Kensho-data-challenge-2

November 15, 2017

1 Kensho-data-challenge-2

1.1 Part 0

1.2 Modules

In [421]:
```
# data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

# model
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from keras.models import Sequential
from keras.layers import Dense,Dropout, Activation
from keras.layers.normalization import BatchNormalization
from keras.callbacks import EarlyStopping
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans, MiniBatchKMeans

# other
from sklearn.model_selection import train_test_split, StratifiedKFold, KF
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
import itertools as itr
from sklearn.preprocessing import LabelBinarizer
from keras.utils import to_categorical
from sklearn.metrics import log_loss
from sklearn.preprocessing import LabelEncoder


# utils
from utils import plot_confusion_matrix
```

1.3 Target

In this challenge we need to build a multiclass classifier to indentify crimes. I will show necessary data exploration and experiment with multiple models and feature engineering

1.4 Data

```
In [422]: train = pd.read_csv("../input/data_science_challenge/NYPD_7_Major_Felony_
          test = pd.read_csv("../input/data_science_challenge/NYPD_7_Major_Felony_I
```

basic check

```
In [423]: train.shape
```

```
Out[423]: (213689, 19)
```

```
In [424]: test.shape
```

```
Out[424]: (100599, 19)
```

```
In [425]: train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 213689 entries, 0 to 213688
Data columns (total 19 columns):
Identifier               213689 non-null object
Occurrence Datetime      213689 non-null object
Day of Week              213689 non-null object
Occurrence Month         213689 non-null object
Occurrence Day           213689 non-null int64
Occurrence Year          213689 non-null int64
Occurrence Hour          213689 non-null int64
CompStat Month           213689 non-null int64
CompStat Day             213689 non-null int64
CompStat Year            213689 non-null int64
Offense                  213689 non-null object
Sector                   213689 non-null object
Precinct                 213689 non-null int64
Borough                  213689 non-null object
Jurisdiction             213689 non-null object
XCoordinate              213689 non-null int64
YCoordinate              213689 non-null int64
Location 1               213689 non-null object
Occurrence Date          213689 non-null object
dtypes: int64(9), object(10)
memory usage: 31.0+ MB
```

```
In [426]: train.head(2)
```

```
Out[426]:   Identifier     Occurrence Datetime Day of Week Occurrence Month  \
          0   4eaf2b62  02/13/2013 12:00:00 AM   Wednesday              Feb
          1   cacec67c  02/13/2013 12:00:00 AM   Wednesday              Feb

             Occurrence Day  Occurrence Year  Occurrence Hour  CompStat Month  \
          0              13             2013                0               2
          1              13             2013                0               5

             CompStat Day  CompStat Year        Offense Sector  Precinct    Borough
          0            14           2013  GRAND LARCENY      H        13  MANHATTAN
          1            20           2013  GRAND LARCENY      I        52      BRONX

                 Jurisdiction  XCoordinate  YCoordinate  \
          0  N.Y. POLICE DEPT       985716       209911
          1  N.Y. POLICE DEPT      1016552       260706

                                      Location 1 Occurrence Date
          0  (40.7428419120001, -73.9947109889999)      2013-02-13
          1            (40.88220104, -73.88318653)      2013-02-13

In [427]: test.head(2)

Out[427]:   Identifier     Occurrence Datetime Day of Week Occurrence Month  \
          0   aae098f0  10/02/2015 12:11:00 PM      Monday              Oct
          1   d71bac4b  09/06/2015 02:00:00 AM   Wednesday              Sep

             Occurrence Day  Occurrence Year  Occurrence Hour  CompStat Month  \
          0               2             2015               12              11
          1               6             2015                2              10

             CompStat Day  CompStat Year          Offense Sector  Precinct    Borough
          0            23           2015    GRAND LARCENY      B        25  MANHATTAN
          1             2           2015  FELONY ASSAULT      G        90   BROOKLYN

                    Jurisdiction  XCoordinate  YCoordinate  \
          0     N.Y. POLICE DEPT      1001575       232339
          1  N.Y. HOUSING POLICE       999983       195658

                                      Location 1 Occurrence Date
          0  (40.8043840460001, -73.9374216689999)      2015-10-02
          1          (40.703707008, -73.943257966)      2015-09-06
```

Find: 1. The basic check shows that we have two large data set, the train have two times number of rows than the test. 2. The target value is "Offense" column, and we need to binarize it so that we could build a binary model. 3. There are both numerical and categorical features in the data, we need to do feature encoding and engineering for them 4. Some features seems to be redundant, containing the same information, we could remove them 5. Some feature extraction has been done on the Datetime, we could remove this column 6. Location 1 and X, Y coordinate are the same information

1.5 Preprocessing

```
In [428]: id_test = test["Identifier"]
```

```
In [429]: col_drop = ['Identifier', 'Occurrence Datetime','Occurrence Month','Occu
```

To be safe, we did not remove the "Occurance Data" column

```
In [430]: train.drop(col_drop, axis=1, inplace=True)
          test.drop(col_drop, axis=1, inplace=True)
```

```
In [431]: X_train = train.drop("Offense", axis=1)
          y_train = train["Offense"]
```

```
In [432]: X_train_copy = X_train.copy()
          y_train_copy = y_train.copy()
```

```
In [433]: X_test = test.drop("Offense", axis=1)
          y_test = test["Offense"]
```

1.6 Part 1

We first need to build a dummy classifier that always predict "GRAND LARCENY". I will work on the test data directly

If we predict all the event to be "GRAND LARCENY"

```
In [434]: y_test.value_counts()["GRAND LARCENY"]/y_test.shape[0]
```

```
Out[434]: 0.40560045328482391
```

Thus, we could get a accuracy of ***0.4056***

This is due to the fact that "GRAND LARCENY" account for 40% of the test events, implying that accuracy may not be a fair metric to measure the performance

1.7 Part 2

Then I will work on training a more detailed model

1.8 EDA

After preprocessing the data, we could start to look at the internal structure of the data

Currently we have the columns of

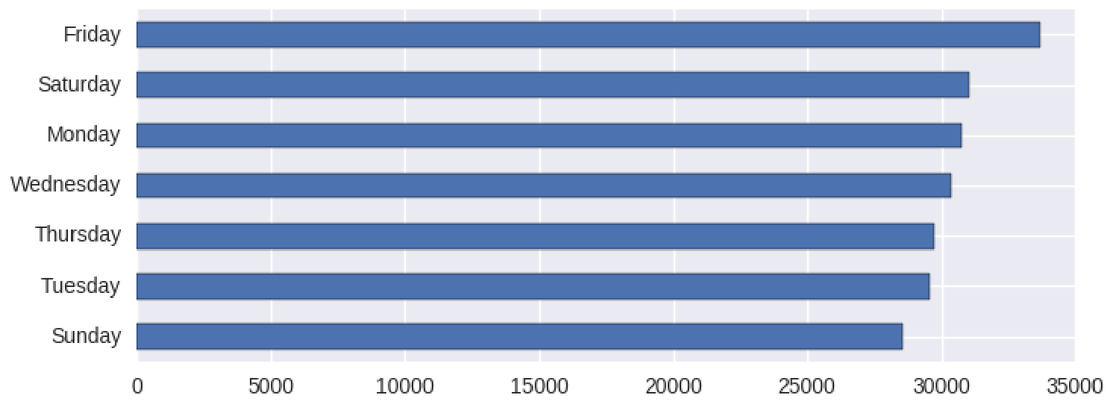
```
In [435]: X_train.columns
```

```
Out[435]: Index(['Day of Week', 'Occurrence Year', 'Occurrence Hour', 'CompStat Mor
                 'CompStat Day', 'CompStat Year', 'Sector', 'Precinct', 'Borough',
                 'Jurisdiction', 'XCoordinate', 'YCoordinate', 'Occurrence Date'],
                dtype='object')
```

