San Francisco Crime Classification

Business Understanding

This dataset presents tremendous opportunity for the city of San Francisco. We are looking into the kaggle dataset (San Francisco Crime Classification) that seeks to provide insight into the crime that affects the 10 police districts of San Francisco. By looking into this study and providing insights we hope that institutions such the San Francisco Police Force will adapt these practices and influence other major cities to drive change. We see countless incidents of policing and crime, such as #blacklivesmatter and #operationheadcam, that could be better handled with more data. We seek to provide answers for classifying where crimes occur in San Francisco and classification methods for violent and non-violent crimes. We hope this study will shed light into how the San Francisco Police Force can better police and stop crime, as well as influence other institutions into adopting open data practices.

In [1]:
```
# Import mathematical libraries for Python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import scipy
```

In [2]:
```
# Draw inline
%matplotlib inline

# Create our dataframes
train_users = pd.read_csv('sf_train.csv')
```

In [3]:
```
print 'We have', train_users.shape[0], 'users in the training set.'
```

```
We have 878049 users in the training set.
```

Data Understanding

We have access to 12 years of crime data within the San Francisco city limits. The data was collected on alternating weeks (Week 1, Week 3, Week 5, etc.) so that off weeks would be put into a new data set (for testing). A few of the data points give a further look into the data (marked below by ‘*’), but are not helpful for prediction because the variables are too hard to predict and we often find them out after the response by the department. The main purpose of this data is to learn through visualization, as well as predict crime type based on location and time of the incident.Date – A timestamp of the crime incidentCategory – category of the crime incident*Descript – description of the crime incidentDaysOFWeek – Day of the week the crime occurredPdDistrict – Name of the police department district*Resolution – The end result of the crime*Address – Approximate address of the crime incidentX – Longitude (negative value signifies western hemisphere)Y – Latitude (positive value signifies

Data Preparation

We begin our data preparation by grouping the categories of crime into 2 groups, Violent vs. Non-Violent, to help answer our first question. Then we will encode the value of category to be an integer between 0 and 39. Using the timestamp, we will break it down to features such as month, day and hour. In preparation of our second question, we will encode the PdDistricts into 3 groups, West, NorthEast and SouthEast. As we continue massaging our dataset, we will One-Hot Encode the PdDistrict feature, which results in 10 new columns. Finally, we will remove any columns that we no longer need in our data.

Our final dataset for question 1 will be:Latitude, Longitude, Time_Hour, Time_Quarter, Month, Day_of_Month, DayOfWeek_Num, PdDistrict_BAYVIEW, PdDistrict_CENTRAL, PdDistrict_INGLESIDE, PdDistrict_MISSION, PdDistrict_NORTHERN, PdDistrict_PARK, PdDistrict_RICHMOND, PdDistrict_SOUTHERN, PdDistrict_TARAVAL, PdDistrict_TENDERLOIN.

Our final dataset for question 2 will be:Category, Latitude, Longitude, violent, Time_Hour, Month, Time_Quarter, Day_of_Month, DayOfWeek_Num.

In [4]:
```
# Grouping the crime into 2 sets: Violent/Non-Violent

violent_crime=['EXTORTION', 'VANDALISM', 'LARCENY/THEFT', 'ROBBERY', 'DI
SORDERLY CONDUCT', 'VEHICLE THEFT', 'ASSAULT',
               'KIDNAPPING','ARSON', 'FAMILY OFFENSES', 'SEX OFFENSES NO
N FORCIBL', 'WEAPON LAWS', 'DRUNKENNESS',
               'SUICIDE', 'DRUG/NARCOTIC', 'SEX OFFENSES FORCIBLE', 'WAR
RANTS', 'BURGLARY']

non_violent_crime=['FRAUD', 'FORGERY/COUNTERFEITING', 'BAD CHECKS', 'EMB
EZZLEMENT', 'SUSPICIOUS OCC', 'BRIBERY',
                   'STOLEN PROPERTY', 'DRIVING UNDER THE INFLUENCE', 'L
IQUOR LAWS', 'TRESPASS', 'RECOVERED VEHICLE',
                   'MISSING PERSON', 'RUNAWAY', 'PORNOGRAPHY/OBSCENE MA
T', 'LOITERING', 'SECONDARY CODES', 'GAMBLING',
                   'PROSTITUTION', 'OTHER OFFENSES', 'NON-CRIMINAL', 'T
REA']

train_users['violent'] = [1 if x in violent_crime else 0 for x in train_
users['Category']]

train_users.head(10)
train_users.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 878049 entries, 0 to 878048
Data columns (total 10 columns):
Dates         878049 non-null object
Category      878049 non-null object
Descript      878049 non-null object
DayOfWeek     878049 non-null object
PdDistrict    878049 non-null object
Resolution    878049 non-null object
Address       878049 non-null object
X             878049 non-null float64
Y             878049 non-null float64
violent       878049 non-null int64
dtypes: float64(2), int64(1), object(7)
memory usage: 73.7+ MB
```

Below we are going to tranform the data into a suitable form to run analysis. The variable that we will be predicting is Category. In our dataset, there is 40 different types of crime that is categorized. In order to perform classification on our dataset, we will encode these Categories with an integer between 0 and 39.

In [5]:

```
# Encoding the Category Variable

train_users['Category'] = train_users.Category.replace(to_replace='FRAU
D',value=0)
train_users['Category'] = train_users.Category.replace(to_replace='FORGE
RY/COUNTERFEITING',value=1)
train_users['Category'] = train_users.Category.replace(to_replace='BAD C
HECKS',value=2)
train_users['Category'] = train_users.Category.replace(to_replace='EXTOR
TION',value=3)
train_users['Category'] = train_users.Category.replace(to_replace='EMBEZ
ZLEMENT',value=4)
train_users['Category'] = train_users.Category.replace(to_replace='SUSPI
CIOUS OCC',value=5)
train_users['Category'] = train_users.Category.replace(to_replace='BRIBE
RY',value=6)
train_users['Category'] = train_users.Category.replace(to_replace='VANDA
LISM',value=7)
train_users['Category'] = train_users.Category.replace(to_replace='LARCE
NY/THEFT',value=8)
train_users['Category'] = train_users.Category.replace(to_replace='STOLE
N PROPERTY',value=9)
train_users['Category'] = train_users.Category.replace(to_replace='ROBBE
RY',value=10)
train_users['Category'] = train_users.Category.replace(to_replace='DRIVI
NG UNDER THE INFLUENCE',value=11)
train_users['Category'] = train_users.Category.replace(to_replace='DISOR
DERLY CONDUCT',value=12)
train_users['Category'] = train_users.Category.replace(to_replace='LIQUO
R LAWS',value=13)
train_users['Category'] = train_users.Category.replace(to_replace='VEHIC
LE THEFT',value=14)
train_users['Category'] = train_users.Category.replace(to_replace='ASSAU
LT',value=15)
train_users['Category'] = train_users.Category.replace(to_replace='KIDNA
PPING',value=16)
train_users['Category'] = train_users.Category.replace(to_replace='TRESP
ASS',value=17)
train_users['Category'] = train_users.Category.replace(to_replace='ARSO
N',value=18)
train_users['Category'] = train_users.Category.replace(to_replace='RECOV
ERED VEHICLE',value=19)
train_users['Category'] = train_users.Category.replace(to_replace='MISSI
NG PERSON',value=20)
train_users['Category'] = train_users.Category.replace(to_replace='RUNAW
AY',value=21)
train_users['Category'] = train_users.Category.replace(to_replace='FAMIL
Y OFFENSES',value=22)
train_users['Category'] = train_users.Category.replace(to_replace='SEX O
FFENSES NON FORCIBL',value=23)
train_users['Category'] = train_users.Category.replace(to_replace='PORNO
GRAPHY/OBSCENE MAT',value=24)
train_users['Category'] = train_users.Category.replace(to_replace='WEAPO
```

```
N LAWS',value=25)
train_users['Category'] = train_users.Category.replace(to_replace='DRUNK
ENNESS',value=26)
train_users['Category'] = train_users.Category.replace(to_replace='SUICI
DE',value=27)
train_users['Category'] = train_users.Category.replace(to_replace='TRE
A',value=28)
train_users['Category'] = train_users.Category.replace(to_replace='DRUG/
NARCOTIC',value=29)
train_users['Category'] = train_users.Category.replace(to_replace='SEX O
FFENSES FORCIBLE',value=30)
train_users['Category'] = train_users.Category.replace(to_replace='LOITE
RING',value=31)
train_users['Category'] = train_users.Category.replace(to_replace='WARRA
NTS',value=32)
train_users['Category'] = train_users.Category.replace(to_replace='BURGL
ARY',value=33)
train_users['Category'] = train_users.Category.replace(to_replace='SECON
DARY CODES',value=34)
train_users['Category'] = train_users.Category.replace(to_replace='GAMBL
ING',value=35)
train_users['Category'] = train_users.Category.replace(to_replace='SEX O
FFENSES NON FORCIBLE',value=36)
train_users['Category'] = train_users.Category.replace(to_replace='PROST
ITUTION',value=37)
train_users['Category'] = train_users.Category.replace(to_replace='OTHER
OFFENSES',value=38)
train_users['Category'] = train_users.Category.replace(to_replace='NON-C
RIMINAL',value=39)

train_users.head(10)
```

Out[5]:

	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Addre
0	2015-05-13 23:53:00	32	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
1	2015-05-13 23:53:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
2	2015-05-13 23:33:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANNI GREE ST
3	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 L LOMB
4	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Bl BROD
5	2015-05-13 23:30:00	8	GRAND THEFT FROM UNLOCKED AUTO	Wednesday	INGLESIDE	NONE	0 Bloc TEDD
6	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	INGLESIDE	NONE	AVALO PERU
7	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	BAYVIEW	NONE	KIRKW DONA
8	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	RICHMOND	NONE	600 Bl 47TH
9	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	CENTRAL	NONE	JEFFE / LEAVE ST

In [6]:
```
# Building more features based on the DateTime field to help with our an
alysis

train_users['Dates'] = pd.to_datetime(train_users['Dates'])
train_users['Time'],train_users['Date']= train_users['Dates'].apply(lamb
da x:x.time()), train_users['Dates'].apply(lambda x:x.date())

train_users['Time_Hour'] = train_users['Time'].apply(lambda x: x.hour)
train_users['Time_Quarter'] = train_users['Time'].map(lambda x: ((x.minu
te - 1)//15 +1))
train_users['Month'] = train_users['Date'].map(lambda x: x.month)
train_users['Day_of_Month'] = train_users['Date'].map(lambda x: x.day)
train_users['DayOfWeek_Num'] = train_users['Date'].map(lambda x: x.weekd
ay())

train_users.head(10)
```

Out[6]:

	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Addre
0	2015-05-13 23:53:00	32	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
1	2015-05-13 23:53:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
2	2015-05-13 23:33:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANN GREE ST
3	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 LOMB
4	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 BI BROD
5	2015-05-13 23:30:00	8	GRAND THEFT FROM UNLOCKED AUTO	Wednesday	INGLESIDE	NONE	0 Bloc TEDD
6	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	INGLESIDE	NONE	AVALO PERU
7	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	BAYVIEW	NONE	KIRKV DONA
8	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	RICHMOND	NONE	600 BI 47TH
9	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	CENTRAL	NONE	JEFFE / LEAVE ST

In [7]:
```
# Encoding the PD Districts into 3 areas (West, NorthEast and SouthEast)

train_users['district'] = train_users['PdDistrict']

train_users.district.replace('RICHMOND', 0, inplace=True)
train_users.district.replace('PARK', 0, inplace=True)
train_users.district.replace('TARAVAL', 0, inplace=True)
train_users.district.replace('NORTHERN', 1, inplace=True)
train_users.district.replace('CENTRAL', 1, inplace=True)
train_users.district.replace('TENDERLOIN', 1, inplace=True)
train_users.district.replace('SOUTHERN', 1, inplace=True)
train_users.district.replace('MISSION', 2, inplace=True)
train_users.district.replace('INGLESIDE', 2, inplace=True)
train_users.district.replace('BAYVIEW', 2, inplace=True)

train_users.head(10)
```

Out[7]:

	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Addre
0	2015-05-13 23:53:00	32	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
1	2015-05-13 23:53:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
2	2015-05-13 23:33:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANNI GREE ST
3	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 L LOMB
4	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Bl BROD
5	2015-05-13 23:30:00	8	GRAND THEFT FROM UNLOCKED AUTO	Wednesday	INGLESIDE	NONE	0 Bloc TEDD
6	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	INGLESIDE	NONE	AVALO PERU
7	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	BAYVIEW	NONE	KIRKW DONA
8	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	RICHMOND	NONE	600 Bl 47TH
9	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	CENTRAL	NONE	JEFFE / LEAVE ST

In [8]:
```
# Changing the Latitude to positive
# This does not effect the grouping of the data
# We received an error early on stating that X could not be a negative n
umber

train_users.rename(columns={'Y': 'Longitude', 'X': 'Latitude'}, inplace=
True)
train_users.Latitude = train_users.Latitude * -1
```

In [9]:
```
# Building dataset to answer for our second question:

district_X = train_users[['Category', 'violent', 'Time_Hour', 'Month',
'Time_Quarter', 'Day_of_Month', 'DayOfWeek_Num']]
district_y = train_users['district']
```

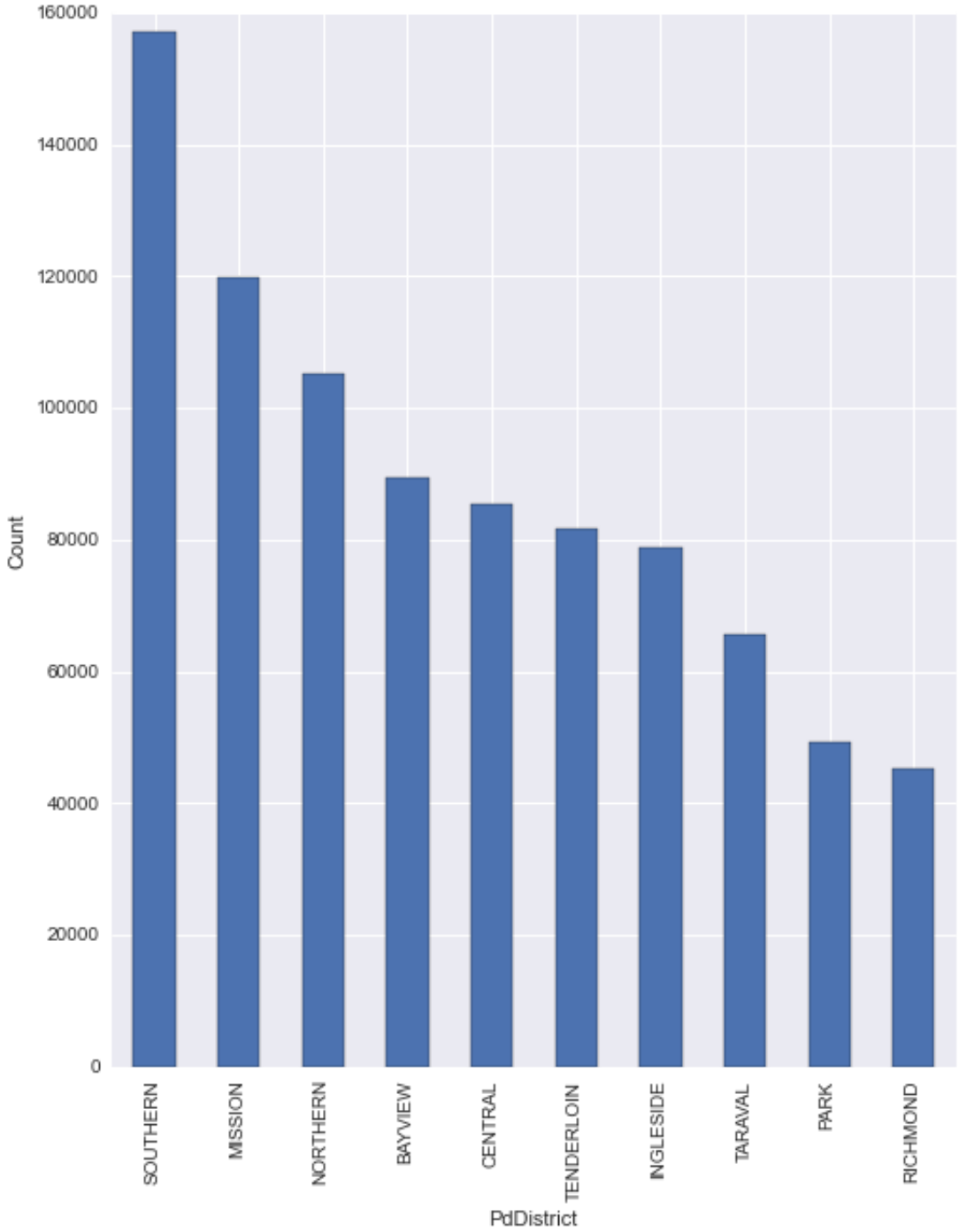
In [10]:
```
# One Hot Encoding the PdDistrict

tmp_df = pd.get_dummies(train_users.PdDistrict,prefix='PdDistrict')
train_users = pd.concat((train_users,tmp_df),axis=1) # add back into the
dataframe

train_users.head(10)
```

Out[10]:

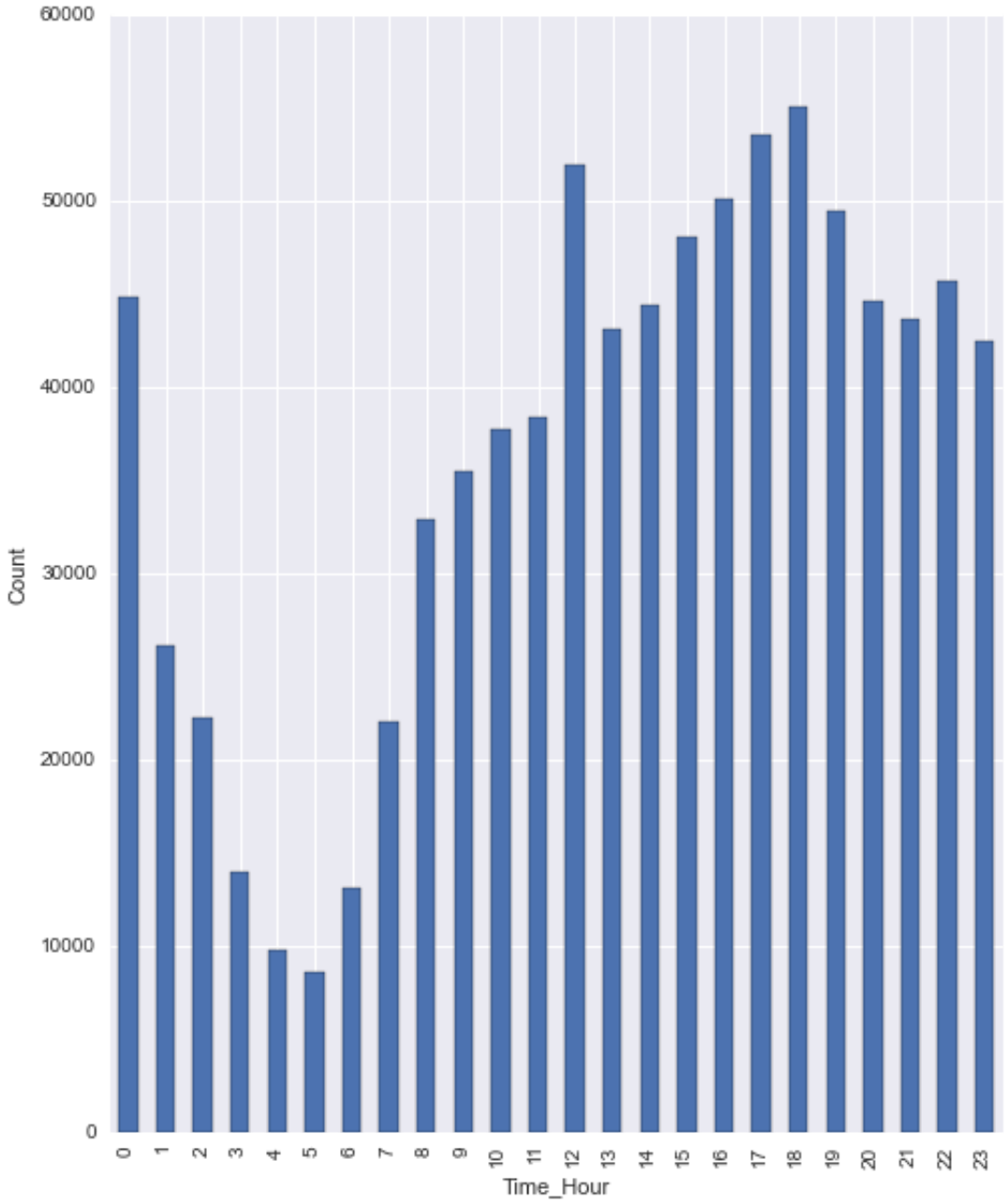
	Dates	Category	Descript	DayOfWeek	PdDistrict	Resolution	Addre
0	2015-05-13 23:53:00	32	WARRANT ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
1	2015-05-13 23:53:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	OAK S LAGU
2	2015-05-13 23:33:00	38	TRAFFIC VIOLATION ARREST	Wednesday	NORTHERN	ARREST, BOOKED	VANNI GREE ST
3	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	NORTHERN	NONE	1500 L LOMB
4	2015-05-13 23:30:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	PARK	NONE	100 Bl BROD
5	2015-05-13 23:30:00	8	GRAND THEFT FROM UNLOCKED AUTO	Wednesday	INGLESIDE	NONE	0 Bloc TEDD
6	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	INGLESIDE	NONE	AVALO PERU
7	2015-05-13 23:30:00	14	STOLEN AUTOMOBILE	Wednesday	BAYVIEW	NONE	KIRKW DONA
8	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	RICHMOND	NONE	600 Bl 47TH
9	2015-05-13 23:00:00	8	GRAND THEFT FROM LOCKED AUTO	Wednesday	CENTRAL	NONE	JEFFE / LEAVE ST

10 rows × 28 columns

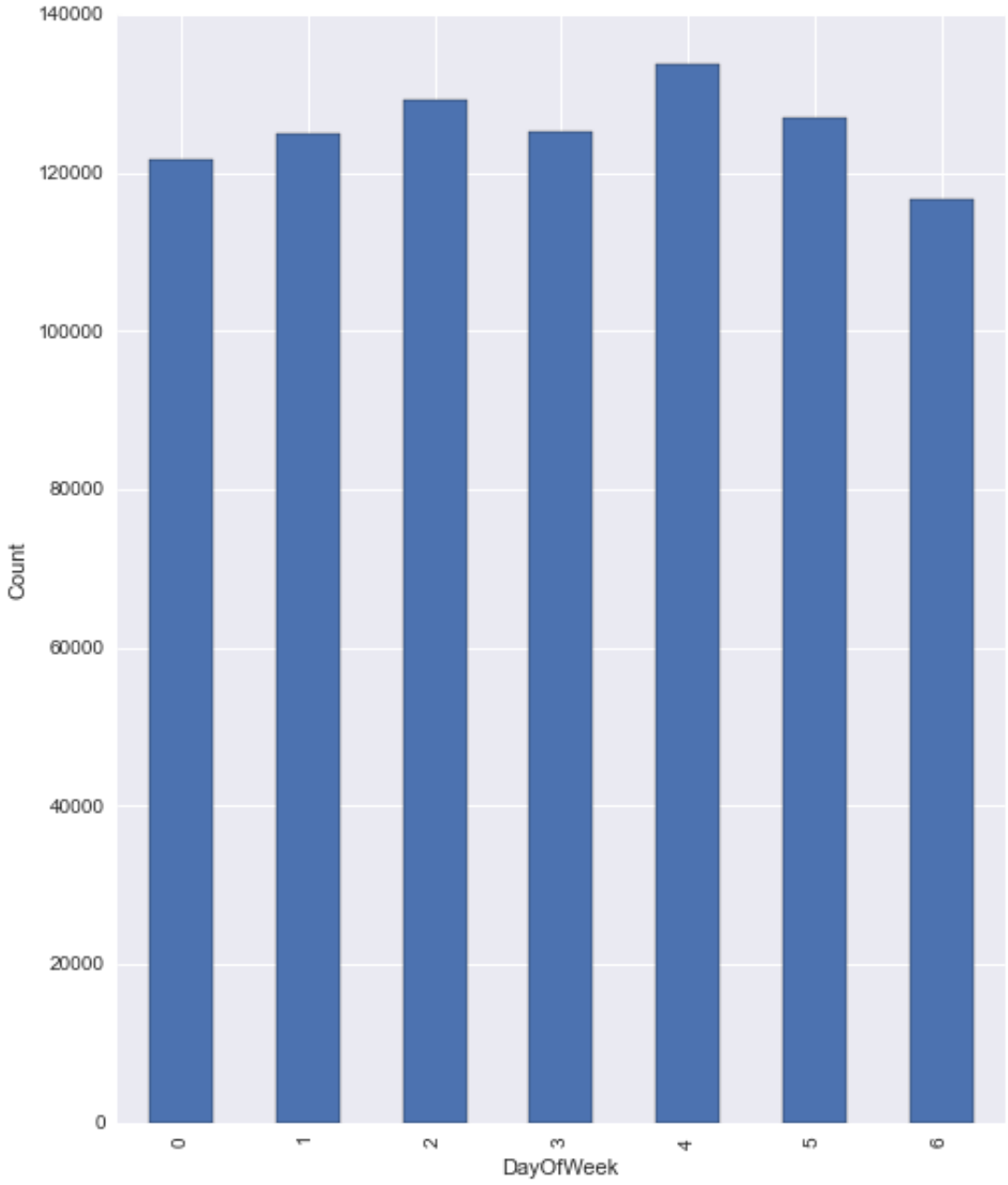
In [11]:
```
# Plot to show the number of crimes in each PD District

train_users.PdDistrict.value_counts().plot(kind='bar', figsize=(8,10))
plt.xlabel('PdDistrict')
plt.ylabel('Count')
sns.despine()
plt.show()
```



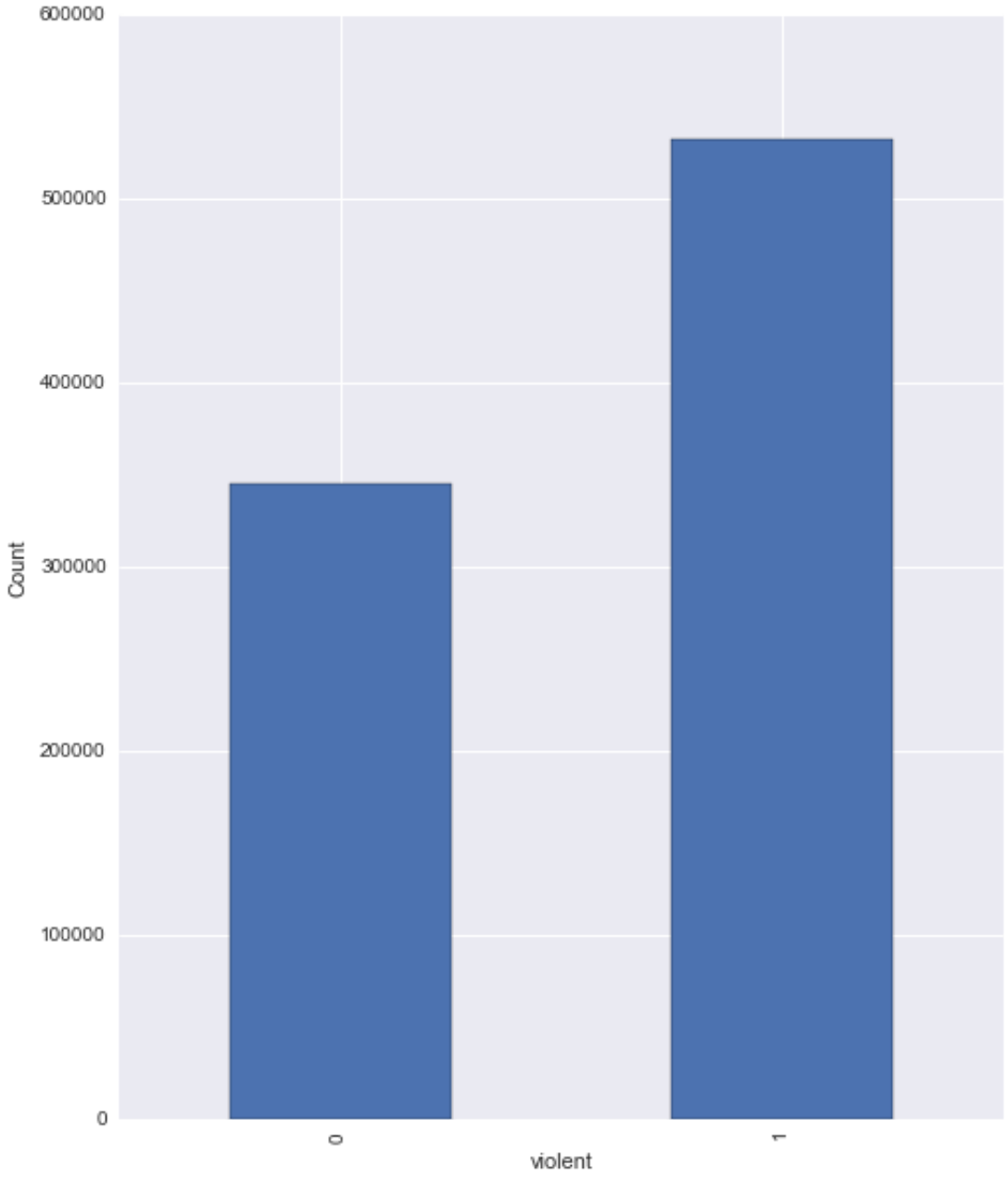
In [12]:
```
# Plot to show number of crime by each hour of the day

train_users.Time_Hour.value_counts(sort=False).plot(kind='bar', figsize=
(8,10))
plt.xlabel('Time_Hour')
plt.ylabel('Count')
sns.despine()
plt.show()
```



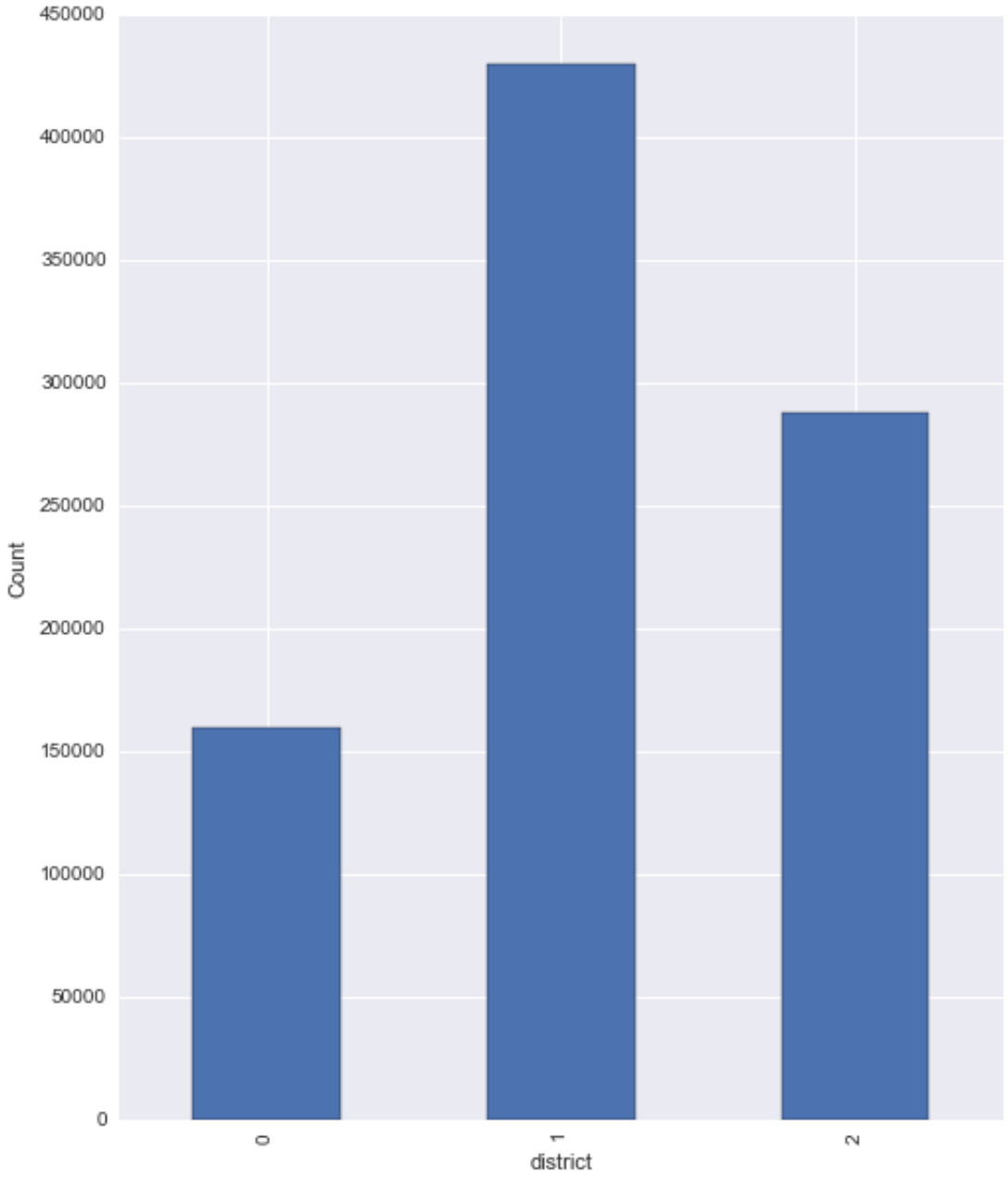
In [13]:
```
# Plot to show number of crimes per day

train_users.DayOfWeek_Num.value_counts(sort=False).plot(kind='bar', figs
ize=(8,10))
plt.xlabel('DayOfWeek')
plt.ylabel('Count')
sns.despine()
plt.show()
```



In [14]:
```
# Plot to show the total number of Violent/Non-Violent crimes

train_users.violent.value_counts(sort=False).plot(kind='bar', figsize=
(8,10))
plt.xlabel('violent')
plt.ylabel('Count')
sns.despine()
plt.show()
```



In [15]:
```
# Plot to show the number of crimes on our 3 District grouping (W, NE, S
E)

train_users.district.value_counts(sort=False).plot(kind='bar', figsize=
(8,10))
plt.xlabel('district')
plt.ylabel('Count')
sns.despine()
plt.show()
```



In [16]:
```
# Cleaning up our data set to use for question 1

Category = train_users['Category']
Violent = train_users['violent']

del train_users['Category']
del train_users['violent']
del train_users['Address']
del train_users['Resolution']
del train_users['PdDistrict']
del train_users['Dates']
del train_users['Time']
del train_users['Descript']
del train_users['DayOfWeek']
del train_users['Date']
del train_users['district']

train_users.head(10)
```

Out[16]:

	Latitude	Longitude	Time_Hour	Time_Quarter	Month	Day_of_Month	Day
0	122.425892	37.774599	23	4	5	13	2
1	122.425892	37.774599	23	4	5	13	2
2	122.424363	37.800414	23	3	5	13	2
3	122.426995	37.800873	23	2	5	13	2
4	122.438738	37.771541	23	2	5	13	2
5	122.403252	37.713431	23	2	5	13	2
6	122.423327	37.725138	23	2	5	13	2
7	122.371274	37.727564	23	2	5	13	2
8	122.508194	37.776601	23	0	5	13	2
9	122.419088	37.807802	23	0	5	13	2

Metric: Accuracy

Accuracy is the best way to prevent Type II error through evaluation. Our models are for non violent vs. violent crime and district of crime (was the correct district predicted). Both models have a relatively low level of harm as a type I error compared to a type II error. As a result accuracy of the prediction is vital in order to use the data. We assume that all police departments are prepared for a violent crime, because those are usually believed to be more severe. If our predictive model predicts a violent crime and its actually a non-violent crime, often retrospective investigation and actions by the legal system can solve that issue and “fix” the Type I Error. Where as if we predict a non-violent crime and a violent crime occurs someone can lose their life, causing irreversible damage to multiple individuals, Type II Error. Police departments, often arming their patrols, show this fear and trying preventing violent crime opposed to reacting to them. Accuracy can help to prevent these type two errors and therefore help to promote our models efficiency as a whole. Type one errors can cause a lot of harm, but losses can often be returned; while for type two errors result in just outright lost.

Modeling: Predicting Violent vs. Non-Violent Crimes

For our first question, Violent vs. Non-Violent, we decided to use Naive-Bayes, Logistic Regression and Random Forest for our classification methods. We chose Naive-Bayes as our baseline test classifier. We will hope to out perform Naive-Bayes using Logistic Regression and Random Forest. Naive-Bayes is a robust classification method against outliers, but it assumes that all the attributes are independent of each other. Logistic Regresion provides a classification for binomial and multi-nomial data. Random Forest provides a classification that is resistant to overfitting. It also provides a model with low bias that runs efficiently on large amounts of data.

Using Naive-Bayes

In [17]:
```
from sklearn.cross_validation import StratifiedKFold
from sklearn import metrics as mt
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB


# create variables we are more familiar with
X = train_users.values
y = Violent.values
yhat = np.zeros(y.shape) # we will fill this with predictions

# create cross validation iterator
cv = StratifiedKFold(y, n_folds=10)

clf_mnb = MultinomialNB(alpha=.01)
clf_bnb = BernoulliNB(alpha= .01, binarize=0.0)

# fill in your code here

yhat_mnb = np.zeros(y.shape)
yhat_bnb = np.zeros(y.shape)

for train, test in cv:
    clf_mnb.fit(X[train],y[train])
    clf_bnb.fit(X[train]>0,y[train]>0)

    yhat_mnb = clf_mnb.predict(X[test])
    yhat_bnb = clf_bnb.predict(X[test]>0)


total_accuracy_mnb = mt.accuracy_score(y[test], yhat_mnb)
total_accuracy_bnb = mt.accuracy_score(y[test]>0, yhat_bnb)


conf_mnb = mt.confusion_matrix(y[test],yhat_mnb)
print 'Accuracy MNB', total_accuracy_mnb
print "confusion matrix\n",conf_mnb

conf_bnb = mt.confusion_matrix(y[test],yhat_bnb)
print 'Accuracy BNB', total_accuracy_bnb
print "confusion matrix\n",conf_bnb
```

```
Accuracy MNB 0.576545487677
confusion matrix
[[ 6274 28243]
 [ 8938 44349]]
Accuracy BNB 0.603024919138
confusion matrix
[[  878 33639]
 [ 1217 52070]]
```

In [18]:
```
from sklearn import metrics as mt

freq_infreq_threshold = 40

# get various measures of performance
#total_accuracy = mt.accuracy_score(y, yhat)

prec_for_freq_classes = []
recall_for_infreq_classes = []
rec_tot = []
prec_tot = []

for cls in np.unique(y):
    idx = (y==cls) # get classes
    ytmp_actual = np.zeros(y.shape) # make binary class problem
    ytmp_actual[idx] = 1 # set the instances for this specific class

    ytmp_predicted = np.zeros(y.shape) # binary prediction array
    ytmp_predicted[yhat_mnb==cls] = 1

    num_in_class = sum(idx)

    rec = mt.recall_score(ytmp_actual, ytmp_predicted)
    prec = mt.precision_score(ytmp_actual, ytmp_predicted)
    rec_tot.append(rec)
    prec_tot.append(prec)

    if num_in_class < freq_infreq_threshold:
        recall_for_infreq_classes.append(rec)
    elif num_in_class >= freq_infreq_threshold:
        prec_for_freq_classes.append(prec)

print 'Total Accuracy:',total_accuracy_mnb
print 'Total Accuracy:',total_accuracy_bnb

print 'Number of infrequent faces:',len(recall_for_infreq_classes), 'wit
h average recall of:', np.mean(recall_for_infreq_classes)
print 'Number of frequent faces:',len(prec_for_freq_classes), 'with aver
age precision of:',np.mean(prec_for_freq_classes)
```

```
/Users/kareemwilliams/anaconda/lib/python2.7/site-packages/ipykernel/__
main__.py:19: VisibleDeprecationWarning: boolean index did not match in
dexed array along dimension 0; dimension is 878049 but corresponding bo
olean dimension is 87804

Total Accuracy: 0.576545487677
Total Accuracy: 0.603024919138
Number of infrequent faces: 0 with average recall of: nan
Number of frequent faces: 2 with average precision of: 0.491183083509

/Users/kareemwilliams/anaconda/lib/python2.7/site-packages/numpy/core/_
methods.py:59: RuntimeWarning: Mean of empty slice.
  warnings.warn("Mean of empty slice.", RuntimeWarning)
```

Using Naive-Bayes, we used a Multi-Nomial and Bernoulli classifiers. We chose to use these two classifiers to give us a baseline accuracy, since the feature we are predicting is binary. Both of these classifiers were able to predict violent crimes more accurately than non-violent crimes. If we could account for the greater number of violent crimes compared to non-violent crimes, this may help the classifiers better predict the categories.

Using Logistic Regression

In [19]:
```
# to use the cross validation object in scikit learn, we need to grab an
instance
#    of the object and set it up. This object will be able to split our
data into
#    training and testing splits

from sklearn.cross_validation import ShuffleSplit
from sklearn.linear_model import LogisticRegression

X = train_users.values
y = Violent.values

num_cv_iterations = 3
num_instances = len(y)

cv = StratifiedKFold(y, n_folds=10)

print cv

lr_clf = LogisticRegression(penalty='l2', C=1.0, class_weight='auto') #
get object

iter_num=0

# the indices are the rows used for training and testing in each iterati
on
for train, test in cv:
    X_train = X[train]
    y_train = y[train]

    X_test = X[test]
    y_test = y[test]

    # train the reusable logisitc regression model on the training data
    lr_clf.fit(X_train,y_train)  # train object
    y_hat = lr_clf.predict(X_test) # get test set precitions

    # now let's get the accuracy and confusion matrix for this iteration
s of training/testing
    acc = mt.accuracy_score(y_test,y_hat)
    conf = mt.confusion_matrix(y_test,y_hat)
    print "====Iteration",iter_num," ===="
    print "accuracy", acc
    print "confusion matrix\n",conf
    iter_num+=1
```

```
sklearn.cross_validation.StratifiedKFold(labels=[1 0 0 ..., 1 1 0], n_f
olds=10, shuffle=False, random_state=None)
====Iteration 0  ====
accuracy 0.558526752158
confusion matrix
[[18688 15830]
 [22934 30354]]
====Iteration 1  ====
accuracy 0.543163337357
confusion matrix
[[19280 15238]
 [24875 28413]]
====Iteration 2  ====
accuracy 0.545053868756
confusion matrix
[[18641 15877]
 [24070 29218]]
====Iteration 3  ====
accuracy 0.542890007517
confusion matrix
[[18556 15962]
 [24175 29113]]
====Iteration 4  ====
accuracy 0.52749843403
confusion matrix
[[18459 16059]
 [25429 27858]]
====Iteration 5  ====
accuracy 0.54245820236
confusion matrix
[[19136 15381]
 [24793 28494]]
====Iteration 6  ====
accuracy 0.545339620063
confusion matrix
[[18506 16011]
 [23910 29377]]
====Iteration 7  ====
accuracy 0.541877363218
confusion matrix
[[17771 16746]
 [23479 29808]]
====Iteration 8  ====
accuracy 0.540499293882
confusion matrix
[[18188 16329]
 [24017 29270]]
====Iteration 9  ====
accuracy 0.535875358754
confusion matrix
[[18189 16328]
 [24424 28863]]
```

In [20]:
```
#iterpret the weights

#iterate over the coefficients

weights = lr_clf.coef_.T # take transpose to make a column vector
variable_names = train_users.columns
for coef, name in zip(weights,variable_names):
    print name, 'has weight of ', coef[0]

print lr_clf.coef_
```

```
Latitude has weight of  -0.00228082355801
Longitude has weight of  0.000510925213055
Time_Hour has weight of  0.0203760592547
Time_Quarter has weight of  -0.091914503718
Month has weight of  0.00246122490213
Day_of_Month has weight of  0.00342555162563
DayOfWeek_Num has weight of  0.0186917017402
PdDistrict_BAYVIEW has weight of  -0.138388622298
PdDistrict_CENTRAL has weight of  0.126626508878
PdDistrict_INGLESIDE has weight of  -0.0601035984715
PdDistrict_MISSION has weight of  -0.101335991511
PdDistrict_NORTHERN has weight of  0.222477950039
PdDistrict_PARK has weight of  -0.0680217362647
PdDistrict_RICHMOND has weight of  -0.0607483241666
PdDistrict_SOUTHERN has weight of  0.0507613568487
PdDistrict_TARAVAL has weight of  -0.119989413564
PdDistrict_TENDERLOIN has weight of  0.148750038969
[[-0.00228082  0.00051093  0.02037606 -0.0919145   0.00246122  0.003425
55
   0.0186917  -0.13838862  0.12662651 -0.0601036  -0.10133599  0.222477
95
  -0.06802174 -0.06074832  0.05076136 -0.11998941  0.14875004]]
```

While using Logistic Regression, our accuracy scores dropped between roughly 3 and 6 percent from our previous classifiers. Looking at the confusion matrix, we notice that our False Positive rate has greatly decreased compared to the Naive Bayes Classifiers. Compared to random chance of 1/2, our classifier out performs randoms chance.

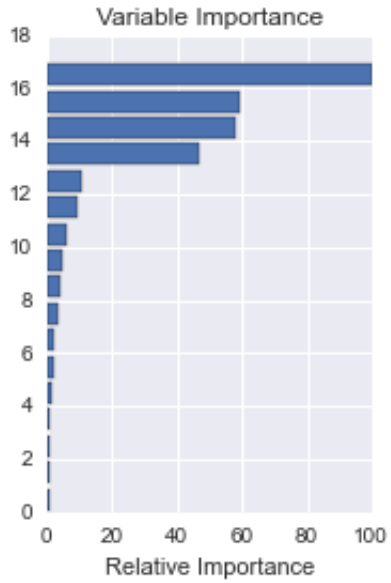
Using Random Forest Classifier

In [21]:
```
from sklearn.ensemble import RandomForestClassifier


#train_users2 = train_users.loc[train_users['Category'].isin([0,1,2,3,
5])]

#kobe_df1_pre = kobe_reg.loc[kobe_reg['season_id'].isin([1, 2])]



X = train_users.values
y = Violent.values

yhat = np.zeros(y.shape)
cv = StratifiedKFold(y, n_folds=10)

clf = RandomForestClassifier(max_depth=10, n_estimators=10, n_jobs=-1, o
ob_score=True, class_weight='auto')

# now iterate through and get predictions, saved to the correct row in y
hat
for train, test in cv:
    clf.fit(X[train],y[train])
    yhat[test] = clf.predict(X[test])

total_accuracy = mt.accuracy_score(y, yhat)
print 'Accuracy', total_accuracy

conf_rf = mt.confusion_matrix(y[test],yhat[test])
print "confusion matrix\n",conf_rf
```

```
/Users/kareemwilliams/anaconda/lib/python2.7/site-packages/sklearn/ense
mble/forest.py:379: UserWarning: Some inputs do not have OOB scores. Th
is probably means too few trees were used to compute any reliable oob e
stimates.
  warn("Some inputs do not have OOB scores. "
```

```
Accuracy 0.56281369263
confusion matrix
[[21842 12675]
 [26161 27126]]
```

In [22]:
```
# Plot feature importance
feature_importance = clf.feature_importances_
# make importances relative to max importance
feature_importance = 100.0 * (feature_importance / feature_importance.ma
x())
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance[sorted_idx], align='center')
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()



clf.fit(X, y)
importances = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indice
s[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances")
plt.bar(range(X.shape[1]), importances[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```




```
Feature ranking:
1. feature 2 (0.326410)
2. feature 1 (0.204847)
3. feature 3 (0.193661)
4. feature 0 (0.133164)
5. feature 5 (0.031038)
6. feature 6 (0.026745)
7. feature 11 (0.020152)
8. feature 4 (0.019698)
9. feature 16 (0.007747)
10. feature 10 (0.007167)
11. feature 13 (0.006222)
12. feature 7 (0.006125)
13. feature 14 (0.004813)
14. feature 9 (0.004391)
15. feature 8 (0.003203)
16. feature 15 (0.003139)
17. feature 12 (0.001478)
```

```
/Users/kareemwilliams/anaconda/lib/python2.7/site-packages/matplotlib/c
ollections.py:590: FutureWarning: elementwise comparison failed; return
ing scalar instead, but in the future will perform elementwise comparis
on
  if self._edgecolors == str('face'):
```

In [34]:
```
from sklearn.cross_validation import StratifiedKFold, cross_val_score


X = train_users.values
y = Violent.values

names=train_users.dtypes.index


scores = []
for i in range(X.shape[1]):
     score = cross_val_score(clf, X, y, scoring="roc_auc", cv=10)
     scores.append((round(np.mean(score), 3), names[i]))
print (sorted(scores, reverse=True))
```

```
[(0.604, 'Time_Quarter'), (0.604, 'PdDistrict_NORTHERN'), (0.604, 'PdDi
strict_INGLESIDE'), (0.603, 'Time_Hour'), (0.603, 'PdDistrict_TENDERLOI
N'), (0.603, 'PdDistrict_TARAVAL'), (0.603, 'PdDistrict_SOUTHERN'), (0.
603, 'PdDistrict_RICHMOND'), (0.603, 'PdDistrict_PARK'), (0.603, 'PdDis
trict_MISSION'), (0.603, 'PdDistrict_CENTRAL'), (0.603, 'PdDistrict_BAY
VIEW'), (0.603, 'Longitude'), (0.603, 'Latitude'), (0.603, 'Day_of_Mont
h'), (0.602, 'Month'), (0.602, 'DayOfWeek_Num')]
```

Using a Random Forest Classifier, our accuracy increased from Logistic Regression to about 56 percent. Also, our False Positive Rate decreased from about 46 percent to about 39 percent.

Modeling: Prediction Region of Crime

Using Naive-Bayes

In [24]:
```
from sklearn.naive_bayes import GaussianNB

# create variables we are more familiar with
X = district_X.values
y = district_y
yhat = np.zeros(y.shape) # we will fill this with predictions

# create cross validation iterator
cv = StratifiedKFold(y, n_folds=10)

clf_mnb = MultinomialNB(alpha=.01, fit_prior=True, class_prior=None)
clf_gnb = GaussianNB()

# fill in your code here

yhat_mnb = np.zeros(y.shape)
yhat_bnb = np.zeros(y.shape)

for train, test in cv:
    clf_mnb.fit(X[train],y[train])
    clf_gnb.fit(X[train],y[train])

    yhat_mnb = clf_mnb.predict(X[test])
    yhat_gnb = clf_gnb.predict(X[test])


total_accuracy_mnb = mt.accuracy_score(y[test], yhat_mnb)
total_accuracy_gnb = mt.accuracy_score(y[test], yhat_gnb)


conf_mnb = mt.confusion_matrix(y[test],yhat_mnb)
print 'Accuracy MNB', total_accuracy_mnb
print "confusion matrix\n",conf_mnb

conf_gnb = mt.confusion_matrix(y[test],yhat_gnb)
print 'Accuracy GNB', total_accuracy_gnb
print "confusion matrix\n",conf_gnb
```

```
Accuracy MNB 0.43685295491
confusion matrix
[[    0 12024  3987]
 [    0 29696 13278]
 [    0 20157  8661]]
Accuracy GNB 0.489436579616
confusion matrix
[[    0 16011     0]
 [    0 42974     0]
 [    0 28818     0]]
```

Using the Naive Bayes Multi-Nomial and Gaussian Classifers, we discovered that while our accuracy was better than chance, 1/3, they do not accurately model the data. Based on the the confusion matrix, we notice that the Multi-Nomial only predicts 2 of the 3 districts, while the Gaussian Classifier only predicts 1 of the 3 districts.

Using Logistic Regression

In [25]:
```
X = district_X.values
y = district_y.values

num_cv_iterations = 3
num_instances = len(y)
cv = StratifiedKFold(y, n_folds=10)

print cv

d2lr_clf = LogisticRegression(penalty='l2', C=1.0, class_weight='auto')
# get object

iter_num=0

# the indices are the rows used for training and testing in each iterati
on
for train, test in cv:
    X_train = X[train]
    y_train = y[train]

    X_test = X[test]
    y_test = y[test]

    # train the reusable logisitc regression model on the training data
    d2lr_clf.fit(X_train,y_train)  # train object
    y_hat = d2lr_clf.predict(X_test) # get test set precitions

    # now let's get the accuracy and confusion matrix for this iteration
s of training/testing
    acc = mt.accuracy_score(y_test,y_hat)
    conf = mt.confusion_matrix(y_test,y_hat)
    print "====Iteration",iter_num," ===="
    print "accuracy", acc
    print "confusion matrix\n",conf

    iter_num+=6
```

```
sklearn.cross_validation.StratifiedKFold(labels=[1 1 1 ..., 1 1 2], n_f
olds=10, shuffle=False, random_state=None)
====Iteration 0  ====
accuracy 0.473862833975
confusion matrix
[[  836 13989  1187]
 [ 1742 38935  2298]
 [ 1288 25694  1837]]
====Iteration 6  ====
accuracy 0.464387399494
confusion matrix
[[  884 13069  2059]
 [ 1622 36669  4684]
 [ 1195 24401  3223]]
====Iteration 12  ====
accuracy 0.46626654215
confusion matrix
[[  960 13234  1818]
 [ 1759 37360  3856]
 [ 1410 24788  2621]]
====Iteration 18  ====
accuracy 0.4642051796
confusion matrix
[[ 1110 12998  1904]
 [ 2026 36581  4368]
 [ 1505 24245  3069]]
====Iteration 24  ====
accuracy 0.470223791356
confusion matrix
[[ 1080 13288  1644]
 [ 1539 37071  4365]
 [ 1259 24422  3137]]
====Iteration 30  ====
accuracy 0.45325437048
confusion matrix
[[  981 12850  2181]
 [ 1491 35498  5986]
 [ 1186 24313  3319]]
====Iteration 36  ====
accuracy 0.473025454131
confusion matrix
[[ 1177 13542  1293]
 [ 1602 37751  3622]
 [ 1372 24840  2606]]
====Iteration 42  ====
accuracy 0.466949113936
confusion matrix
[[ 1187 13427  1398]
 [ 1866 37037  4071]
 [ 1428 24614  2776]]
====Iteration 48  ====
accuracy 0.468309738847
confusion matrix
```

```
[[ 1125 13411  1475]
 [ 1814 36824  4336]
 [ 1337 24311  3170]]
====Iteration 54  ====
accuracy 0.45729644773
confusion matrix
[[ 1132 13026  1853]
 [ 2095 35158  5721]
 [ 1376 23580  3862]]
```

In [26]:

```
#iterpret the weights
import sklearn
#iterate over the coefficients
print district_X.columns

from sklearn.linear_model import LogisticRegression



weights2 = d2lr_clf.coef_.T # take transpose to make a column vector
variable_names2 = district_X.columns
for coef, name in zip(weights2,variable_names2):
    print name, 'has weight of ', coef[0]

print d2lr_clf.coef_
```

```
Index([u'Category', u'violent', u'Time_Hour', u'Month', u'Time_Quarte
r',
       u'Day_of_Month', u'DayOfWeek_Num'],
      dtype='object')
Category has weight of  -0.00724356724258
violent has weight of  -0.214196394384
Time_Hour has weight of  0.000973212326384
Month has weight of  -0.000982455353872
Time_Quarter has weight of  -0.114609912687
Day_of_Month has weight of  6.84555742262e-06
DayOfWeek_Num has weight of  -0.00639276074001
[[ -7.24356724e-03  -2.14196394e-01   9.73212326e-04  -9.82455354e-04
   -1.14609913e-01   6.84555742e-06  -6.39276074e-03]
 [  6.34431776e-04   2.30198438e-01  -2.37031715e-03   4.89181507e-03
    5.23775535e-02   6.26232060e-04   1.06782793e-02]
 [  4.14877105e-03  -1.14084829e-01   1.96355131e-03  -5.02381716e-03
    1.69726011e-02  -7.62855041e-04  -7.88055276e-03]]
```

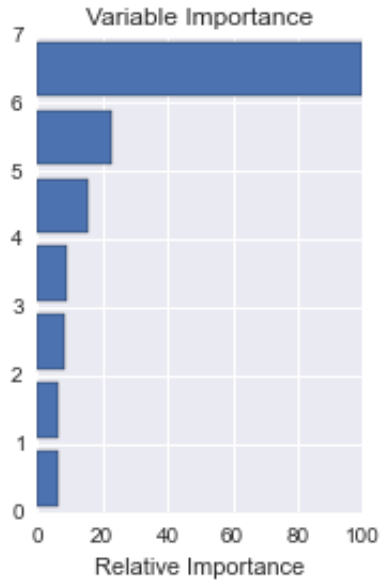
Our logistic Regression model for predicting districts was our strongest model. With consistent scores of greater than 46%, this model is definitely better than simply guessing which district. Additionally the correct predictions in the confusion matrix are almost directly equal to the incorrectly guessed classifiers. For example correctly predicting the 2nd district 74,357 times versus incorrectly predicting district 1 and 3 is 76,075.

Using Random Forest Classifier

In [37]:
```
from sklearn.ensemble import RandomForestClassifier

X = district_X.values
y = district_y.values

cv = StratifiedKFold(y, n_folds=10)

secclf = RandomForestClassifier(max_depth=5, n_estimators=5, n_jobs=-1,
oob_score=True, class_weight='auto')

# now iterate through and get predictions, saved to the correct row in y
hat
for train, test in cv:
    clf.fit(X[train],y[train])
    yhat[test] = clf.predict(X[test])

total_accuracy = mt.accuracy_score(y, yhat)
print 'Accuracy', total_accuracy

conf_rf = mt.confusion_matrix(y[test],yhat[test])
print "confusion matrix\n",conf_rf
```

```
Accuracy 0.415941479348
confusion matrix
[[ 5788  4635  5588]
 [10799 17579 14596]
 [ 8183  8116 12519]]
```

In [33]:
```
# Plot feature importance
feature_importance1 = clf.feature_importances_
# make importances relative to max importance
feature_importance1 = 100.0 * (feature_importance1 / feature_importance
1.max())
sorted_idx = np.argsort(feature_importance1)
pos = np.arange(sorted_idx.shape[0]) + .5
plt.subplot(1, 2, 2)
plt.barh(pos, feature_importance1[sorted_idx], align='center')
plt.xlabel('Relative Importance')
plt.title('Variable Importance')
plt.show()



clf.fit(X, y)
importances1 = clf.feature_importances_
std = np.std([tree.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances1)[::-1]

# Print the feature ranking
print("Feature ranking:")

for f in range(X.shape[1]):
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances1[indic
es[f]]))

# Plot the feature importances of the forest
plt.figure()
plt.title("Feature importances1")
plt.bar(range(X.shape[1]), importances1[indices],
       color="r", yerr=std[indices], align="center")
plt.xticks(range(X.shape[1]), indices)
plt.xlim([-1, X.shape[1]])
plt.show()
```




```
Feature ranking:
1. feature 0 (0.583963)
2. feature 4 (0.153129)
3. feature 2 (0.090763)
4. feature 5 (0.052260)
5. feature 1 (0.041035)
6. feature 3 (0.040777)
7. feature 6 (0.038072)
```

Using a Random Forest Classifier, we are able to classify districts with about 41 percent accuracy. At 41 percent accuracy, this classifier is better than random chance of 1/3.

Evaluation

Attribute Importance

When modeling violent vs. non-violent crimes the most important attributes for analysis are time, day, longitude (X), and latitude (Y). These values are the most important because they are the basis for all other values that will go into the model. Police Department District is important because it gives us insight into the proportions of crimes within a smaller region.When modeling the district, our important attributes are time, day, and category. Time and day allow us to place on frequencies of crimes at certain times on specific days during the week. Where as the category of crime, linked with the other attributes can give us information into the district of the incident, because each category has a different frequency for each region.

Deployment

The model will be tremendously useful for the San Francisco, Our model predicted the violent crimes correctly far greater than the non-violent crimes. Out of our three models we ran: Logistic Regression, Naïve-Bayes, and Random Forest Classifier; Naïve-Bayes ran the best but we have the most confidence in Random Forest Classifier. When we originally ran our algorithm, Random Forest performed the best with an accuracy of 61%, however after tweaking and optimizing our algorithm we ended with a score of 56%. We wanted to take into account the variation of the distribution of the variables being classified therefore we added a balance weight for our variables. When looking at the confusion matrix for Random Forest Classifier, the algorithm performed far greater for predicting Non-violent crime rather than predicting Violent Crime. Contrastingly, the Naïve-Bayes performed tremendously well for predicting violent crimes but fared poorly when predicting non-violent crimes. When looking into predicting the location.

Our model predicted the violent crimes correctly far greater than the non-violent crimes. Out of our three models we ran: Logistic Regression, Naïve-Bayes, and Random Forest Classifier; Naïve-Bayes ran the best but we have the most confidence in Random Forest Classifier. When we originally ran our algorithm, Random Forest performed the best with an accuracy of 61%, however after tweaking and optimizing our algorithm we ended with a score of 56%. We wanted to take into account the variation of the distribution of the variables being classified therefore we added a balance weight for our variables. When looking at the confusion matrix for Random Forest Classifier, the algorithm performed far greater for predicting Non-violent crime rather than predicting Violent Crime. Contrastingly, the Naïve-Bayes performed tremendously well for predicting violent crimes but fared poorly when predicting non-violent crimes. When looking into predicting the location of the district, our models were fairly similar as well. However, our Logistic model was the most accurate with an accuracy rating of 46%+. As a result, we can confidently say using our model is better than simply flipping a coin or guessing.

As a result we believe that this test would be readily usable for interested parties as it will accurately predict which side of town to deploy police to. Additionally, the algorithm is fairly simple in construction which allows for ability to scale for a greater number of districts thus leading to a more accurate location of the crime. We could measure the model's value by looking at deployment time of sending out police units when an initial call is made. With our current variables used it would be very helpful for a police officer to know if the situation is violent or non-violent when first responding. For looking at the location prediction it will be extremely beneficial from a preparation standpoint. It will allow for police units to deploy more patrol units in an area as well as optimize where backup units are placed. Gaining access to socio-economic and ethnic data will greatly enhance the data being used from an accountability standpoint. By gaining access to those two datapoints, there can be further analysis performed to look at the correlation between race, income, and crime. Additionally, accountability for police units will increase as there is a current strong correlation between disparity of arrests for races and socio-economic backgrounds. We believe the model should be updated on a bi-weekly or monthly basis to account for new tenants in an area as well as for distribution of paychecks.

Extra Work

Geospatial Plotting (Violent vs. Non-Violent)

In [ ]:
```
# re-import a clean version of the data set
train_data=pd.read_csv('sf_train.csv')

violent_crime=['EXTORTION', 'VANDALISM', 'LARCENY/THEFT', 'ROBBERY', 'DI
SORDERLY CONDUCT', 'VEHICLE THEFT', 'ASSAULT',
               'KIDNAPPING','ARSON', 'FAMILY OFFENSES', 'SEX OFFENSES NO
N FORCIBL', 'WEAPON LAWS', 'DRUNKENNESS',
               'SUICIDE', 'DRUG/NARCOTIC', 'SEX OFFENSES FORCIBLE', 'WAR
RANTS', 'BURGLARY']

non_violent_crime=['FRAUD', 'FORGERY/COUNTERFEITING', 'BAD CHECKS', 'EMB
EZZLEMENT', 'SUSPICIOUS OCC', 'BRIBERY',
                   'STOLEN PROPERTY', 'DRIVING UNDER THE INFLUENCE', 'L
IQUOR LAWS', 'TRESPASS', 'RECOVERED VEHICLE',
                   'MISSING PERSON', 'RUNAWAY', 'PORNOGRAPHY/OBSCENE MA
T', 'LOITERING', 'SECONDARY CODES', 'GAMBLING',
                   'PROSTITUTION', 'OTHER OFFENSES', 'NON-CRIMINAL', 'T
REA']

train_data['violent'] = [1 if x in violent_crime else 0 for x in train_d
ata['Category']]

# loading a map of SF from openstreetmaps
SF_map= np.loadtxt("map1.txt")

# Map bounding box:
#    ll.lon     ll.lat   ur.lon     ur.lat
#    -122.52469 37.69862 -122.33663 37.82986
lon_lat_box = (-122.5247, -122.3366, 37.699, 37.8299)
asp = SF_map.shape[0] * 1.0 / SF_map.shape[1]

# Plot the image of SF Map
fig = plt.figure(figsize=(16,16))
plt.imshow(SF_map, cmap='gray', extent=lon_lat_box, aspect=1/asp)
ax=plt.gca()

# Plot Violent vs. Non-Violent data
# train_data[(train_data['Y']<38)].plot(x='X', y='Y', ax=ax, kind='scatt
er', marker='o', s=4, color='blue', alpha=0.01)
train_data[(train_data['violent']==0)].plot(x='X', y='Y', ax=ax, kind='s
catter', marker='o', s=4, color='blue', alpha=0.01)
train_data[(train_data['violent']==1)].plot(x='X', y='Y', ax=ax, kind='s
catter', marker='o', s=4, color='red', alpha=0.01)
ax.set_axis_off()
```

Visualization for weekdays

For my going further I decided to take a dive into the spread of crime over the days, as well as visualizing the spread of crime per charge. I wanted to gain some more experience using the visually impressive seaborn library and decided to do some simple barcharts for analysis.

In [ ]:
```
dailycrime = pd.read_csv("sf_train.csv")
#dailycrime.head(5)

#Plotting Crimes x the weekday
plt.figure(figsize=(20,12))
sns.set_context("poster",)


#Converting date into datetime
dailycrime["Dates"] = pd.to_datetime(dailycrime["Dates"])
#sum of crimes per day
colnames = dailycrime.columns.values
colnames = colnames[1:(len(colnames)-2)]
colnames

ax1 =  plt.subplot2grid((2,1),(0,0))
ax1.set_title('Daily Amount of Crimes',color='black',size='20')
weekdata = pd.DataFrame(dailycrime["DayOfWeek"].value_counts())
sns.barplot(x=weekdata.index, y="DayOfWeek", data=weekdata, palette="Set
1", order=['Sunday','Monday','Tuesday',

'Wednesday','Thursday','Friday','Saturday'])
sns.axlabel("","")

ax2 =  plt.subplot2grid((2,1),(1,0))
ax2.set_title('Daily Look At Crime Per District',color='black',size='2
0')
sns.countplot(x="DayOfWeek", hue="PdDistrict", data=dailycrime, palette
="muted", order=['Sunday','Monday','Tuesday',

'Wednesday','Thursday','Friday','Saturday']);
sns.axlabel("","")
ax2.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.,fontsize=1
7)
```

In [ ]:
```
crimetype = pd.DataFrame(dailycrime["Category"].value_counts())

fig = plt.figure()
plt.figure(figsize=(15,10))
sns.set_context("poster")

ax1 =  plt.subplot2grid((1,2),(0,0))
ax1.set_title('Top 10 crime Categories',color='black',size='20')
sns.barplot(x=crimetype[0:10].index, y="Category", data=crimetype[0:10],
palette = "RdBu")
sns.axlabel("","")
for label in ax1.xaxis.get_ticklabels():
        label.set_rotation(90)

ax2 =  plt.subplot2grid((1,2),(0,1))
ax2.set_title('Bottom 10 crime Categories',color='black',size='20')
sns.axlabel("","")
sns.barplot(x=crimetype[len(crimetype)-10:len(crimetype)].index, y="Cate
gory", data=crimetype[(len(crimetype)-10):len(crimetype)])
for label in ax2.xaxis.get_ticklabels():
        label.set_rotation(90)
```

Statistical Analysis

The models all performed statistically similar as there was no difference detected via a 95% confidence interval

MNB:

ACC* | *Precision* | *Recall* | *F-Measure

0.576545488 | 0.412437549 | 0.181765507 | 0.252327616

Bnb:

ACC* | *Precision* | *Recall* | *F-Measure

0.603024919 |0.419093079 | 0.025436741 | 0.047962417

Logistic :

ACC | precision | recall | F-measure

0.558526752 | 0.541398691 | 0.448993321 | 0.490885211

RandomForest:

ACC | precision | recall | F-measure

0.556899458 | 0.619984356 | 0.453495518 | 0.523829339

Confidence Interval

Comparing the Mnb vs Bnb

(0.031078916, 0.021879947)

Comparing the Mnb vs Logistic

(-0.02170561, -0.03098125)

Comparing the Mnb vs RandomForest

(-0.015011594, -0.024280466)

Comparing the Bnb vs Logistic

(-0.048207524, -0.057438199)

Comparing the Bnb vs RandomForest

(-0.041513525, -0.050737398)

Comparing the Logistic vs Random Forest

(0.01134757, 0.00204723)

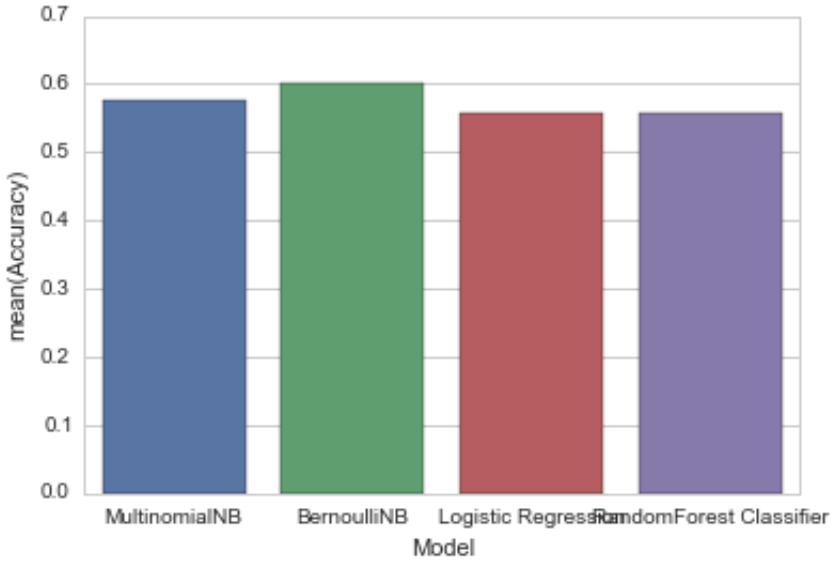
Visual Comparison of Model Performance

In [118]:

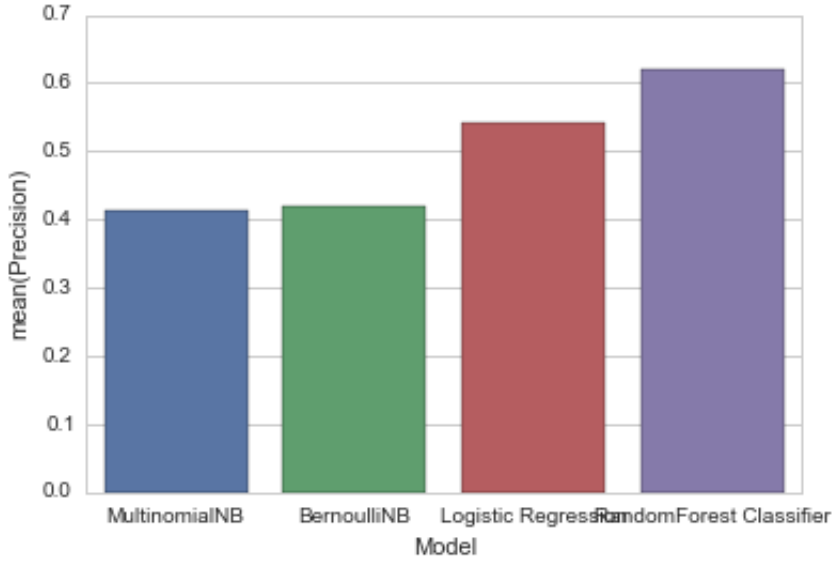
```
import seaborn as sns

#help(sns.barplot)
# Create our dataframes
ratings = pd.read_csv('accuracy.csv')

ratings.info()

# Plot to show the number of crimes on our 3 District grouping (W, NE, S
E)

#model accuracy comparison
#sns.set_style("whitegrid")


#plt.figure(figsize=((15,5))

ax = sns.barplot(x=ratings.Model, y=ratings.Accuracy)

#plt.subplot(2,2,1)
#plt.title ('Model Accuracy Comparison')
#plt.bar(x=ratings.Model, y=ratings.Accuracy, width=.15)
#plt.grid()

#model precision comparison

#ax1 = sns.barplot(x=ratings.Model, y=ratings.Precision)


#plt.subplot(2,2,1)plt.title('Shots Made by Home/Away')plt.scatter(h1.lo
c_x, h1.loc_y, c=h1.home_game, cmap=plt.cm.rainbow, s=20, linewidths=0)p
lt.grid()



#model recall comparison
#ax2 = sns.barplot(x=ratings.Model, y=ratings.Recall)



#model F-measure comparison
#ax3 = sns.barplot(x=ratings.Model, y=ratings.FMeasure)



#plt.barplot(x="Models", y="Accuracy Score", data = ratings.Accuracy)


#ratings.Accuracy.value(sort=False).plot(kind='bar', figsize=(8,10))
#plt.xlabel('Accuracy For Models')
#plt.ylabel('Accuracy Score')
#sns.despine()
#plt.show()
```

```
# Plot to show the number of crimes on our 3 District grouping (W, NE, S
E)

#train_users.district.value_counts(sort=False).plot(kind='bar', figsize=
(8,10))
#plt.xlabel('district')
#plt.ylabel('Count')
#sns.despine()
#plt.show()
```

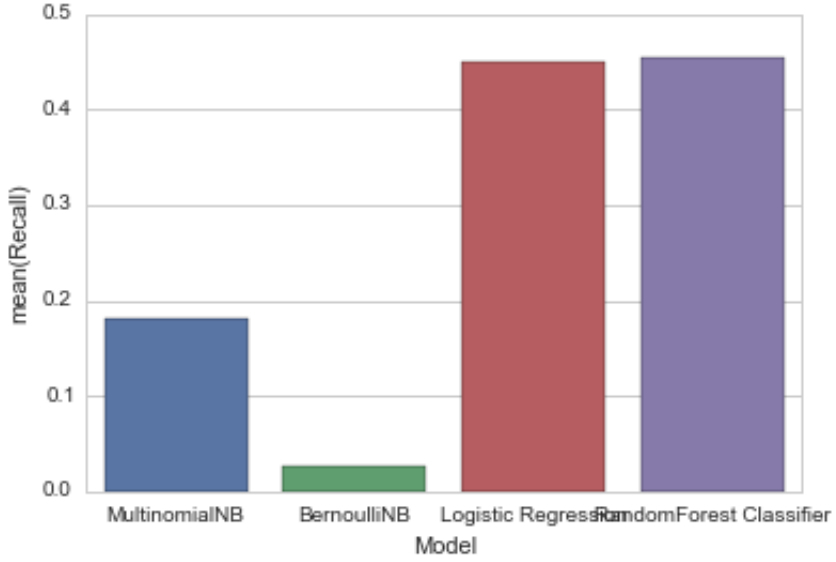
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 4 entries, 0 to 3
Data columns (total 5 columns):
Model        4 non-null object
Accuracy     4 non-null float64
Precision    4 non-null float64
Recall       4 non-null float64
FMeasure     4 non-null float64
dtypes: float64(4), object(1)
memory usage: 192.0+ bytes
```



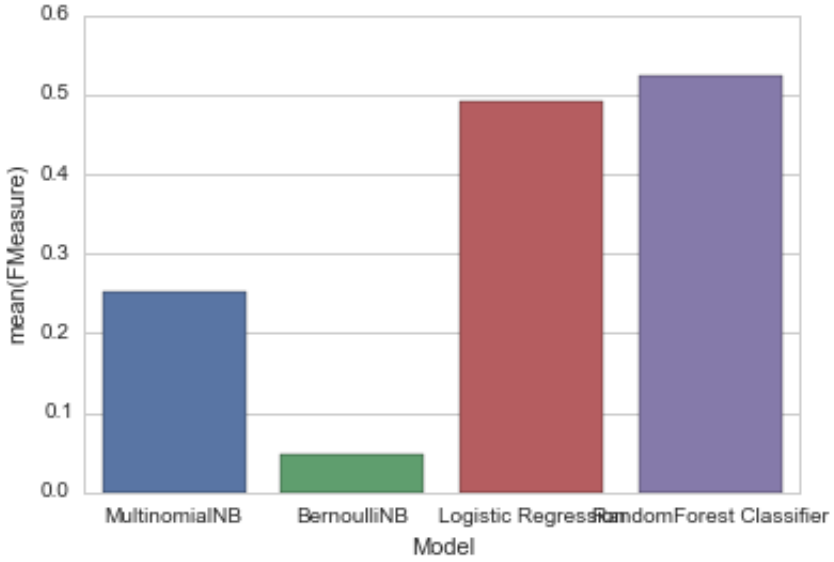
In [115]:
```
ax = sns.barplot(x=ratings.Model, y=ratings.Precision)
```



In [116]:
```
ax2 = sns.barplot(x=ratings.Model, y=ratings.Recall)
```



In [117]:
```
ax3 = sns.barplot(x=ratings.Model, y=ratings.FMeasure)
```



Advantages of each model (statistical analysis)

After performing a statistical analysis of a 95% confident test on the 4 models with a null hypothesis that the models all performed the same (H(0) = No difference between the 4 models). We found that after running the confidence tests that there was no statistical difference between the 4 models as none of the confidence intervals contained a 0 value.

Feature Importance

We tested for feature importance by looking into the features of the Random Forest Classifier and the weights and coefficients for the Logistic Regression.

As seen in the graphs above for the feature analysis the Random Forest Classifier for crime type held only a few features of high importance: Feature 2 (32%), 1 (20%), 3 (19%), and 0 (13%)

The Random Forest Classifier for District held fewer features in importance. Primarily feature 0 with 58% and feature 4 with 15%.

The Roc AUC allowed us additional insight into the features for the Random Classifiers as it showed that the features importance are similar to the logistic regression weights. For example the top 5 roc scores for Random Forest Classifier are 'Time Quarter', 'PdDistrict_NORTHERN', 'PdDistrict_INGLESIDE', 'Time_Hour', and 'PdDistrict_TENDERLOIN'

The Logistic Regression for crime type additionally held a few weights with high values: The PdDistrict_CENTRAL, PdDistrict_TARAVAL and PdDistrict_TENDERLOIN

The Logistic Regression for districs held the majority of the weight value in Violent crime, and day of the month.

In [ ]: