```
week.head()
```

Out[370]:

	courier	week	feature_1	feature_2	feature_3	feature_4	feature_6	feature_7	feature_8	fe
0	3767	2	6	34	38	0.0789	140.4737	0.1316	2162.4737	
1	3767	4	-1	42	37	0.0000	135.5946	0.0811	2097.4054	
2	3767	5	24	41	43	0.0233	131.0930	0.0233	2043.8837	
3	3767	6	-22	65	66	0.0606	120.1515	0.0000	2124.2727	
4	6282	2	9	33	27	0.0741	100.0000	0.0370	4075.7407	

In [34]:

```
corr_mt=week.corr()
corr_mt
```

Out[34]:

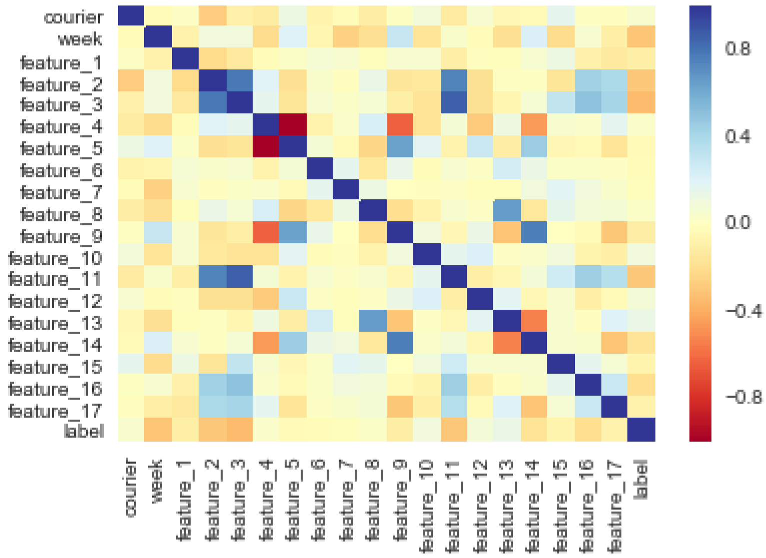
	courier	week	feature_1	feature_2	feature_3	feature_4	feature_5	feature_
courier	1.000000	-0.024238	0.016642	-0.276750	-0.093481	-0.119213	0.119214	-0.07453
week	-0.024238	1.000000	-0.084761	0.098734	0.095960	-0.206059	0.206057	-0.05965
feature_1	0.016642	-0.084761	1.000000	-0.212915	-0.132828	-0.029416	0.029420	0.06383
feature_2	-0.276750	0.098734	-0.212915	1.000000	0.788146	0.188848	-0.188848	0.03964
feature_3	-0.093481	0.095960	-0.132828	0.788146	1.000000	0.162444	-0.162446	0.06223
feature_4	-0.119213	-0.206059	-0.029416	0.188848	0.162444	1.000000	-1.000000	-0.07338
feature_5	0.119214	0.206057	0.029420	-0.188848	-0.162446	-1.000000	1.000000	0.07338
feature_6	-0.074533	-0.059651	0.063833	0.039640	0.062238	-0.073383	0.073384	1.00000
feature_7	-0.016657	-0.253702	0.056426	-0.003138	0.027473	0.038013	-0.038012	0.16207
feature_8	-0.115310	-0.192313	-0.008650	0.133437	0.068195	0.227298	-0.227299	-0.14491
feature_9	0.015373	0.302481	0.057826	-0.152309	-0.103423	-0.636237	0.636239	0.14823
feature_10	0.086274	-0.165839	0.062050	-0.146601	-0.173151	-0.175879	0.175880	-0.01660
feature_11	-0.129342	0.042760	-0.100593	0.749931	0.863150	0.073725	-0.073725	0.06101
feature_12	0.058263	-0.028613	-0.004104	-0.183353	-0.179758	-0.285753	0.285752	0.01912
feature_13	-0.046374	-0.189233	-0.013375	0.012738	-0.060513	0.115765	-0.115766	0.24293
feature_14	-0.035208	0.223040	0.062439	0.010894	0.063962	-0.454770	0.454773	0.12923
feature_15	0.161679	-0.205423	0.124982	-0.162611	0.316081	0.047795	-0.047795	0.03060
feature_16	0.010081	0.055816	-0.091912	0.430875	0.505898	0.032682	-0.032686	0.03124
feature_17	-0.002531	-0.107284	-0.139890	0.396970	0.421079	0.171434	-0.171435	0.02053
label	0.051439	-0.307316	-0.101587	-0.294234	-0.349718	0.034596	-0.034597	-0.03104

In [35]:

```
sns.heatmap(corr_mt, vmax=1., square=False,cmap="RdYlBu")
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfa9c53668>
```



In [284]:

```
week.drop('feature_5', axis=1, inplace=True)
week.drop('feature_11', axis=1, inplace=True)
```

if we look at the correlation matrix we can see that feature 4 and 5 are highly correlated, hence we will get rid of the column 5, we also see that 3 and eleven are have also a high correlation coefficient

In [285]:

```
week_2=week[(week.week<8)]
```

In [287]:

```
transposed= week_2.pivot(index='courier', columns='week')
```

In [288]:

```
transposed.columns.get_level_values(0)
```

Out[288]:

```
Index(['feature_1', 'feature_1', 'feature_1', 'feature_1', 'feature_1',
       'feature_1', 'feature_1', 'feature_1', 'feature_2', 'feature_2',
       ...
       'feature_17', 'feature_17', 'label', 'label', 'label', 'label', 'labe
l',
       'label', 'label', 'label'],
      dtype='object', length=128)
```

In [289]:

```
transposed.columns = [transposed + '_' + i for transposed, i in zip(transposed.columns
```

In [ ]:

```
lifetime_encoded.set_index('courier', inplace=True)
```

In [290]:

```
transposed=transposed.merge(lifetime_encoded, how='inner', left_index=True, right_inde
```

In [291]:

```
transposed.head()
```

Out[291]:

courier	feature_1_0	feature_1_1	feature_1_2	feature_1_3	feature_1_4	feature_1_5	feature_1_6
3767	NaN	NaN	6.0	NaN	-1.0	24.0	-22.0
6282	NaN	NaN	9.0	-20.0	9.0	21.0	-12.0
10622	5.0	-12.0	NaN	NaN	NaN	NaN	NaN
13096	NaN	NaN	NaN	NaN	NaN	-10.0	10.0
14261	4.0	-16.0	2.0	3.0	7.0	-1.0	-1.0

5 rows × 133 columns

In [ ]:

```

```

In [371]:

```
transposed_mt=transposed.corr()
transposed_mt.head()
```

Out[371]:

	feature_1_0	feature_1_1	feature_1_2	feature_1_3	feature_1_4	feature_1_5	feature
feature_1_0	1.000000	-0.251950	-0.029288	-0.082137	-0.117810	0.038922	0.08
feature_1_1	-0.251950	1.000000	-0.374919	0.128965	0.211154	0.001474	-0.02
feature_1_2	-0.029288	-0.374919	1.000000	-0.267218	0.007909	0.114968	-0.00
feature_1_3	-0.082137	0.128965	-0.267218	1.000000	-0.289214	0.115630	-0.05
feature_1_4	-0.117810	0.211154	0.007909	-0.289214	1.000000	-0.347923	0.06

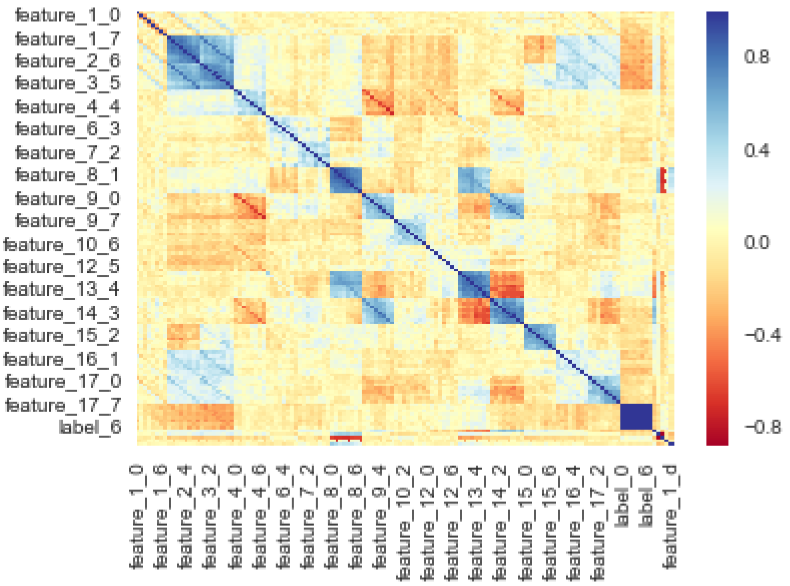
5 rows × 126 columns

In [293]:

```
sns.heatmap(transposed_mt, vmax=1., square=False,cmap="RdYlBu")
```

Out[293]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfba30d2b0>
```



In [294]:

```
def get_label(row):
    label=0
    for i in range(8):
        column='label_'+ str(i)
        if row[column]==1:
            label=1
    return label

```

In [295]:

```
transposed['y']=transposed.apply(get_label,axis=1)
```

In [296]:

```
columns_dl=['label_0','label_1','label_2','label_3','label_4','label_5','label_6','lab
```

In [297]:

```
transposed.drop(['label_0','label_1','label_2','label_3','label_4','label_5','label_6'
```

In [372]:

```
1 transposed.head()
```

Out[372]:

courier	feature_1_0	feature_1_1	feature_1_2	feature_1_3	feature_1_4	feature_1_5	feature_1_6
3767	NaN	NaN	6.0	NaN	-1.0	24.0	-22.0
6282	NaN	NaN	9.0	-20.0	9.0	21.0	-12.0
10622	5.0	-12.0	NaN	NaN	NaN	NaN	NaN
13096	NaN	NaN	NaN	NaN	NaN	-10.0	10.0
14261	4.0	-16.0	2.0	3.0	7.0	-1.0	-1.0

5 rows × 126 columns

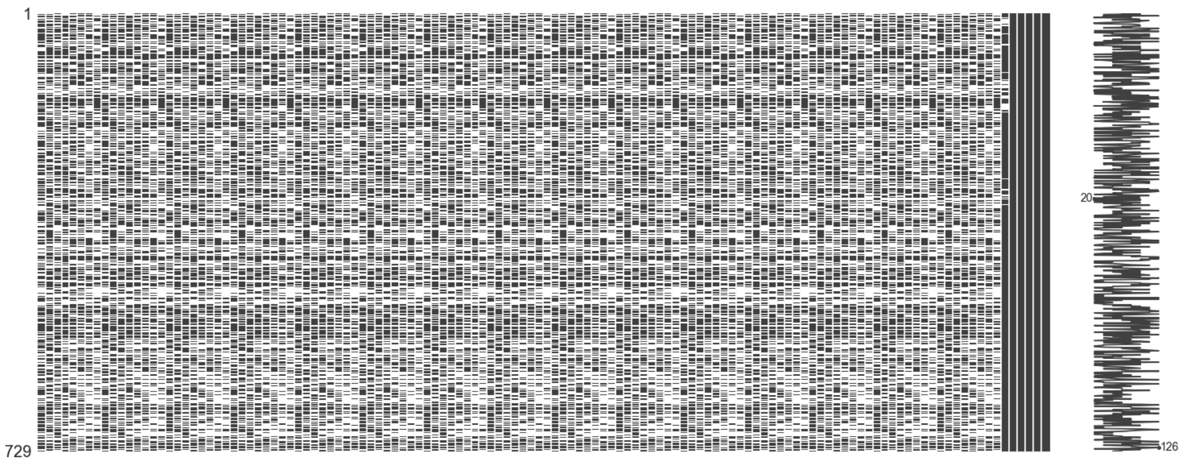
Imputing Missing values using KNN and FancyImpute

In [299]:

```
1 msno.matrix(transposed)
```

Out[299]:

<matplotlib.axes._subplots.AxesSubplot at 0x1dfba39ab38>



I chose to use KNN imputation because I feel like it gives us a better idea od the users than other imputation methods like means and modes,

I also tried imputing zero (0) to the features where we had no information but that approach gave me less predicting power than KNN Imputation

In [300]:

```
transposed_filled =  pd.DataFrame(KNN(k=2).fit_transform(transposed), columns=transpos
```

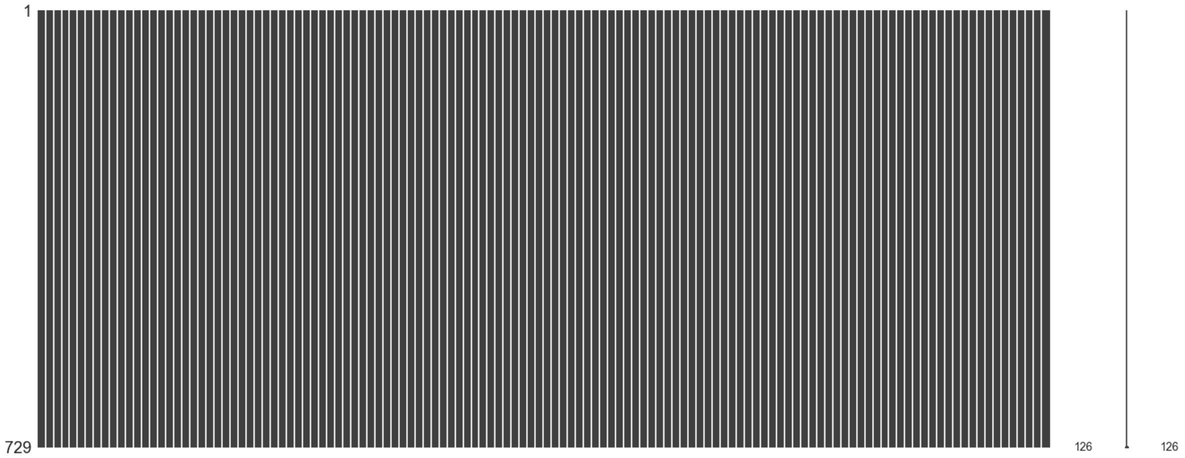
```
Imputing row 1/729 with 60 missing, elapsed time: 0.896
Imputing row 101/729 with 105 missing, elapsed time: 1.022
Imputing row 201/729 with 15 missing, elapsed time: 1.138
Imputing row 301/729 with 60 missing, elapsed time: 1.366
Imputing row 401/729 with 105 missing, elapsed time: 1.533
Imputing row 501/729 with 0 missing, elapsed time: 1.700
Imputing row 601/729 with 90 missing, elapsed time: 1.838
Imputing row 701/729 with 105 missing, elapsed time: 2.015
```

In [301]:

```
msno.matrix(transposed_filled)
```

Out[301]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1dfba46ee10>
```



Predicting Classes

Labels are almost balance at 50/50

In [302]:

```
golden_x= transposed_filled.drop(['y'], axis=1)
golden_y= transposed_filled['y']
```

In [303]:

```
x_train, x_test, y_train, y_test = train_test_split(golden_x,golden_y , test_size=0.25
```

In [304]:

```
golden_y.value_counts()
```

Out[304]:

```
0.0    365
1.0    364
Name: y, dtype: int64
```

Using Logistic Regression w Grid Search

In [362]:

```
clf = LogisticRegression()
```

In [334]:

```
logisticRegr = LogisticRegression()
logisticRegr.fit(X=x_train, y=y_train)
test_y_pred = logisticRegr.predict(x_test)
cf_mt = confusion_matrix(y_test, test_y_pred)
print('Intercept: ' + str(logisticRegr.intercept_))
print('Regression: ' + str(logisticRegr.coef_))
print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logisticl
print(classification_report(y_test, test_y_pred))
```

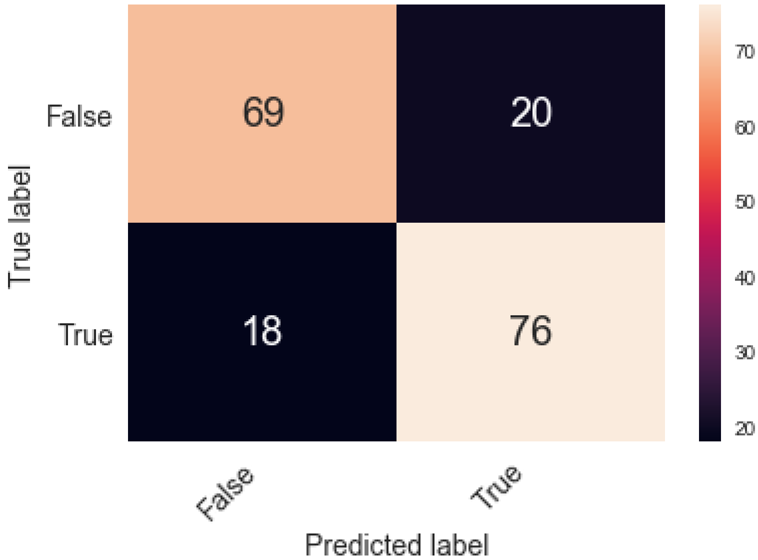
```
Intercept: [0.08250367]
Regression: [[-2.33585280e-02  2.23843424e-02  2.90318108e-03 -4.97461652e-0
2
  -4.55416559e-02 -4.00217777e-02 -5.01389168e-02 -6.51825045e-02
   2.12225610e-02 -4.08226525e-03 -4.50195507e-03 -5.02704635e-02
   4.06239407e-02 -1.04078398e-02  2.42477353e-02 -4.49304715e-02
  -2.01439034e-02 -3.31138091e-02  2.85915143e-02  2.64477053e-02
  -3.87853683e-02  6.32579392e-03 -3.50290578e-02  1.54860836e-02
   2.72099330e-02  3.31997524e-01  1.55754312e-01  2.13096596e-01
  -8.74812485e-02  8.35084541e-02  3.55258397e-01  2.95026245e-01
  -4.64646054e-02  3.30660486e-02  1.19925375e-01 -6.86550009e-03
   3.45330504e-02 -2.26436556e-02 -3.50423441e-02 -1.35494007e-01
  -1.54684973e-01 -7.45526199e-02 -4.89261942e-02 -1.75692309e-01
  -2.45132022e-01 -2.19442584e-01 -8.96292455e-02  2.85270621e-03
  -3.10961885e-04  1.02242770e-03  7.04419029e-04 -5.15864045e-04
  -5.24781319e-04 -2.21963170e-04  1.79660613e-04  6.22287164e-04
   9.93032797e-02  1.88471722e-01 -1.18169809e-01 -5.26891529e-01
  -1.12227822e-01 -2.52203622e-01 -2.40071696e-01 -4.41510958e-01
   2.06468724e-01 -1.62731958e-01  1.37459154e-01  9.23105767e-02
   1.27127652e-01  1.83976451e-01  2.35317682e-01 -2.67812355e-01
  -1.49208490e-03  1.60283144e-01  1.52228973e-01 -5.17015997e-02
   4.79080321e-02  3.96172661e-03 -1.35737920e-01 -2.09202164e-02
   4.99636533e-01 -6.79362118e-01 -1.60576250e+00  2.22487018e-01
   5.69455213e-01  4.24346880e-01  1.05381649e+00  2.60465940e-01
  -4.45071530e-02  1.42882516e-01  9.96511914e-02 -3.25139684e-01
   5.22873426e-01 -4.20723472e-02 -4.09887397e-01 -3.20157612e-01
   1.44207259e-02  4.52463544e-05 -2.03665615e-02  1.42798672e-03
   6.03142835e-03 -8.73470413e-03  1.16894701e-02 -3.53672933e-02
  -9.49074407e-03  1.64685921e-01 -9.31360653e-01  3.56551548e-01
  -4.09565816e-01 -7.27424496e-01 -5.29651816e-01 -2.31923629e-01
  -4.08153598e-02 -1.94702323e-02  5.71749753e-02 -2.44126983e-02
   7.09380087e-02 -2.32544502e-02 -4.96074572e-02 -8.47006029e-02
   9.82610436e-03  8.81166869e-02 -1.49438433e-01  2.81994134e-01
  -1.38168714e-01]]
Accuracy of logistic regression classifier on test set: 0.79
             precision    recall  f1-score   support

        0.0       0.79      0.78      0.78        89
        1.0       0.79      0.81      0.80        94

avg / total       0.79      0.79      0.79       183
```

In [335]:

```
confusion_matrix_df = pd.DataFrame(cf_mt, ('False', 'True'), ('False', 'True'))
heatmap = sns.heatmap(confusion_matrix_df, annot=True, annot_kws={"size": 20}, fmt="d"
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right',
plt.ylabel('True label', fontsize = 14)
plt.xlabel('Predicted label', fontsize = 14)
```

Out[335]:

```
Text(0.5,16,'Predicted label')
```



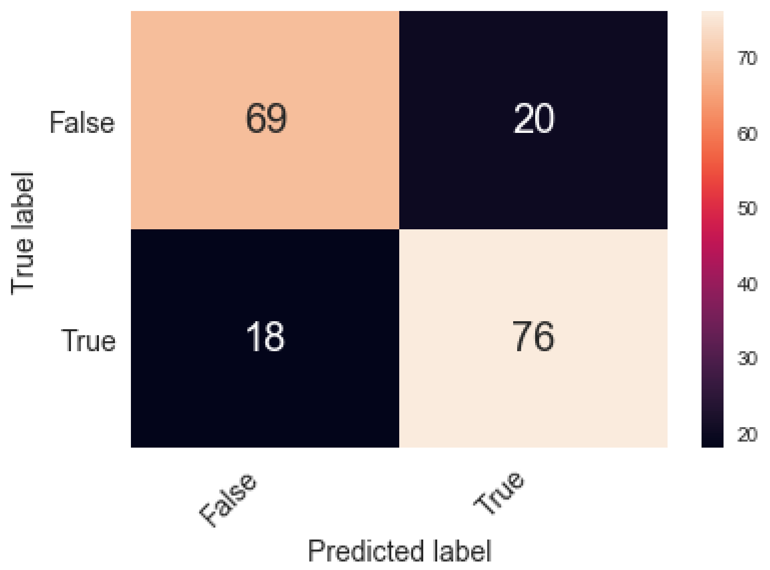
In [336]:

```

clf.fit(x_train,y_train)
print(clf.score(x_test, y_test))
seed = 7
k_fold = KFold(n_splits=10, random_state=seed)
scoring = 'accuracy'
results = results=cross_val_score(clf, x_test, y_test, cv=k_fold, n_jobs=1, scoring=sc
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'neg_log_loss'
results = results=cross_val_score(clf, x_test, y_test, cv=k_fold, n_jobs=1, scoring=sc
print("Logloss: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'roc_auc'
results = results=cross_val_score(clf, x_test, y_test, cv=k_fold, n_jobs=1, scoring=sc
print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))

clf.fit(x_train,y_train)
predicted=clf.predict(x_test)
matrix = confusion_matrix(y_test, predicted)
print(matrix)


confusion_matrix_df = pd.DataFrame(matrix, ('False', 'True'), ('False', 'True'))
heatmap = sns.heatmap(confusion_matrix_df, annot=True, annot_kws={"size": 20}, fmt="d"
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', 
plt.ylabel('True label', fontsize = 14)
plt.xlabel('Predicted label', fontsize = 14)

```

```
0.7923497267759563
Accuracy: 0.672 (0.135)
Logloss: -4.142 (1.447)
AUC: 0.729 (0.162)
[[69 20]
 [18 76]]
```

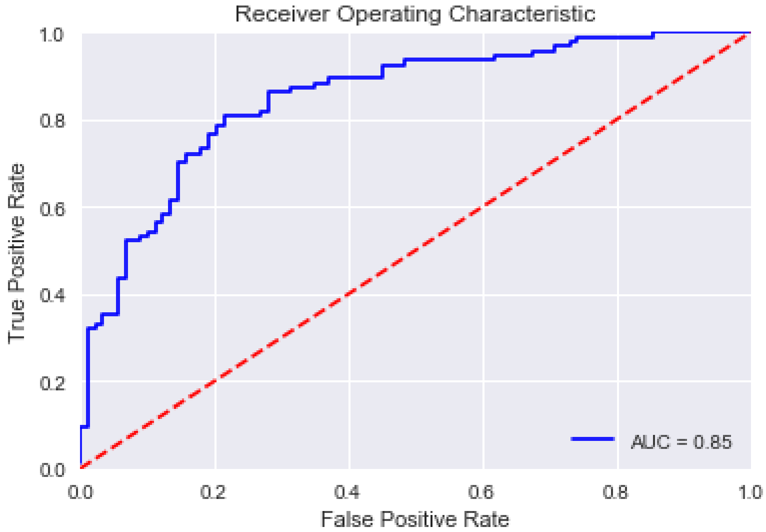
Out[336]:

```
Text(0.5,16,'Predicted label')
```



In [337]:

```
probs = clf.predict_proba(x_test)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(y_test, preds)
roc_auc = auc(fpr, tpr)

precis

plt.title('Receiver Operating Characteristic')
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.show()
```



I Used Randomize search with Logistic regression since with fewer iterations and less time is more likely to find the optimal parameters

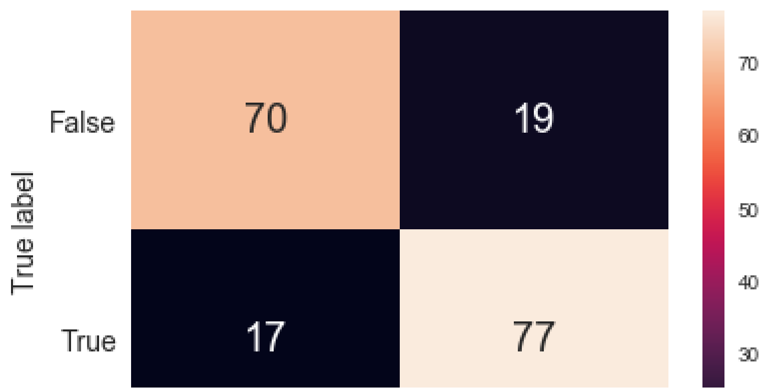
In [363]:

```
penalty = ['l2']
#C = np.logspace(0, 4, 10)
C = uniform(loc=0, scale=4)
hyperparameters = dict(C=C, penalty=penalty)

#clf_CV = GridSearchCV(clf, hyperparameters, cv=5, verbose=0)
clf_CV = RandomizedSearchCV(clf, hyperparameters, random_state=1, n_iter=10, cv=5, ver
best_model = clf_CV.fit(x_train, y_train)
clf_CV.fit(x_train,y_train)
print(clf_CV.score(x_test, y_test))
seed = 7
k_fold = KFold(n_splits=10, random_state=seed)
scoring = 'accuracy'
results = results=cross_val_score(clf_CV, x_test, y_test, cv=k_fold, n_jobs=1, scoring
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'neg_log_loss'
results = results=cross_val_score(clf_CV, x_test, y_test, cv=k_fold, n_jobs=1, scoring
print("Logloss: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'roc_auc'
results = results=cross_val_score(clf_CV, x_test, y_test, cv=k_fold, n_jobs=1, scoring
print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))

predicted=best_model.predict(x_test)
matrix = confusion_matrix(y_test, predicted)


confusion_matrix_df = pd.DataFrame(matrix, ('False', 'True'), ('False', 'True'))
heatmap = sns.heatmap(confusion_matrix_df, annot=True, annot_kws={"size": 20}, fmt="d"
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right',
plt.ylabel('True label', fontsize = 14)
plt.xlabel('Predicted label', fontsize = 14)

```

```
0.8032786885245902
Accuracy: 0.744 (0.082)
Logloss: -0.592 (0.160)
AUC: 0.811 (0.098)
```

Out[363]:

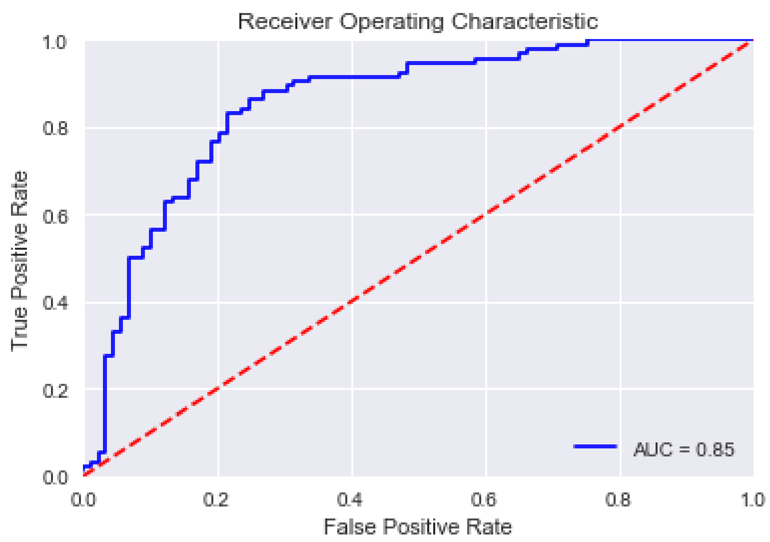
```
Text(0.5,16,'Predicted label')
```



In [366]:

```
probs = best_model.predict_proba(x_test)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(y_test, preds)
roc_auc = auc(fpr, tpr)

print(accuracy_score(y_test, predicted))

plt.title('Receiver Operating Characteristic')
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.show()
```

0.8032786885245902



I've chosen to use the ROC Curve since we have balanced classes, in this curve we can see how "powerful" our model is, and how false positives are related to true positives, the higher true positive values with a minimum rate of false positive values the better, and we can see in the case of Random Forest, the curve has a good shape and the AUC is higher

In [138]:

```

```

In [ ]:

```

```

Using Random Forest Classifier

To complement the prediction part, we try as well the RandomForest algorithm which after parameter tuning yields better results than logistic regression.

In [347]:

```
clf = RandomForestClassifier(n_jobs=-1)

param_grid = {
    'min_samples_split': [3, 5, 10],
    'n_estimators' : [100, 300],
    'max_depth': [3, 5, 15, 25],
    'max_features': [3, 5, 10, 20]
}

scorers = {
    'precision_score': make_scorer(precision_score),
    'recall_score': make_scorer(recall_score),
    'accuracy_score': make_scorer(accuracy_score)
}
```

In [348]:

```
def grid_search_wrapper(refit_score='precision_score'):
    """
    fits a GridSearchCV classifier using refit_score for optimization
    prints classifier performance metrics
    """
    skf = StratifiedKFold(n_splits=10)
    grid_search = GridSearchCV(clf, param_grid, scoring=scorers, refit=refit_score,
                               cv=skf, return_train_score=True, n_jobs=-1)
    grid_search.fit(x_train.values, y_train.values)


    y_pred = grid_search.predict(x_test.values)

    print('Best params for {}'.format(refit_score))
    print(grid_search.best_params_)

    # confusion matrix on the test data.
    print('\nConfusion matrix of Random Forest optimized for {} on the test data:'.for
    print(pd.DataFrame(confusion_matrix(y_test, y_pred),
                 columns=['pred_neg', 'pred_pos'], index=['neg', 'pos']))
    return grid_search
```

In [349]:

```
grid_search_clf = grid_search_wrapper(refit_score='precision_score')
```

```
Best params for precision_score
{'max_depth': 25, 'max_features': 3, 'min_samples_split': 10, 'n_estimator
s': 300}

Confusion matrix of Random Forest optimized for precision_score on the test
data:
     pred_neg  pred_pos
neg        76        13
pos        11        83
```

In [ ]:

```

```

In [ ]:

```
 
```

In [ ]:

```
 
```

In [ ]:

```
 
```

In [350]:

```
randomForest = RandomForestClassifier( max_depth=25, max_features=3, min_samples_split
randomForest.fit(x_train, y_train)
print('Accuracy of random forest classifier on test set: {:.2f}'.format(randomForest.s
```

Accuracy of random forest classifier on test set: 0.86

In [351]:

```
test_y_pred = randomForest.predict(x_test)
cf_mt = confusion_matrix(y_test, test_y_pred)
cf_mt
```

Out[351]:

```
array([[75, 14],
       [11, 83]], dtype=int64)
```

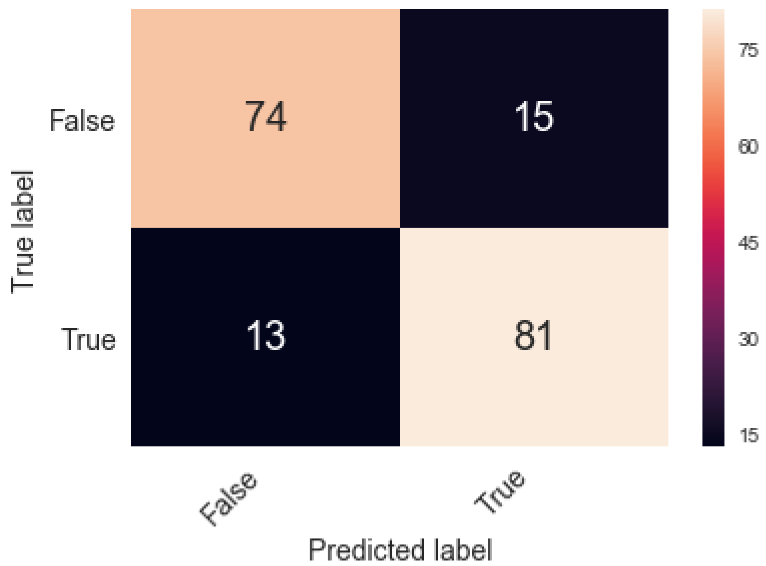
In [357]:

```

print(randomForest.score(x_test, y_test))
seed = 7
k_fold = KFold(n_splits=10, random_state=seed)
scoring = 'accuracy'
results = results=cross_val_score(randomForest, x_test, y_test, cv=k_fold, n_jobs=1, s
print("Accuracy: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'neg_log_loss'
results = results=cross_val_score(randomForest, x_test, y_test, cv=k_fold, n_jobs=1, s
print("Logloss: %.3f (%.3f)" % (results.mean(), results.std()))
scoring = 'roc_auc'
results = results=cross_val_score(randomForest, x_test, y_test, cv=k_fold, n_jobs=1, s
print("AUC: %.3f (%.3f)" % (results.mean(), results.std()))

randomForest.fit(x_train,y_train)
predicted=randomForest.predict(x_test)
matrix = confusion_matrix(y_test, predicted)
print(matrix)


confusion_matrix_df = pd.DataFrame(matrix, ('False', 'True'), ('False', 'True'))
heatmap = sns.heatmap(confusion_matrix_df, annot=True, annot_kws={"size": 20}, fmt="d"
heatmap.yaxis.set_ticklabels(heatmap.yaxis.get_ticklabels(), rotation=0, ha='right', f
heatmap.xaxis.set_ticklabels(heatmap.xaxis.get_ticklabels(), rotation=45, ha='right', 
plt.ylabel('True label', fontsize = 14)
plt.xlabel('Predicted label', fontsize = 14)

```

```
0.8688524590163934
Accuracy: 0.820 (0.063)
Logloss: -0.501 (0.041)
AUC: 0.907 (0.053)
[[74 15]
 [13 81]]
```

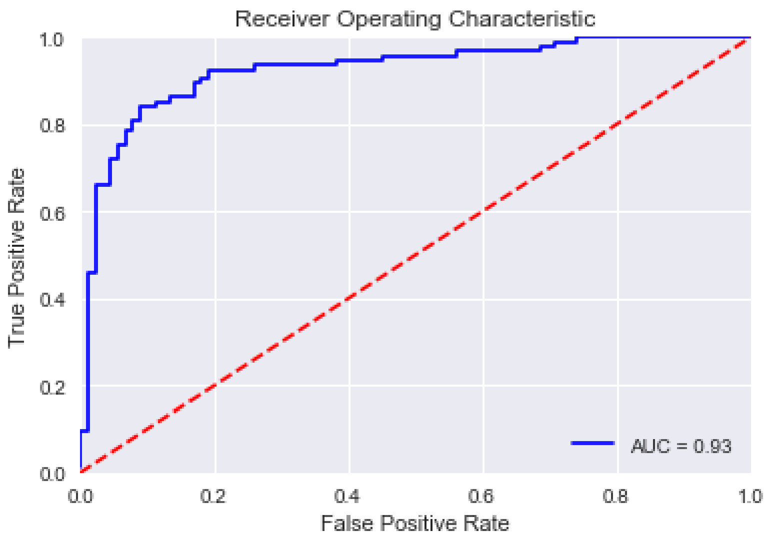
Out[357]:

```
Text(0.5,16,'Predicted label')
```



In [358]:

```
probs = randomForest.predict_proba(x_test)
preds = probs[:,1]
fpr, tpr, threshold = roc_curve(y_test, preds)
roc_auc = auc(fpr, tpr)


plt.title('Receiver Operating Characteristic')
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.show()
```



In [ ]:

```

```