I will start from ***univariate exploration*** to check the distribution of each feature

Day of Week

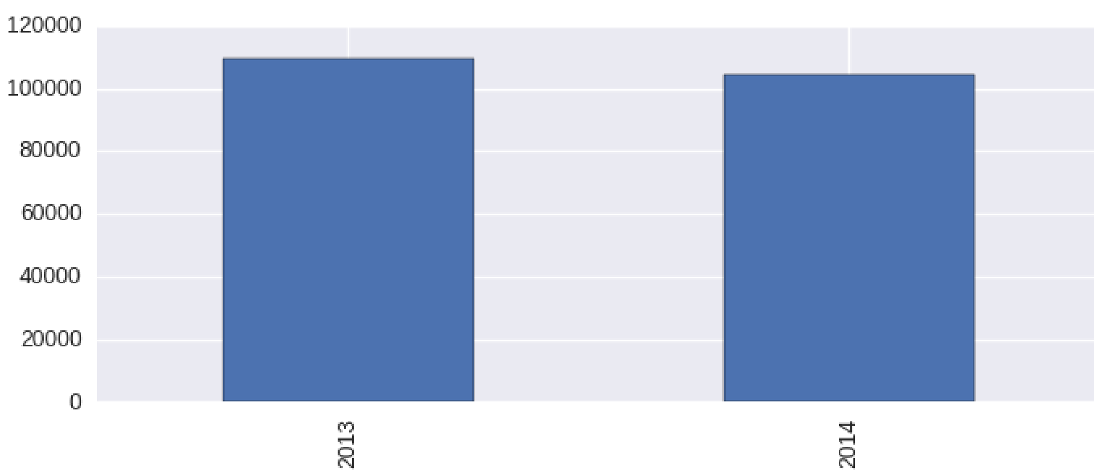
```
In [436]: X_train["Day of Week"].value_counts().sort_values().plot.barh(figsize=(8,
```



Generally, Friday has higher amount of crimes than other weekdays

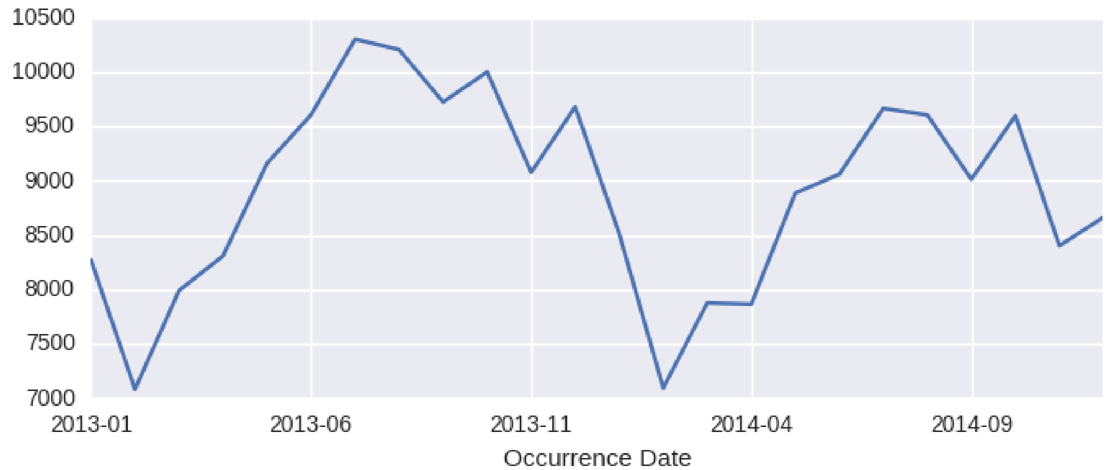
Occurrence Year

```
In [437]: X_train["Occurrence Year"].value_counts().plot.bar(figsize=(8,3));
```



From 2013 to 2014, the crimes amount decrease, we could zoom into each month data and see the trend

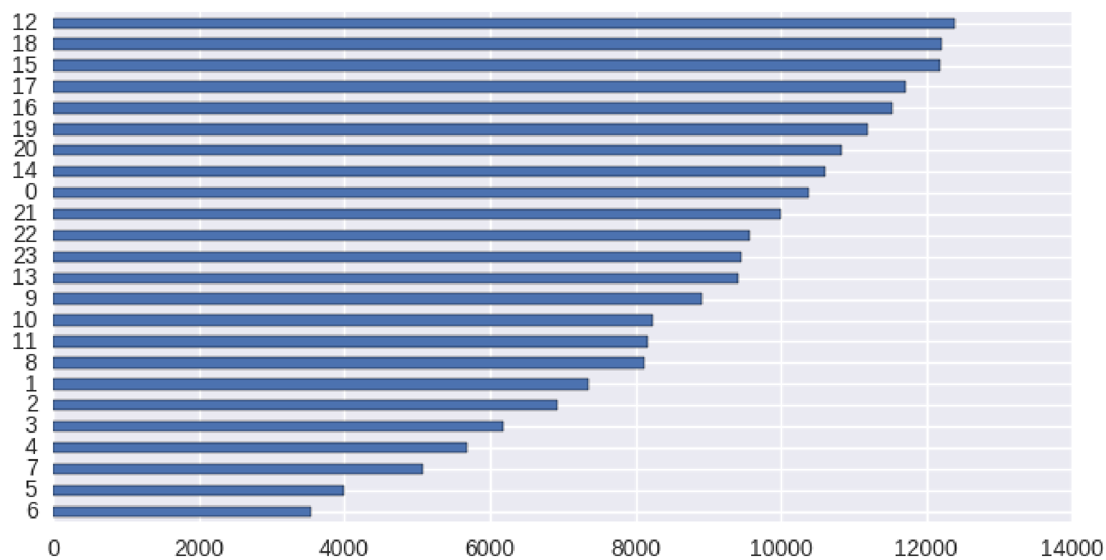
```
In [438]: year_month = X_train["Occurrence Date"].apply(lambda x: "-".join((x.split

In [439]: y_train.groupby(year_month).count().plot(figsize=(8,3));
```



We could see a yearly pattern in the general amount of crimes, may be a time series model could be used to fit this pattern

Occurrence Hour

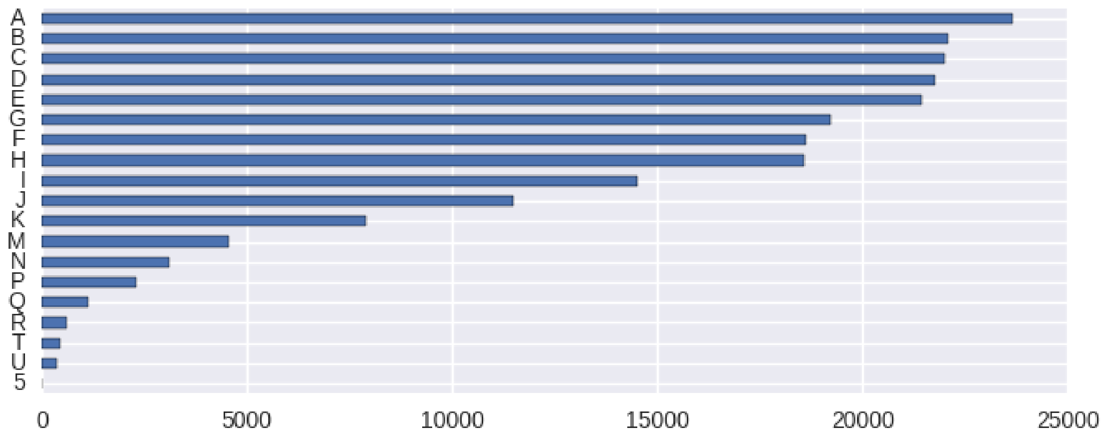
```
In [440]: X_train["Occurrence Hour"].value_counts().sort_values().plot.barh(figsize
```



From this barplot, we could feel that generally, the crime is more likely to happen from noon to evening, morning and the time after midnight will be less likely to witness crimes

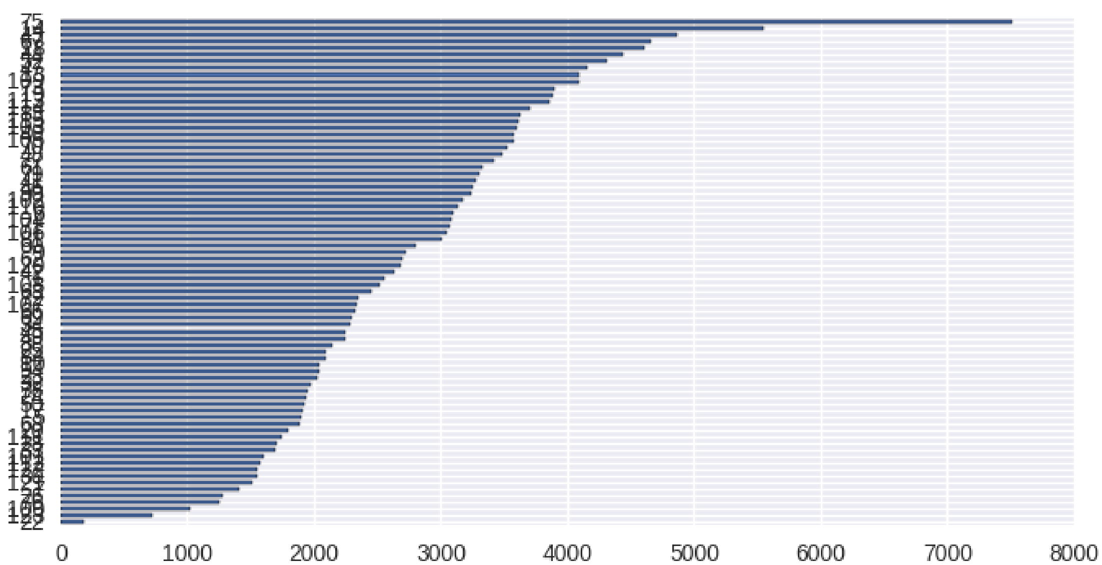
Sector and Precinct

```
In [441]: X_train["Sector"].value_counts().sort_values().plot.barh(figsize=(8,3));
```



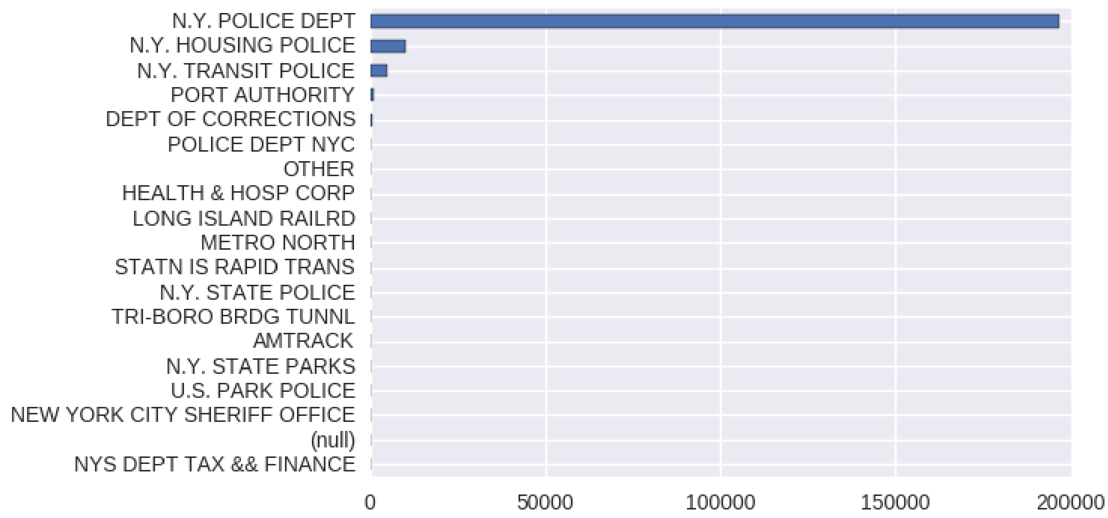
There seems to very large variation of categories in this feature, we should engineer it

```
In [442]: X_train["Precinct"].value_counts().sort_values().plot.barh(figsize=(8,4))
```



This feature is similar to the sector feature, we need to drop or create new categories based on this feature to avoid overfitting if we used tree-based models on it.(tree model prefer to overfit on feature with many categories)

```
In [443]: X_train["Jurisdiction"].value_counts().sort_values().plot.barh();
```



Most event were dealt with by NYPD, we could engineer and experiment with this feature

Multivariate exploration We will mainly rely on the ***heatmap*** for data exploration

We have check how many kinds of crimes we have

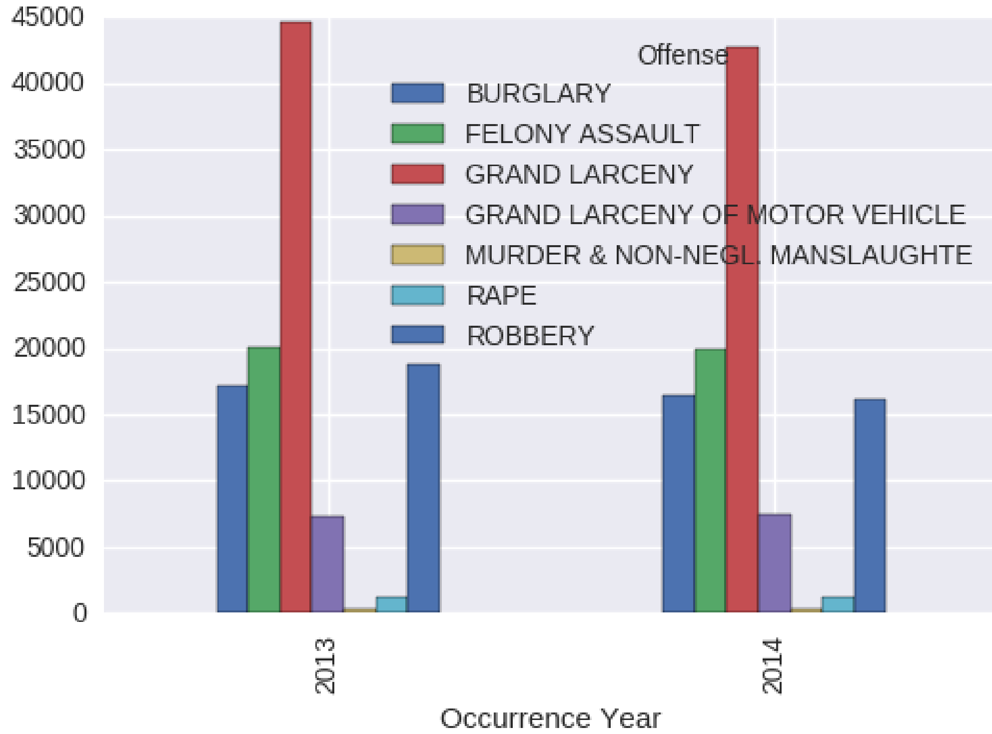
```
In [444]: y_train.value_counts()

Out[444]: GRAND LARCENY                     87261
          FELONY ASSAULT                    39948
          ROBBERY                           34997
          BURGLARY                          33685
          GRAND LARCENY OF MOTOR VEHICLE    14749
          RAPE                               2434
          MURDER & NON-NEGL. MANSLAUGHTE      615
          Name: Offense, dtype: int64
```

We have totally 7 crimes, GRAND LARCENY,FELONY ASSAULT,ROBBERY,BURGLARY are the four major ones

Let's ask some questions

Q: Does the pattern of crime changes over years?

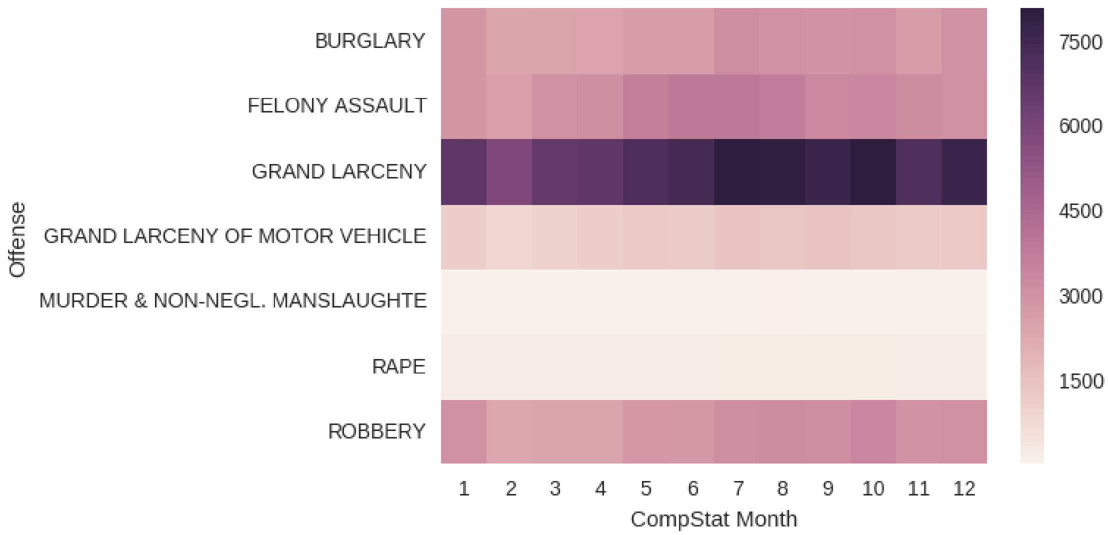
```
In [445]: y_train.groupby(X_train["Occurrence Year"]).value_counts().unstack().plot

Out[445]: <matplotlib.axes._subplots.AxesSubplot at 0x7f816e6cd630>
```



The pattern seems to stay the same

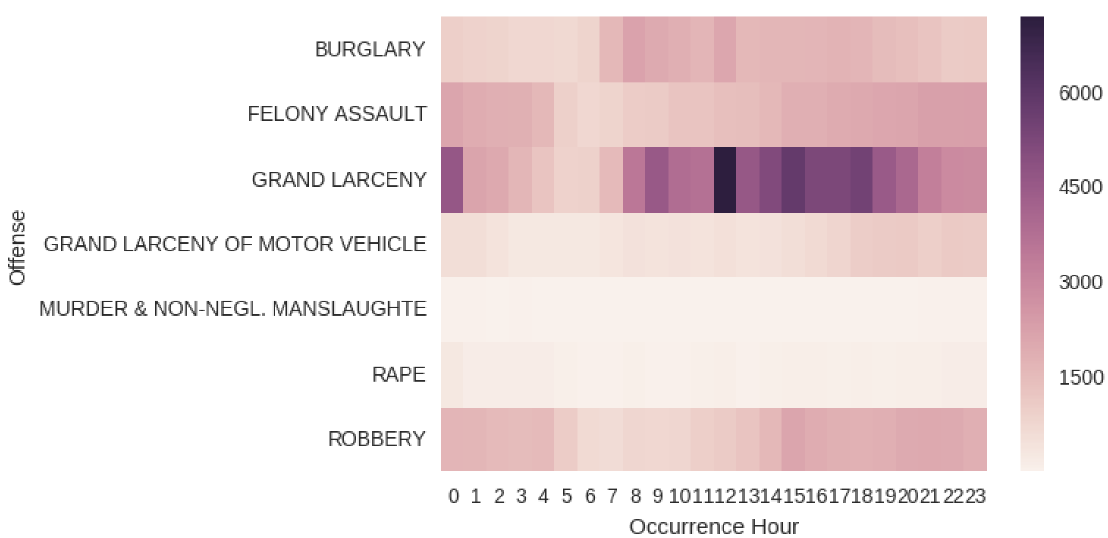
Q: How will the hour, days, month and weekday influence the crime pattern

```
In [446]: X_train.columns

Out[446]: Index(['Day of Week', 'Occurrence Year', 'Occurrence Hour', 'CompStat Mon
                 'CompStat Day', 'CompStat Year', 'Sector', 'Precinct', 'Borough',
                 'Jurisdiction', 'XCoordinate', 'YCoordinate', 'Occurrence Date'],
                dtype='object')

In [447]: #Month
          sns.heatmap(y_train.groupby(X_train["CompStat Month"]).value_counts().uns
```



1. We can see that Grand Larceny is more likely to happen at month 1, 7, 8 ,10 and 12 , month 2 is the lowest for Grand Larceny
2. The pattern of most kinds of crimes are similar
3. month is a good feature

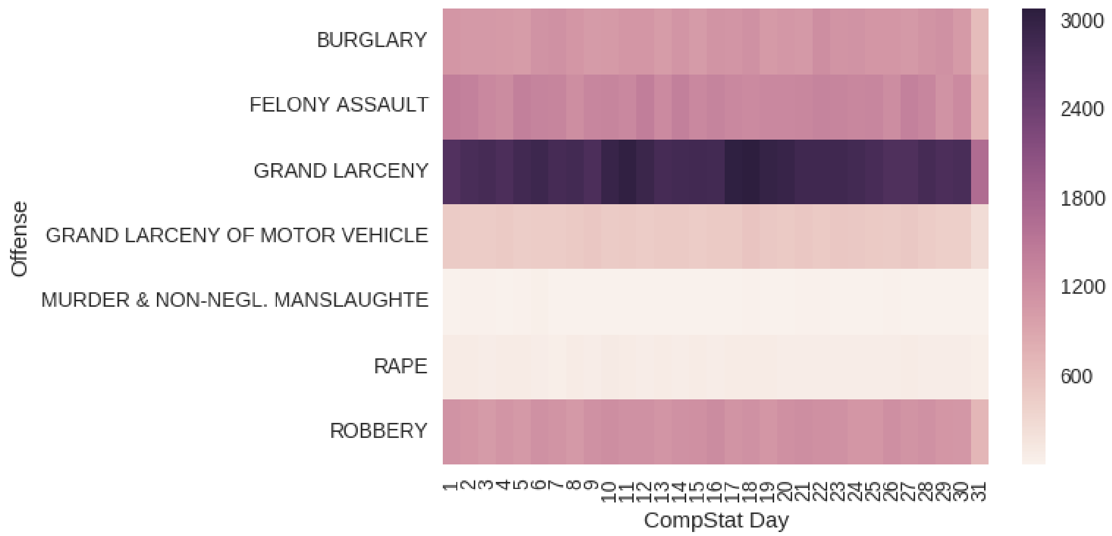
```
In [448]: #hour
          sns.heatmap(y_train.groupby(X_train["Occurrence Hour"]).value_counts().ur
```



1. We can see that the grand larceny centered at from hour 10 to hour 18, reaches the peak at 12. Midnight is also a good time for larceny.

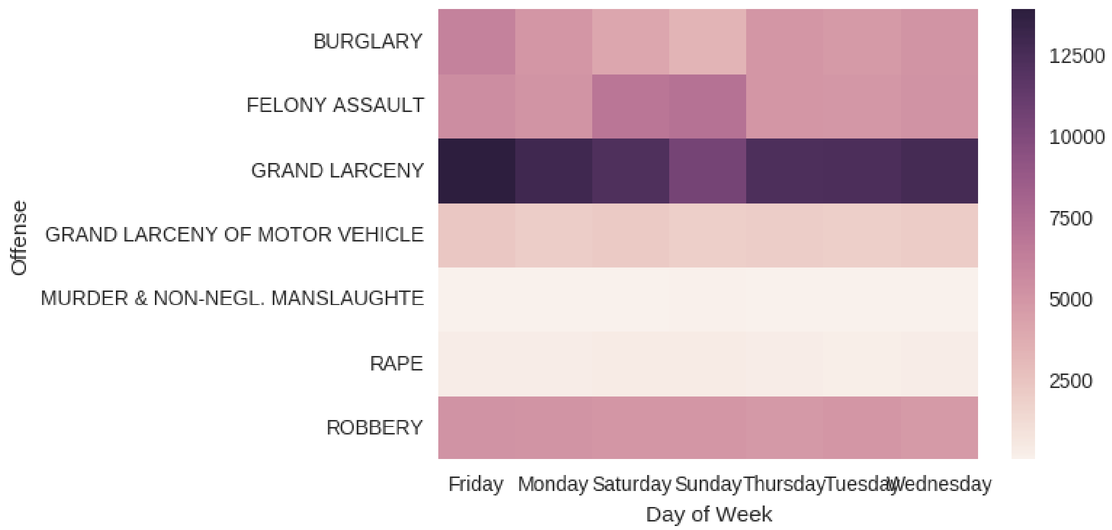
2. Felony assault and robbery less likely to happen in the morning from 6 am to 12 pm
3. This feature should be kept

```
In [449]: #Month
          sns.heatmap(y_train.groupby(X_train["CompStat Day"]).value_counts().unsta
```



No specific pattern found here, we should bin this feature in feature engineering

```
In [450]: # Weekday
          sns.heatmap(y_train.groupby(X_train["Day of Week"]).value_counts().unstac
```



1. It is easy to find that Burglary is more likely to happen at weekdays

2. Felony assault crimes increases at Saturday and Sunday(Maybe because of more people goging out to have fun)
3. Larceny of motor seems to slightly more likely to happen at Friday
4. Grand larceny is less likely to happen at Sumday

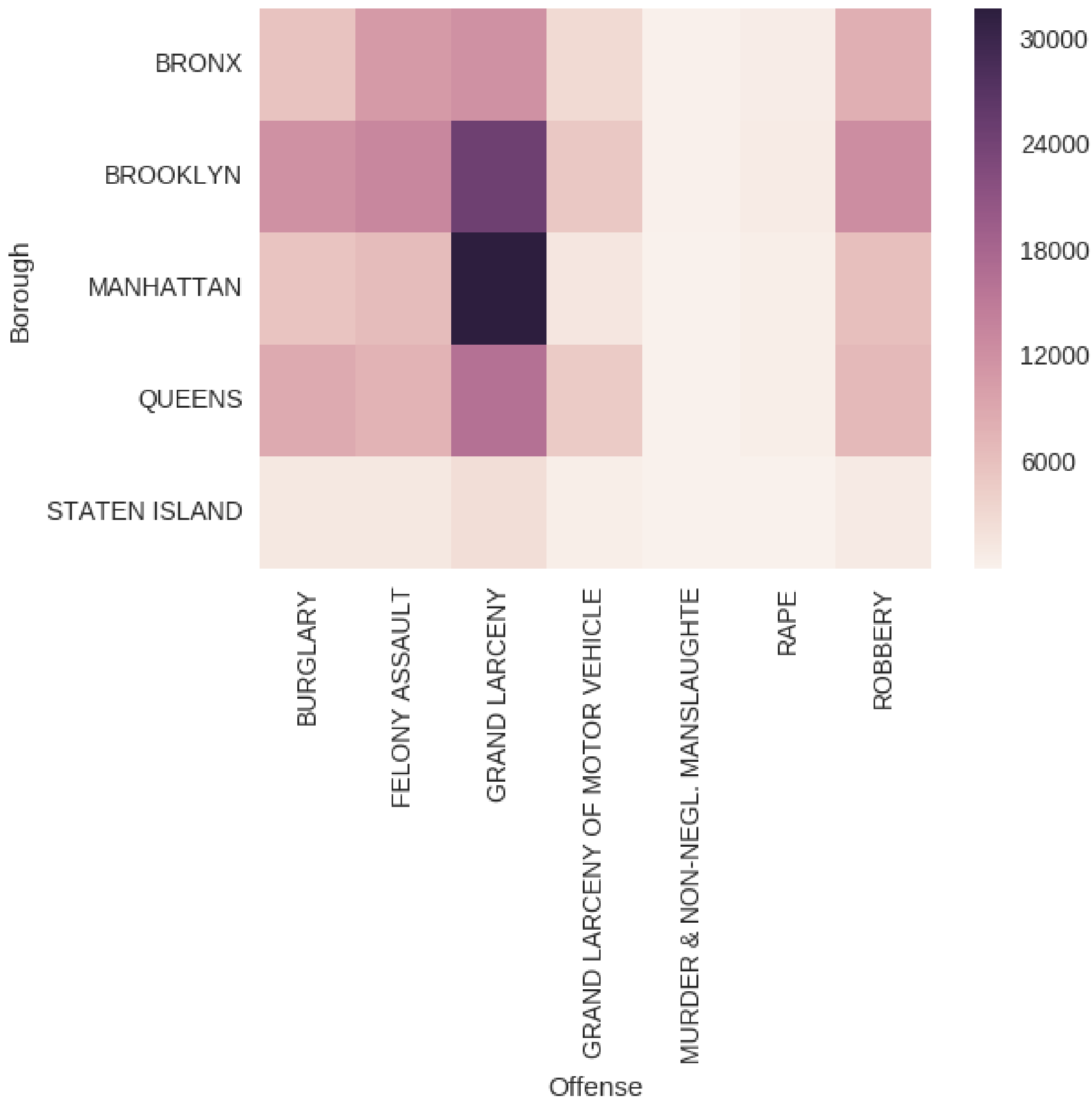
Q: Will the crime pattern change at different Borough?

```
In [451]: X_train["Borough"].value_counts()
```

```
Out[451]: BROOKLYN         68849
          MANHATTAN        52282
          QUEENS           45682
          BRONX            40410
          STATEN ISLAND     6462
          (null)               4
          Name: Borough, dtype: int64
```

Two crimes did not have there district belongings, we could remove them from the train

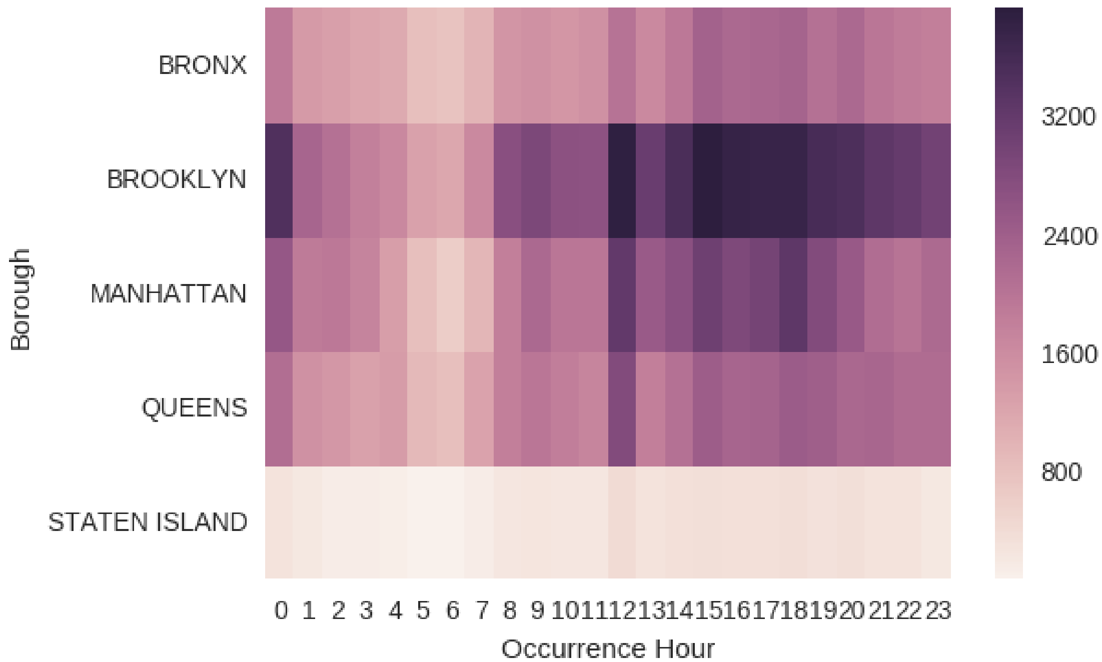
```
In [452]: sns.heatmap(y_train.groupby(X_train[X_train["Borough"] != "(null)"]["Boro
```



1. Brooklyn, Manhattan and Queens are places with more grand larceny
2. Brooklyn seems to have more felony assault and robbery
3. larceny of motor is more likely to happen at Queens and Brooklyn

Q: Does differnt Borough has different crime hour time patterns?

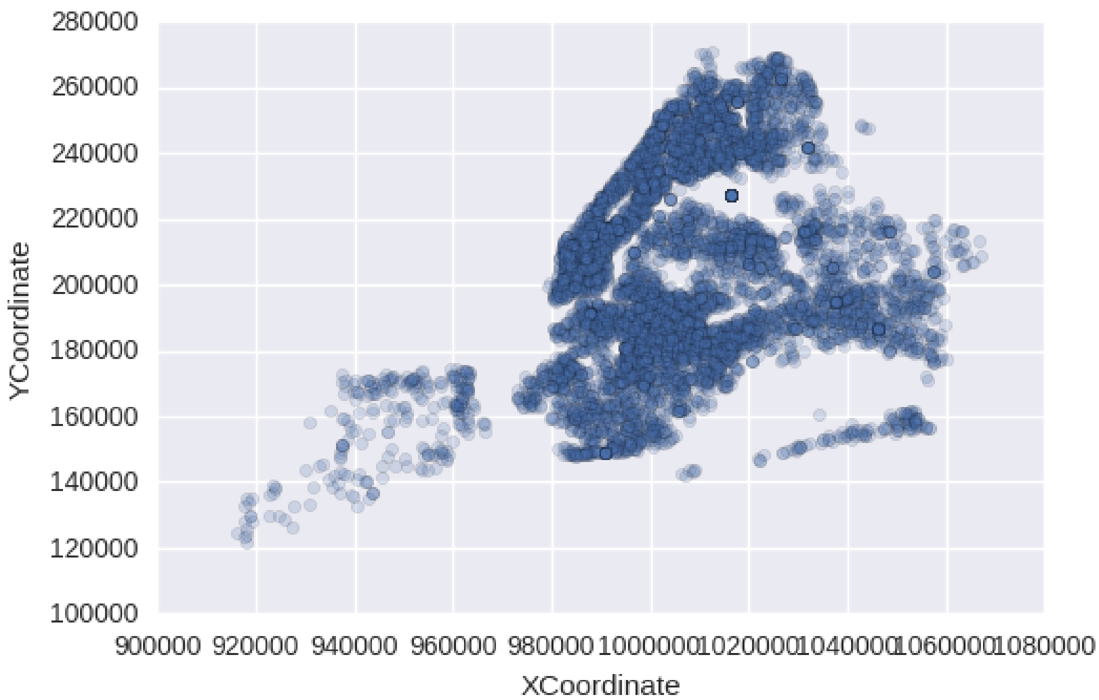
```
In [453]: sns.heatmap(X_train.groupby(X_train[X_train["Borough"] != "(null)"]["Boro
```



The general pattern is influenced by the major crimes, but we can still see that Brooklyn sees more crimes at midnight compared to other areas. (I am living in Brookly, I never went on street after 10 pm)

Q: Crime locations distribution

```
In [454]: X_train.sample(n=10000).plot(kind="scatter", x="XCoordinate", y="YCoordir
                                                alpha=0.2);
```



Most data point centered at Manhattan and Brooklyn

There are many interesting questions to ask about and visualize the relation between features and targets, I will stop here to jump to the modeling part

1.9 Feature Engineering

This part is done iteratively

Basic Engineering

```
In [455]: # drop rows with (null) Borough
          drop_row = (X_train["Borough"] != "(null)")
          X_train = X_train[drop_row]
          y_train = y_train[drop_row]
```

Since it is a short-time challenge, I will drop the data which requires more time to work with

```
In [456]: X_train.columns

Out[456]: Index(['Day of Week', 'Occurrence Year', 'Occurrence Hour', 'CompStat Mor
                 'CompStat Day', 'CompStat Year', 'Sector', 'Precinct', 'Borough',
                 'Jurisdiction', 'XCoordinate', 'YCoordinate', 'Occurrence Date'],
                dtype='object')

In [457]: drop_col_more = ['XCoordinate', 'YCoordinate', 'Occurrence Date', 'CompSt

In [458]: X_train.drop(drop_col_more, axis=1, inplace=True)
          X_test.drop(drop_col_more, axis=1, inplace=True)
```

```
In [459]: X_train.columns

Out[459]: Index(['Day of Week', 'Occurrence Hour', 'CompStat Month', 'CompStat Day'
                 'Sector', 'Precinct', 'Borough', 'Jurisdiction'],
                dtype='object')
```

We are working with categorical features, so we have to encode them before modeling
Basic encoding

```
In [460]: lb = LabelBinarizer()

In [461]: DoW_train = pd.DataFrame(lb.fit_transform(X_train["Day of Week"]),
                      columns=["Day_of_Week_{}".format(i) for i in range(X_train['

          DoW_test = pd.DataFrame(lb.fit_transform(X_test["Day of Week"]),
                      columns=["Day_of_Week_{}".format(i) for i in range(X_test["D

In [462]: Hr_train = pd.DataFrame(lb.fit_transform(X_train["Occurrence Hour"]),
                      columns=["Occurrence_Hour_{}".format(i) for i in range(X_tra

          Hr_test = pd.DataFrame(lb.fit_transform(X_test["Occurrence Hour"]),
                      columns=["Occurrence_Hour_{}".format(i) for i in range(X_tes

In [463]: Br_train = pd.DataFrame(lb.fit_transform(X_train["Borough"]),
                      columns=["Borough_{}".format(i) for i in range(X_train["Boro

          Br_test = pd.DataFrame(lb.fit_transform(X_test["Borough"]),
                      columns=["Borough_{}".format(i) for i in range(X_test["Borou
```

Encoding and engineering
I will transform the data before encoding
For CompStat Day, I will cut the day of month into 3 ranges (1,10), (11,20) and (20,30)

```
In [464]: D_train = pd.cut(X_train["CompStat Day"], bins=[0, 10, 20, 31], right=Tru
          D_train = pd.DataFrame(lb.fit_transform(D_train),
                      columns=["Day_{}".format(i) for i in range(D_train.value_cou

          D_test = pd.cut(X_test["CompStat Day"], bins=[0, 10, 20, 31], right=True,
          D_test = pd.DataFrame(lb.fit_transform(D_test),
                      columns=["Day_{}".format(i) for i in range(D_test.value_coun

In [465]: assert D_train.shape[1] == D_test.shape[1]
```

For Sector, I will use the top 10 major categories and group the left to “others” category

```
In [466]: sector_major = list(X_train["Sector"].value_counts().sort_values(ascendin

In [467]: Sc_train = X_train["Sector"].apply(lambda x: x if x in sector_major else

          Sc_train = pd.DataFrame(lb.fit_transform(Sc_train),
                      columns=["Sc_{}".format(i) for i in range(Sc_train.value_cou
```

```
In [468]: Sc_test = X_test["Sector"].apply(lambda x: x if x in sector_major else "c

          Sc_test = pd.DataFrame(lb.fit_transform(Sc_test),
                      columns=["Sc_{}".format(i) for i in range(Sc_test.value_coun

In [469]: assert Sc_train.shape[1] == Sc_test.shape[1]
```

For Precinct, to be safe, I will only create an indicator of the top major category

```
In [470]: Pr_major = X_train["Precinct"].value_counts().sort_values(ascending=False

          Pr_train = (X_train["Precinct"] == Pr_major) * 1
          Pr_test = (X_test["Precinct"] == Pr_major) * 1
```

For Jurisdiction, I will extract top 3 major classes

```
In [471]: Ju_major = X_train["Jurisdiction"].value_counts().sort_values(ascending=F

In [472]: Ju_train = X_train["Jurisdiction"].apply(lambda x: x if x in Ju_major els

          Ju_train = pd.DataFrame(lb.fit_transform(Ju_train),
                      columns=["Ju_{}".format(i) for i in range(Ju_train.value_cou

In [473]: Ju_test = X_test["Jurisdiction"].apply(lambda x: x if x in Ju_major else

          Ju_test = pd.DataFrame(lb.fit_transform(Ju_test),
                      columns=["Ju_{}".format(i) for i in range(Ju_test.value_coun

In [474]: assert Ju_train.shape[1] == Ju_test.shape[1]
```

Combine the engineered data

```
In [475]: X_train_eg = np.concatenate([DoW_train.as_matrix(), Hr_train.as_matrix(),
                                       , D_train.as_matrix(), Sc_train.as_matrix(),
                                       Ju_train.as_matrix()], axis=1)

In [476]: X_train_eg.shape

Out[476]: (213685, 55)

In [477]: X_test_eg = np.concatenate([DoW_test.as_matrix(), Hr_test.as_matrix(), Br
                                      , D_test.as_matrix(), Sc_test.as_matrix(), P
                                      Ju_test.as_matrix()], axis=1)

In [478]: X_test_eg.shape

Out[478]: (100599, 55)
```

We have finished the basic engineering and could start to split the data and build the model

1.10 Train and Validation set split — Stratefied

Now split the data into 2 data set 1. train 2. validation

Only use the data of train and validation for model training Use the test to evaluate the generalization of the final model

```
In [479]: X_tr, X_val, y_tr, y_val = train_test_split(X_train_eg, y_train, test_siz
```

Transform y from str to int

```
In [480]: ecd = LabelEncoder()
          ecd.fit(y_tr)

Out[480]: LabelEncoder()

In [481]: y_tr = ecd.transform(y_tr)
          y_val = ecd.transform(y_val)
          y_test = ecd.transform(y_test)
```

1.11 Model prototyping

Since our data set contains multiple binary encoding data, it is better to start from the ***tree based model***

I will only experiment with gradient boosting tree based on the time limitation

```
In [487]: params = {'eta': 0.08, 'max_depth': 4, 'subsample': 0.8, 'colsample_bytre
                    'objective': 'multi:softmax', 'silent': True}

          num_round = 500

In [488]: xg_train = xgb.DMatrix(X_tr, label=y_tr)
          xg_val = xgb.DMatrix(X_val, label=y_val)

In [489]: watchlist = [(xg_train, 'train'), (xg_val, 'val')]

In [490]: def acc(preds, dtrain):
              labels = dtrain.get_label()
              acc_score = accuracy_score(labels, preds, normalize=True, sample_weig
              return 'acc', acc_score

In [491]: bst = xgb.train(params, xg_train, num_round,
                          watchlist, early_stopping_rounds=50,
                          feval=acc, maximize=True, verbose_eval=20)

[0]	train-acc:0.423842	val-acc:0.422234
Multiple eval metrics have been passed: 'val-acc' will be used for early stopping.

Will train until val-acc hasn't improved in 50 rounds.
[20]	train-acc:0.428557	val-acc:0.428294
[40]	train-acc:0.433085	val-acc:0.431125
```

```
[60]        train-acc:0.436326       val-acc:0.434588
[80]        train-acc:0.438455       val-acc:0.436343
[100]        train-acc:0.439953       val-acc:0.437841
[120]        train-acc:0.441263       val-acc:0.438917
[140]        train-acc:0.442333       val-acc:0.439385
[160]        train-acc:0.443404       val-acc:0.439455
[180]        train-acc:0.443474       val-acc:0.439432
[200]        train-acc:0.443907       val-acc:0.439619
Stopping. Best iteration:
[168]        train-acc:0.443386       val-acc:0.439876
```

1.11.1 Error Analysis

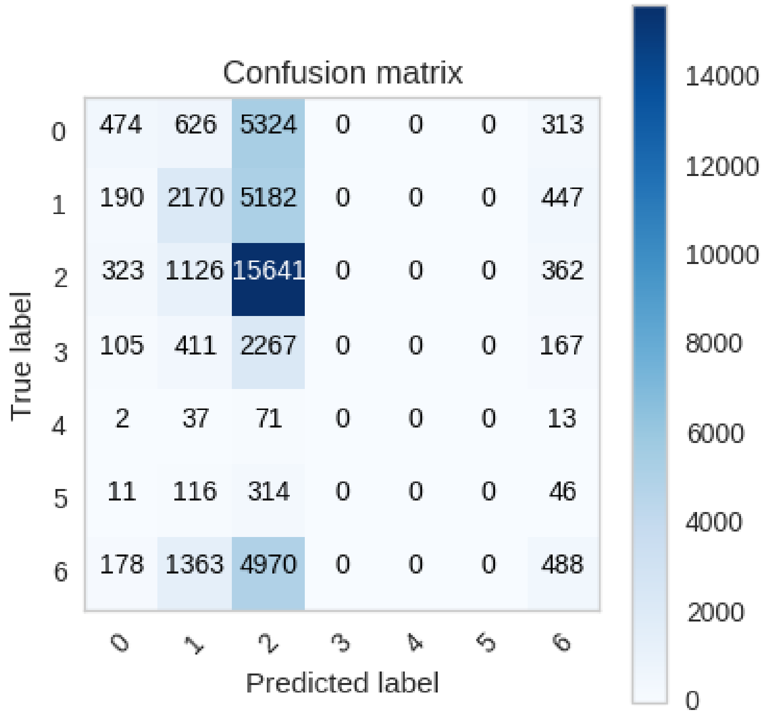
The model seems perform not very well, only surpass the dummy model for about 3%, what happened?

In [500]:
```
# get prediction
val_pred = bst.predict(xg_val)
accuracy_score(y_val, val_pred, normalize=True, sample_weight=None)
```

Out[500]:
```
0.4392680815218663
```

In [510]:
```
# Compute confusion matrix
cnf_matrix = confusion_matrix(y_val, val_pred)
np.set_printoptions(precision=2)

# Plot confusion matrix
plt.figure(figsize=(4,4));
plot_confusion_matrix(cnf_matrix, classes = [0,1,2,3,4,5,6], title='Confu
print(ecd.classes_)
```

```
Confusion matrix, without normalization
['BURGLARY' 'FELONY ASSAULT' 'GRAND LARCENY'
 'GRAND LARCENY OF MOTOR VEHICLE' 'MURDER & NON-NEGL. MANSLAUGHTE' 'RAPE'
 'ROBBERY']
```



1. We can see that we missed a lot on the prediction of class 3,4,5, all of them are mis-classified to be the major crime: GRAND LARCENY.
2. The classification of 'BURGLARY', 'FELONY ASSAULT' seems to be hard to classify because they share similar mis-classification problems. We should dive into data more to dig out the features for class 'GRAND LARCENY OF MOTOR VEHICLE', 'MURDER & NON-NEGL. MANSLAUGHTE','RAPE'. They are minor classes and will be seriously influenced by the major class
3. Class 6 ROBBERY seems to be easier to classify, but still missed a lot on the 3 major classes

```
In [506]: ecd.classes_
```

```
Out[506]: array(['BURGLARY', 'FELONY ASSAULT', 'GRAND LARCENY',
                 'GRAND LARCENY OF MOTOR VEHICLE', 'MURDER & NON-NEGL. MANSLAUGHTE'
                 'RAPE', 'ROBBERY'], dtype=object)
```

```
In [ ]:
```

1.12 Model fine tuning

I only tried manual tuning because of time limitation, the best one is the one above

```
In [ ]: params = {'eta': 0.08, 'max_depth': 4, 'subsample': 0.8, 'colsample_bytree'
                  'objective': 'multi:softmax', 'silent': True}
```

```
In [ ]:
```

1.13 Ensemble

Can not finish because fitting a single model will consume much time

```
In [ ]:
```

1.14 Prediction on Test

```
In [512]: xg_test =  xgb.DMatrix(X_test_eg)
```

```
In [513]: test_pred = bst.predict(xg_test)
          accuracy_score(y_test, test_pred, normalize=True, sample_weight=None)
```

```
Out[513]: 0.43733039095816062
```

Classification results on test
The best performance with this model is: ***0.43733039095816062***

1.15 Export

```
In [522]: test_output = ecd.inverse_transform(test_pred.astype(int))
```

```
In [523]: test_output)
```

```
Out[523]: array(['GRAND LARCENY', 'FELONY ASSAULT', 'GRAND LARCENY', ...,
                 'FELONY ASSAULT', 'FELONY ASSAULT', 'FELONY ASSAULT'], dtype=objec
```

```
In [526]: test_submit = pd.DataFrame({"id":id_test,"Offense":test_output})
          # fe_submit.to_csv("../output/fe_single_reconstruct_1_2_3_4_6_md_params2.
```

```
In [529]: test_submit.to_csv("../output/test_predict.csv", index=False)
```

1.16 Proposals

```
What I will do with more time
```

1.How would you improve your model if you had another hour?

I will try to check any error in the feature engineering part, which may influence the results. Have a look at the feature with multiple categories, to see whether it is possible to transform them in another way

2.How would you improve your model if you had another week?

If I have one more week, I will dive into the error analysis. Based on it, I will check if it is possible to do more feature engineering that will extract features revealling the patterns of the mis-classified classes. I will also ensemble multiple single model to push the accuracy forward

3.What approach did you use?

I used the gradient boosting tree model for this multiclass classification problem

4.Why did you use this approach?

The model is based on tree model, which gives great flexibility. GBT model is fast to train and could reveal high order interactions and non-linear relations. Further, xgboost is inplemented to run fast and in parallel

If you've explored the data, please summarize key observations that you've made.

All the exploration and explanation about the data are in the EDA part above

In [ ]:

In [ ]:

In [ ]:

In [ ]: