aug_5

September 5, 2019

[1]:
```
# src: https://www.kaggle.com/c/sf-crime
```

[2]:
```
# From kaggle 'Data Description' section:

# This dataset contains incidents derived from SFPD Crime
# Incident Reporting system.

# The data ranges from 1/1/2003 to 5/13/2015.

# The training set and test set rotate every week,
# meaning week 1,3,5,7... belong to test set,
# week 2,4,6,8 belong to training set.

# Data fields
# Dates - timestamp of the crime incident
# Category - category of the crime incident (only in train.csv).
    # This is the target variable you are going to predict.TRwe
# Descript - detailed description of the crime incident (only in train.csv)
# DayOfWeek - the day of the week
# PdDistrict - name of the Police Department District
# Resolution - how the crime incident was resolved (only in train.csv)
# Address - the approximate street address of the crime incident
# X - Longitude
# Y - Latitude
```

[3]:
```
# Load libraries

import IPython

import matplotlib
%matplotlib inline
import matplotlib.pyplot as plt

import gmplot

import numpy as np
```

```python
import pandas as pd
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)

from scipy.stats import kstest, probplot

from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import log_loss

from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier

from lightgbm import LGBMClassifier
```

[4]:
```python
# Load the data

TRAIN_DF_PATH = 'data/train.csv'
TEST_DF_PATH = 'data/test.csv'

raw_train_df = pd.read_csv(TRAIN_DF_PATH, header=0)
raw_test_df = pd.read_csv(TEST_DF_PATH, header=0)

raw_concat_traintest_df = pd.concat(
    [raw_train_df, raw_test_df],
    ignore_index=True, sort=False
)
```

[5]:
```python
def overview_df(dataset_df):
    # Elements in dataset
    display(dataset_df.sample(5))
    # Dataset shape
    display(dataset_df.shape)
    # Columns and dtypes
    display(dataset_df.dtypes)
    # .describe method
    display(dataset_df.describe(include='all').T)
    # Empty columns
    display(dataset_df.isnull().sum())
```

[6]:
```python
# overview_df(raw_train_df)

# overview_df(raw_test_df)

# overview_df(raw_concat_traintest_df)
```

[7]:
```python
# Overview rows with unusual Longtitude and Latitude
```

```
# "Unusual" means incorrect latitude/longtitude range
# Latitudes range: [-90;+90]. Longtitudes range: [-180;+180].

# Note: all the rows have the same feature values: X=-120.5, Y=90.0.
# Note: there is same type of invalid rows in both train and test sets.

# Note: using overview_col_name_occurences_fulldf_subsetdf function: some␣
↪"invalid" rows have
    # valid X,Y coordinates in the training set.
# Note: using overview_col_name_occurences_fulldf_subsetdf, it is better to use␣
↪concat df


def overview_invalid_long_lat(dataset_df):
    # Look for invalid rows
    invalid_long_rows = dataset_df[
        (dataset_df['X'] <= -180) | (dataset_df['X'] >= 0)
    ]
    invalid_lat_rows = dataset_df[
        (dataset_df['Y'] >= 90) | (dataset_df['Y'] <= 0)
    ]
    # Review amount of rows with invalid longtitude / latitude values
    display("Found longtitude invalid values: {0}".format(invalid_long_rows.
↪shape))
    display("Found latitude invalid values: {0}".format(invalid_lat_rows.shape))


def overview_col_name_occurences_fulldf_subsetdf(full_dataset_df, subset_df,␣
↪col_name):
    # Remove subset from the full dataset to omit using subset_df values.
    # It is expected that subset_df is in full_dataset_df.
    tosearch_df = full_dataset_df.drop(subset_df.index)
    # Iterate through subset_df and find which rows in full_dataset_df have the␣
↪same value in col_name.
    for subset_col_name_val in subset_df[col_name].sort_values():
        occurences = tosearch_df[ tosearch_df[col_name] == subset_col_name_val ]
        if occurences.shape[0] != 0:
            display(
                'value "{0}" from subset has {1} occurences in full_dataset'.
↪format(
                    subset_col_name_val, occurences.shape[0]
                )
            )
#             display(occurences)
```

[8]:
```
display("Incorrect coords: train")
overview_invalid_long_lat(raw_train_df)
```

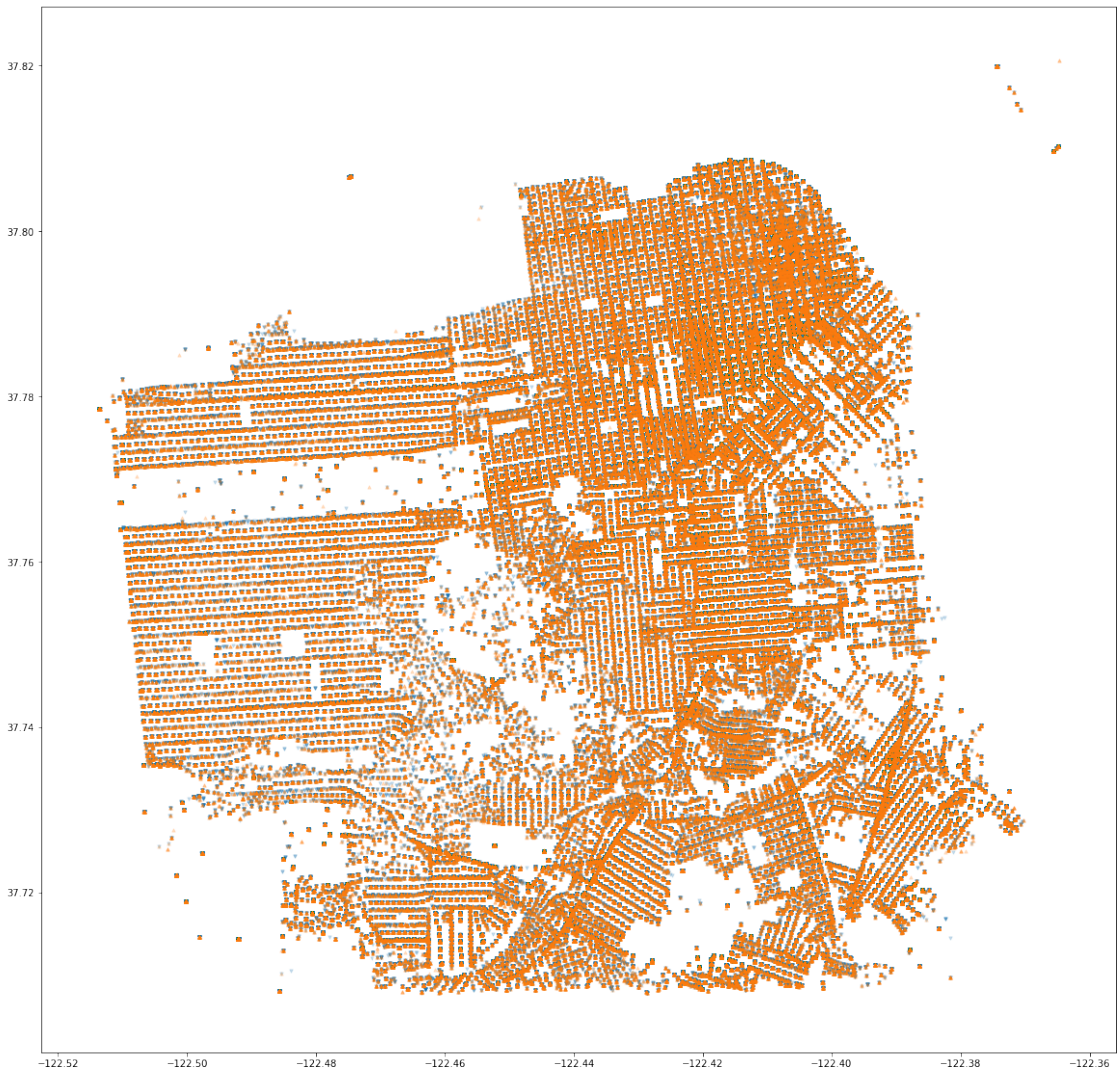
```python
display("Incorrect coords: test")
overview_invalid_long_lat(raw_test_df)

# display("Incorrect coords: concat train_test")
# overview_invalid_long_lat(raw_concat_traintest_df)

display("Occurences: train")
overview_col_name_occurences_fulldf_subsetdf(
    raw_train_df, raw_train_df[ raw_train_df['Y'] == 90.0 ],
    'Address'
)

display("Occurences: test")
overview_col_name_occurences_fulldf_subsetdf(
    raw_test_df, raw_test_df[ raw_test_df['Y'] == 90.0 ],
    'Address'
)

# display("Occurences: concat")
# overview_col_name_occurences_fulldf_subsetdf(
#     raw_concat_traintest_df, raw_concat_traintest_df[␣
↪raw_concat_traintest_df['Y'] == 90.0 ],
#     'Address'
# )
```

```
'Incorrect coords: train'

'Found longtitude invalid values: (0, 9)'

'Found latitude invalid values: (67, 9)'

'Incorrect coords: test'

'Found longtitude invalid values: (0, 7)'

'Found latitude invalid values: (76, 7)'

'Occurences: train'

'value "BRYANT ST / SPEAR ST" from subset has 1 occurences in full_dataset'
```

```
'value "I-280 / CESAR CHAVEZ ST" from subset has 1 occurences in full_dataset'

'value "I-280 / PENNSYLVANIA AV" from subset has 1 occurences in full_dataset'

'value "JAMES LICK FREEWAY HY / CESAR CHAVEZ ST" from subset has 1 occurences in full_dataset'

'value "JAMES LICK FREEWAY HY / CESAR CHAVEZ ST" from subset has 1 occurences in full_dataset'

'Occurences: test'

'value "INTERSTATE280 HY / OCEAN AV" from subset has 1 occurences in full_dataset'

'value "JAMES LICK FREEWAY HY / BAY SHORE BL" from subset has 1 occurences in full_dataset'

'value "JAMES LICK FREEWAY HY / CESAR CHAVEZ ST" from subset has 1 occurences in full_dataset'

'value "JAMES LICK FREEWAY HY / CESAR CHAVEZ ST" from subset has 1 occurences in full_dataset'

'value "SPEAR ST / THE EMBARCADERO SOUTH ST" from subset has 1 occurences in full_dataset'

'value "SPEAR ST / THE EMBARCADERO SOUTH ST" from subset has 1 occurences in full_dataset'
```

[9]:

```
# Plot coordinates separately from train/test sets

train_no_invalid = raw_train_df[ raw_train_df['Y'] != 90.0 ]
test_no_invalid = raw_test_df[ raw_test_df['Y'] != 90.0 ]

fig = plt.figure(figsize=(20, 20))
plt.scatter(
    x=train_no_invalid['X'], y=train_no_invalid['Y'],
    s=10, marker='v', alpha=0.2
)
plt.scatter(
    x=test_no_invalid['X'], y=test_no_invalid['Y'],
    s=10, marker='^', alpha=0.2
)

plt.show()
```



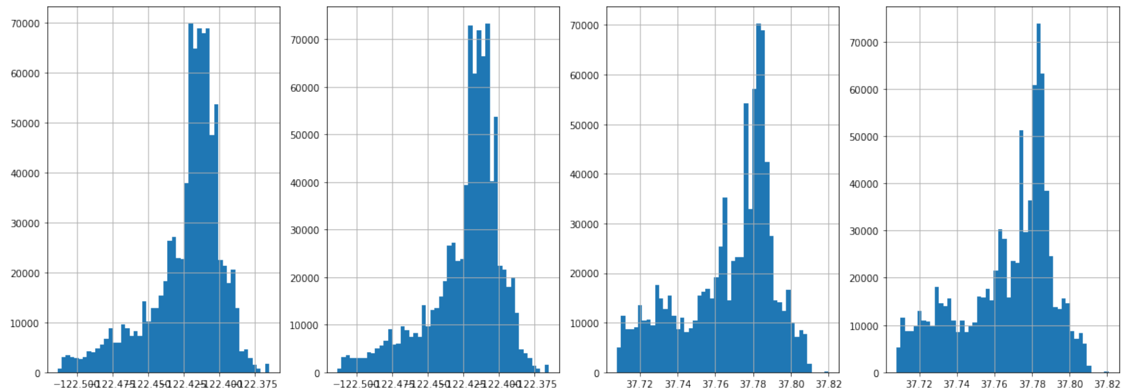
[10]:
```
# Generate geographical heatmaps
# Loading full train/test sets requires too much memory - plot subset of all␣
↪coords

# Train set
train_lon, train_lat = train_no_invalid['X'], train_no_invalid['Y']

train_gmap = gmplot.GoogleMapPlotter(train_lat[0], train_lon[0], 12)
train_gmap.heatmap(train_lat[:50000], train_lon[:50000])
train_gmap.draw('train_50k_rows_heatmap.html')

train_gmap_html_iframe = IPython.display.IFrame(src='train_50k_rows_heatmap.
↪html', width=800, height=400)
display(train_gmap_html_iframe)
```

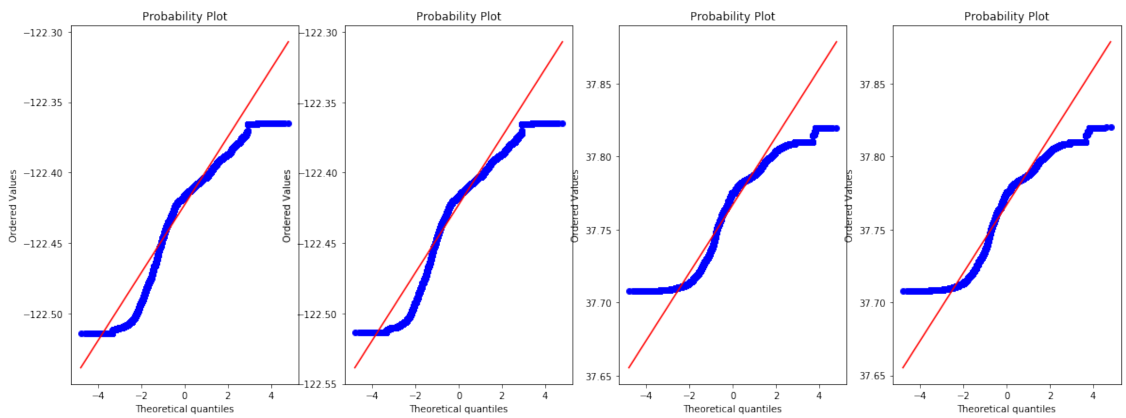
```
# Test set
test_lon, test_lat = test_no_invalid['X'], test_no_invalid['Y']

test_gmap = gmplot.GoogleMapPlotter(train_lat[0], train_lon[0], 12)  # not␣
 ↪test_*[0] to compare pics
test_gmap.heatmap(test_lat[:50000], test_lon[:50000])
test_gmap.draw('test_50k_rows_heatmap.html')

test_gmap_html_iframe = IPython.display.IFrame(src='test_50k_rows_heatmap.
 ↪html', width=800, height=400)
display(test_gmap_html_iframe)
```

```
<IPython.lib.display.IFrame at 0x7f38895d7c50>
```

```
<IPython.lib.display.IFrame at 0x7f38d010f3c8>
```

[11]:
```
# Note: in train and test datasets, X and Y coordinates are almost the same
fig, [ax_0, ax_1, ax_2, ax_3] = plt.subplots(1, 4, figsize=(20, 7))
train_no_invalid['X'].hist(ax=ax_0, bins=50)
test_no_invalid['X'].hist(ax=ax_1, bins=50)
train_no_invalid['Y'].hist(ax=ax_2, bins=50)
test_no_invalid['Y'].hist(ax=ax_3, bins=50)
plt.show()

# Note: X and Y are somehow "normally" distributed (kstat tells they are not),␣
 ↪left skewed distribution.
fig, [ax_0, ax_1, ax_2, ax_3] = plt.subplots(1, 4, figsize=(20, 7))
probplot(train_no_invalid['X'], plot=ax_0)
probplot(test_no_invalid['X'], plot=ax_1)
probplot(train_no_invalid['Y'], plot=ax_2)
probplot(test_no_invalid['Y'], plot=ax_3)
plt.show()

# Kolmogorov-Smirnov
# The null-hypothesis for the KT test is that the distributions are the same
# Thus, the lower your p value -> conclude the distributions are different
display( kstest(train_no_invalid['X'], 'norm') )  # p=0
display( kstest(test_no_invalid['Y'], 'norm') )  # p=0
display( kstest(train_no_invalid['X'], 'norm') )  # p=0
display( kstest(test_no_invalid['Y'], 'norm') )  # p=0
```






```
KstestResult(statistic=1.0, pvalue=0.0)
```

```
KstestResult(statistic=1.0, pvalue=0.0)
```

```
KstestResult(statistic=1.0, pvalue=0.0)
```

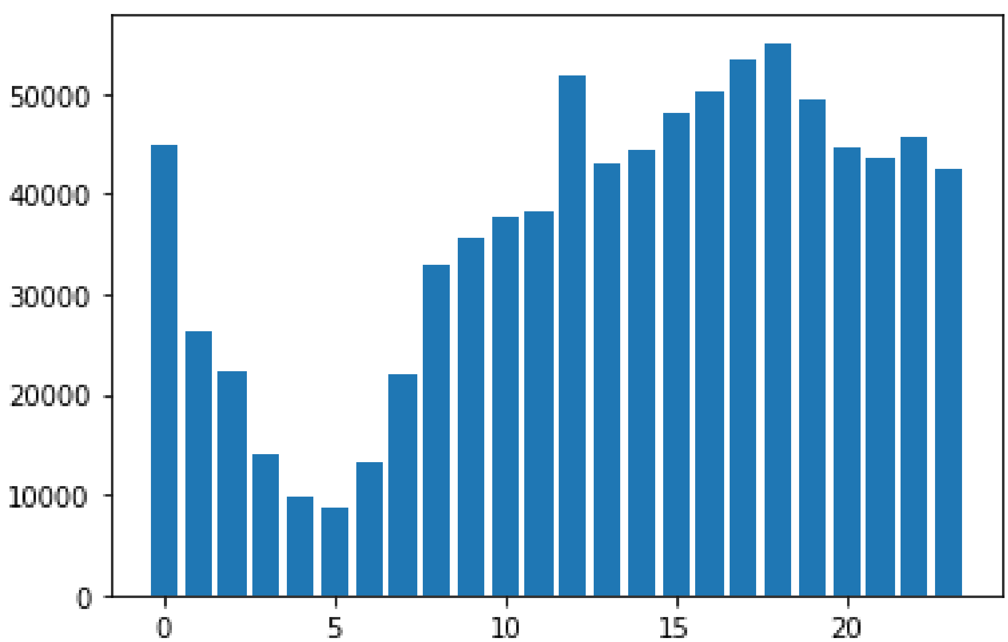
```
KstestResult(statistic=1.0, pvalue=0.0)
```

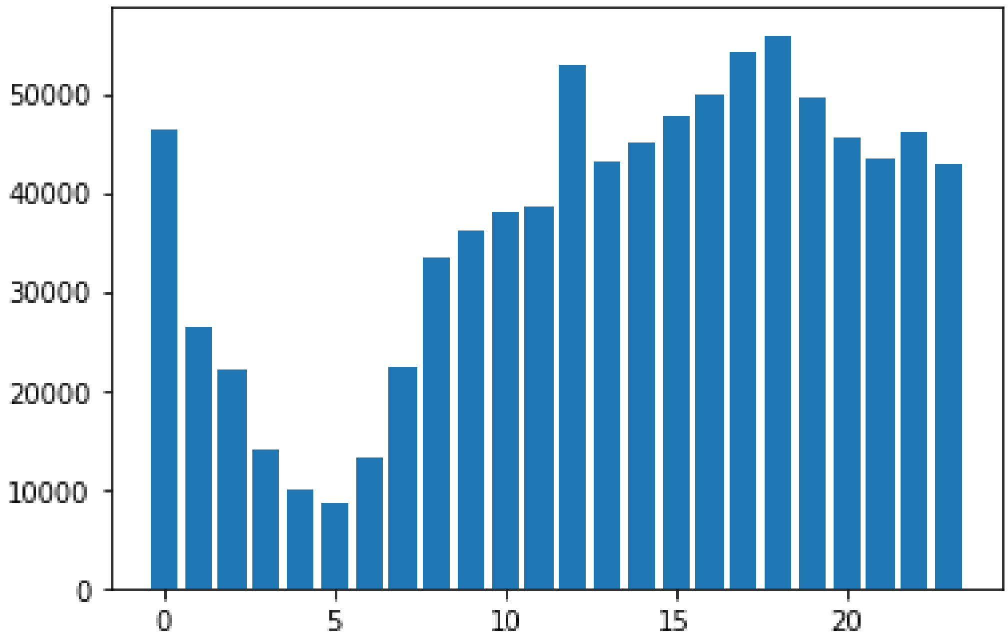
[12]:

```
# Explore 'Dates' feature

def hist_by_groupby_valuecounts(dataset_df, col_name_to_groupby):
    col_valuecounts = dataset_df.groupby(by=col_name_to_groupby).size()
    plt.bar(col_valuecounts.index, col_valuecounts); plt.show()
```

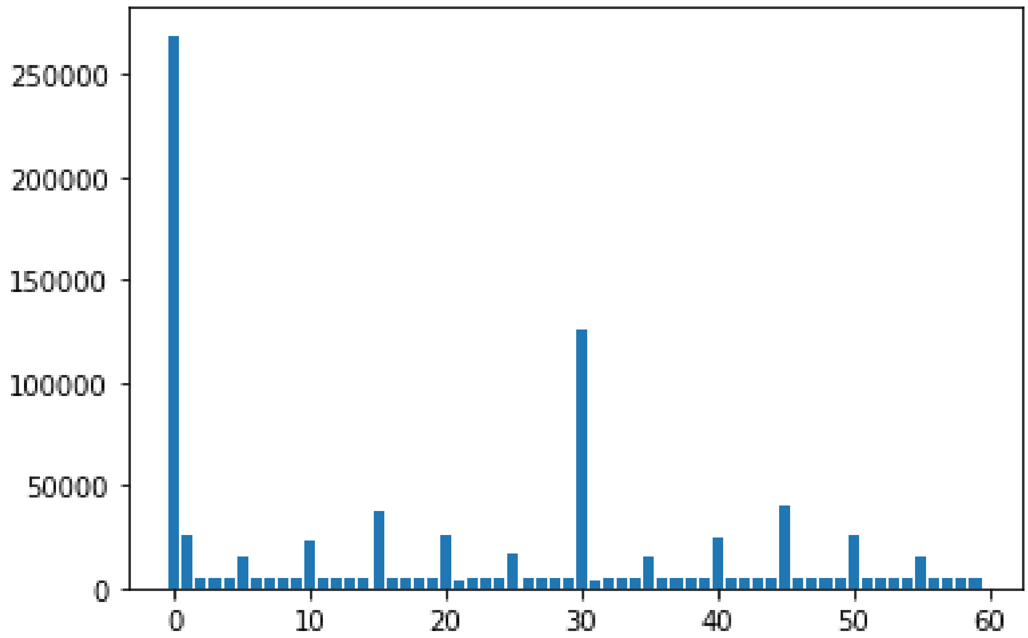
[13]:
```
# Intermediate arrays: exploring 'Dates' feature

train_eda_dates = raw_train_df.copy()
train_eda_dates['Dates'] = pd.to_datetime(train_eda_dates['Dates'])

test_eda_dates = raw_test_df.copy()
test_eda_dates['Dates'] = pd.to_datetime(test_eda_dates['Dates'])
```

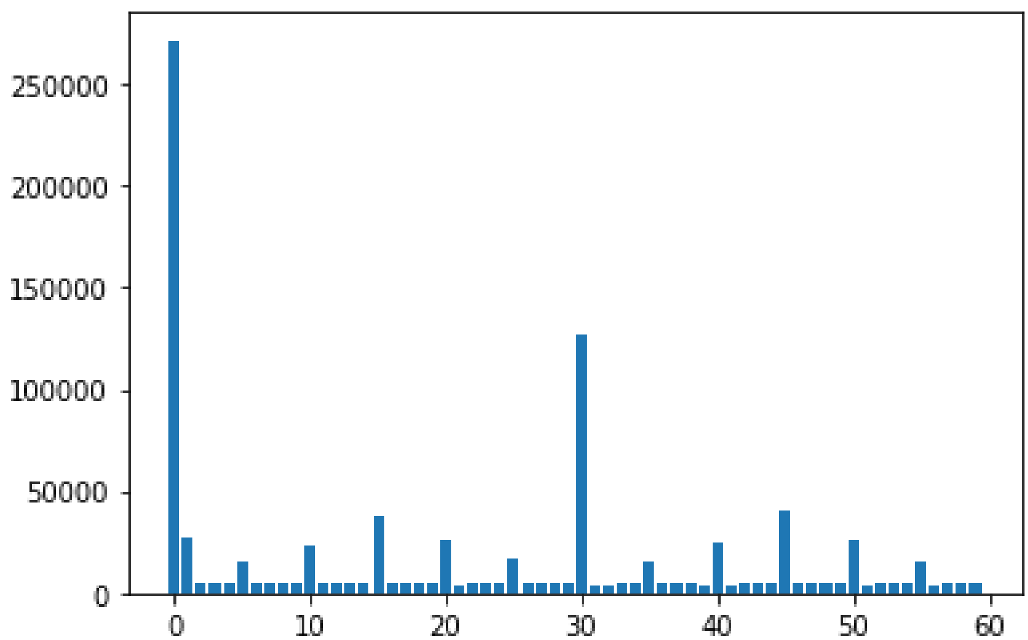
[14]:
```
# 1. Hours

# Conclusion: 'Hour' looks like a good feature

# Training set
train_eda_dates['Hour'] = train_eda_dates['Dates'].dt.hour

hist_by_groupby_valuecounts(train_eda_dates, 'Hour')

# Test set
test_eda_dates['Hour'] = test_eda_dates['Dates'].dt.hour

hist_by_groupby_valuecounts(test_eda_dates, 'Hour')
```





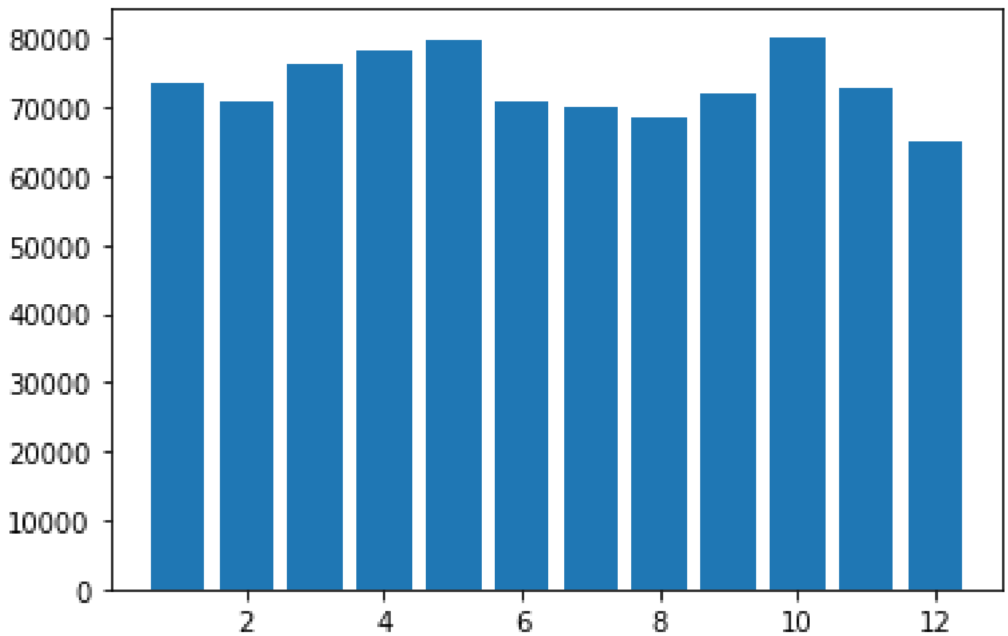
[15]:
```
# 2. Minutes

# Note: Minutes are rigged because of lot of 0, 15, 30, 45, 60 values in␣
↪reports -
# this might be because to human factor.

# Conclusion: don't use 'Minutes' as a feature

# Training set
train_eda_dates['Minutes'] = train_eda_dates['Dates'].dt.minute

hist_by_groupby_valuecounts(train_eda_dates, 'Minutes')

# Test set
test_eda_dates['Minutes'] = test_eda_dates['Dates'].dt.minute

hist_by_groupby_valuecounts(test_eda_dates, 'Minutes')
```

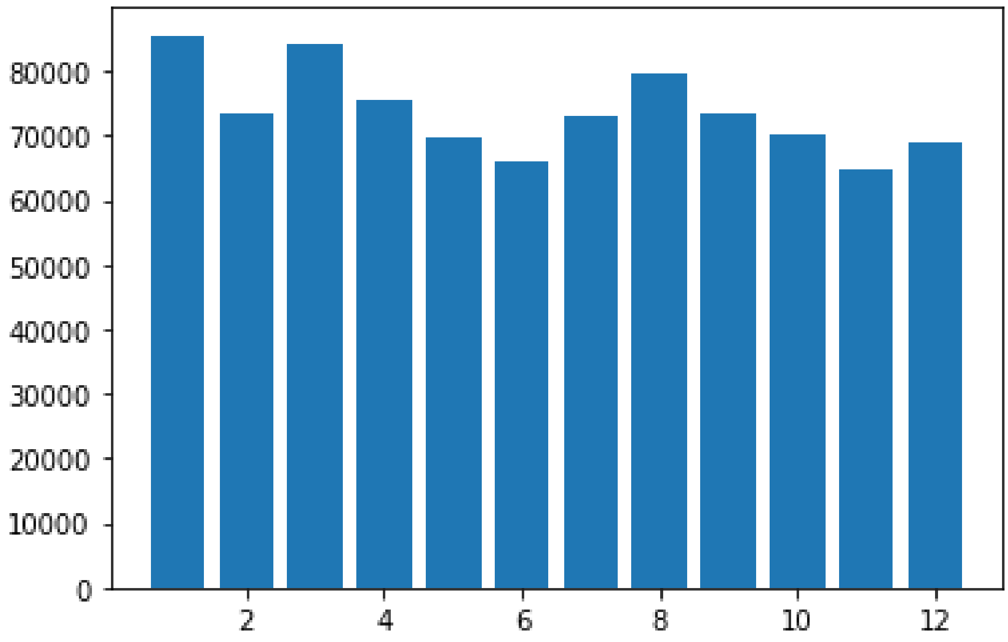




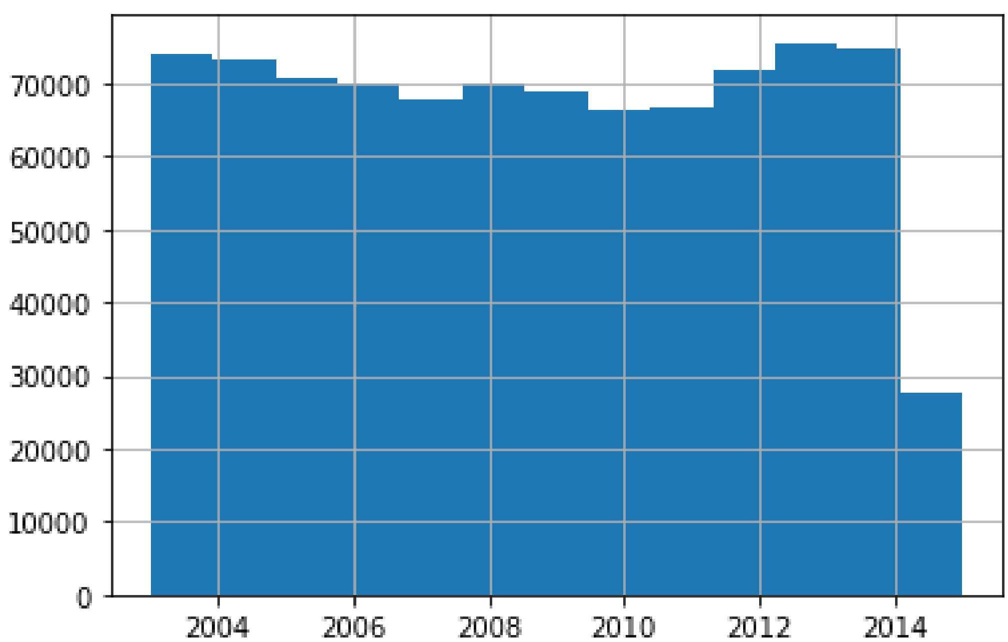
[16]:
```
# 3. Month

# Conclusion: try out Month as a feature

# Training set
```

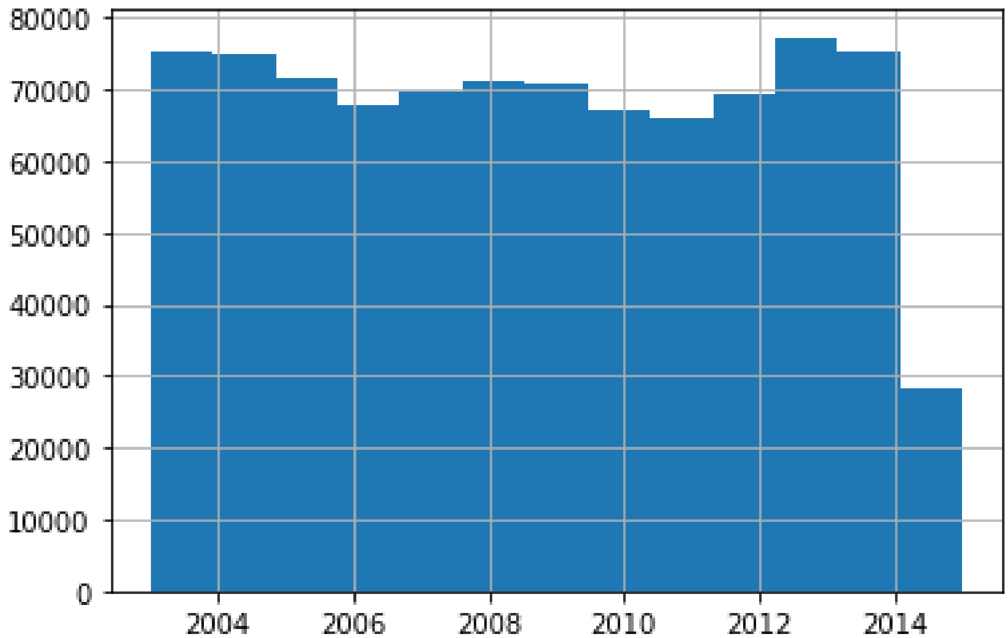
```python
train_eda_dates['Month'] = train_eda_dates['Dates'].dt.month

hist_by_groupby_valuecounts(train_eda_dates, 'Month')

# Test set
test_eda_dates['Month'] = test_eda_dates['Dates'].dt.month

hist_by_groupby_valuecounts(test_eda_dates, 'Month')
```



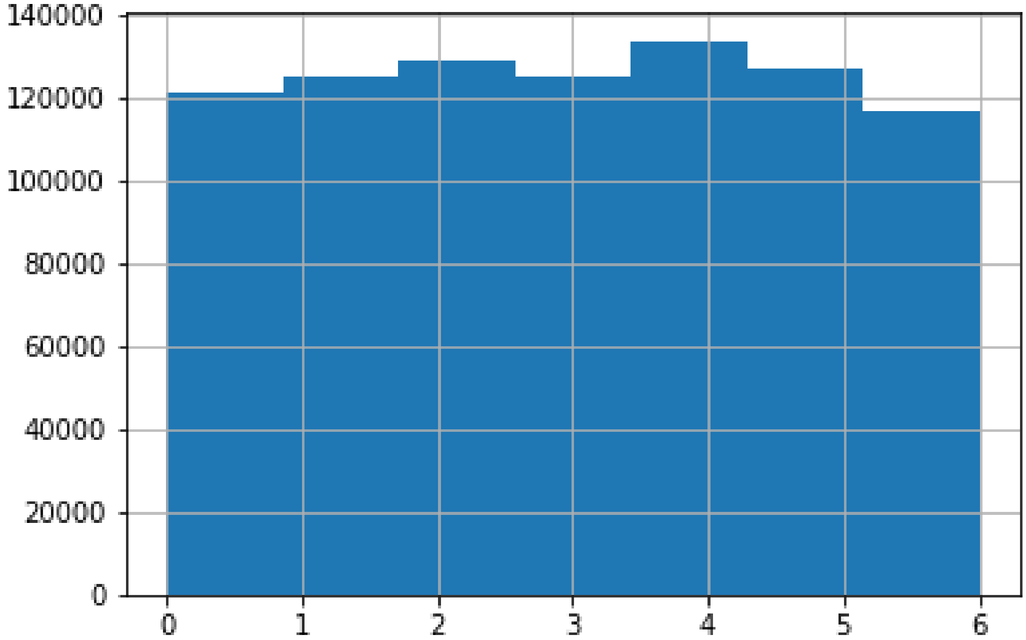


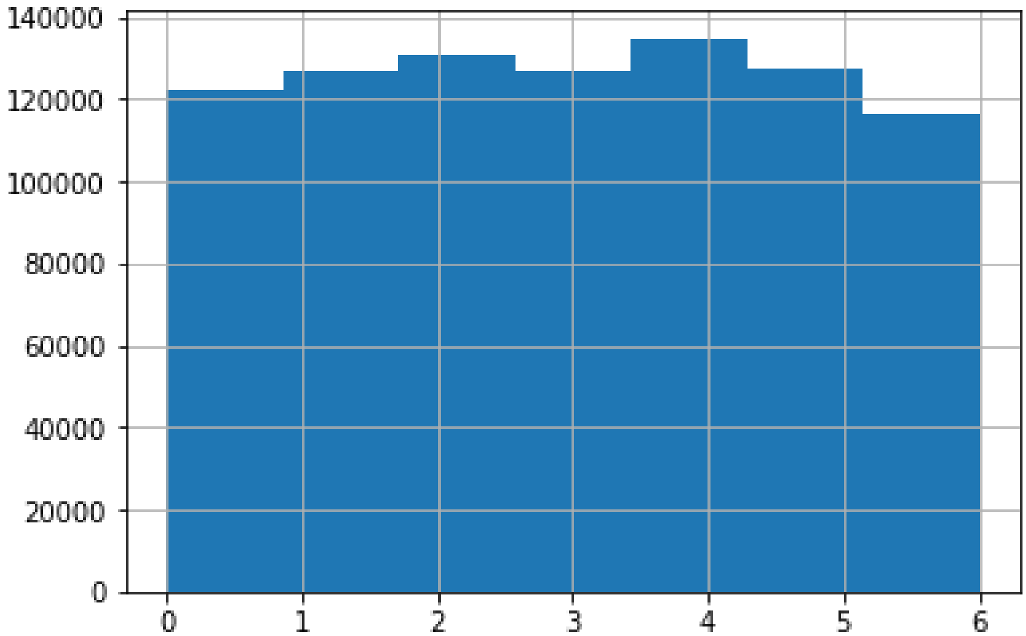
[17]:
```
# 4. Year

# Conclusion: try out Year as a feature in baseline model

train_eda_dates['Dates'].dt.year.hist(bins=13); plt.show()

test_eda_dates['Dates'].dt.year.hist(bins=13); plt.show()
```





[18]:
```
# 5. Day of the week

# Note: feature 'DayOfWeek' already exists in both training and test sets as a␣
↪str.
# Try to replace it with a numerical value.

# Conclusion:
# Try out IsWednesday/IsFriday/IsSaturday features (highest crime rate)
# Tru out IsSunday feature (lowest crime rate)

train_eda_dates['Dates'].dt.dayofweek.hist(bins=7); plt.show()

test_eda_dates['Dates'].dt.dayofweek.hist(bins=7); plt.show()
```

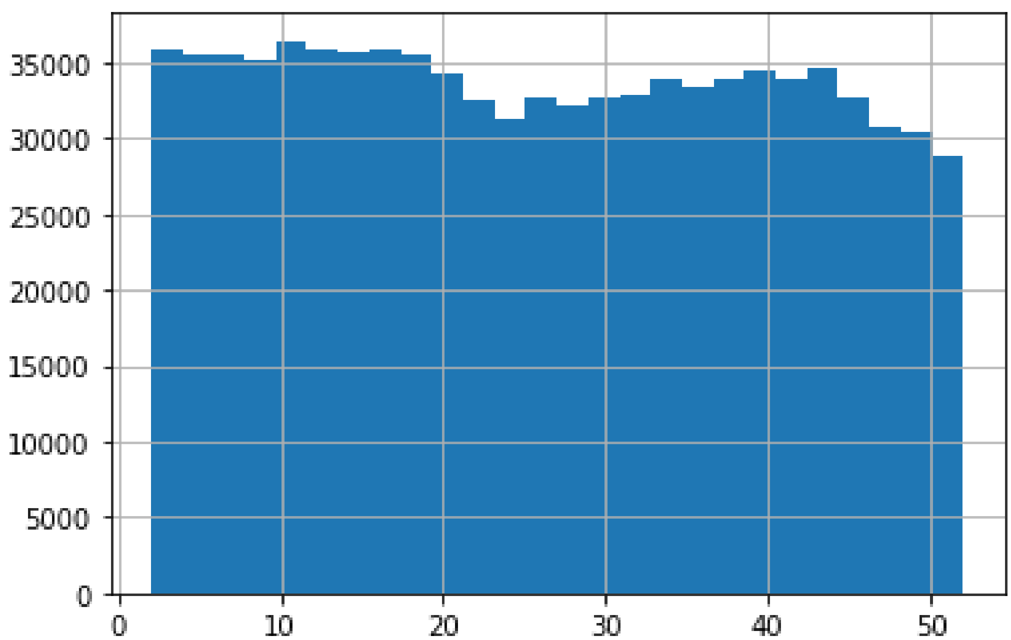


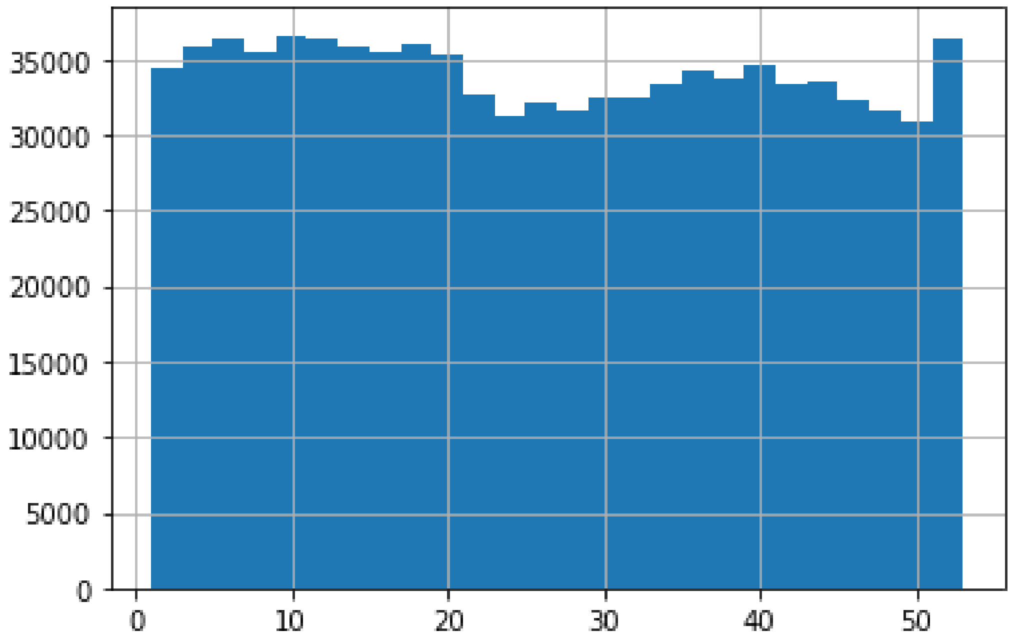


[19]:
```
# 6. Week of year

# Note: test_eda_dates : last week of year has a spike while train_eda_dates␣
↪does not
```

```
# Conclusion: might be a feature, try it out

display(
    len(
        pd.unique(train_eda_dates['Dates'].dt.weekofyear)
    )
)

train_eda_dates['Dates'].dt.weekofyear.hist(bins=26); plt.show()

test_eda_dates['Dates'].dt.weekofyear.hist(bins=26); plt.show()
```

26

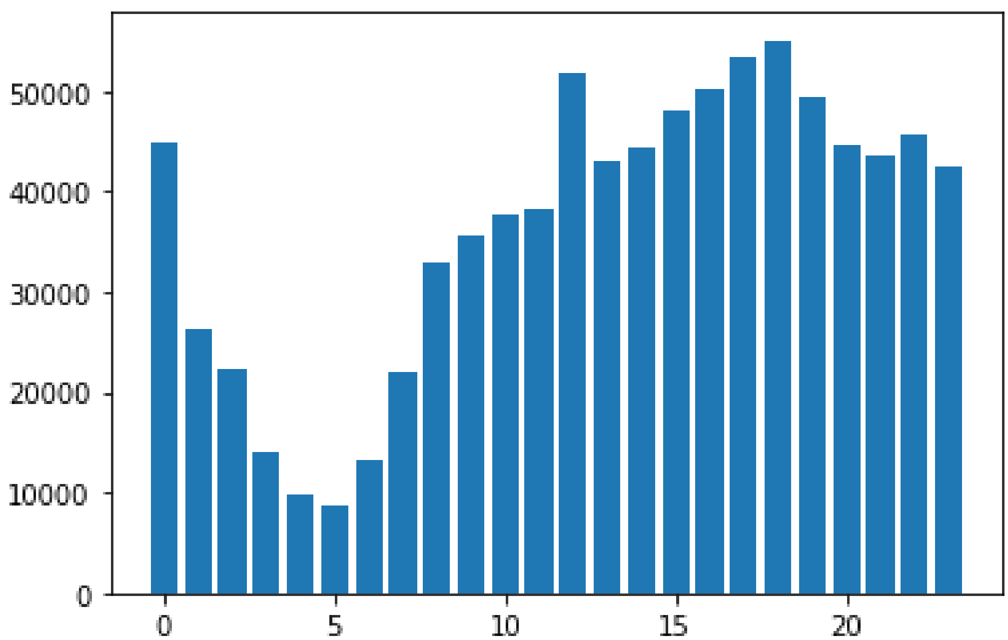




[20]:
```
# 7. quantile cuts: which hours represent biggest crime rate - weird results␣
↪for now, can't understand it

train_hour_valuecnts = train_eda_dates['Hour'].value_counts()

display(train_hour_valuecnts.sort_values())

# plt.bar(
#     train_hour_valuecnts.index, train_hour_valuecnts
# )
# plt.show()

plt.bar(
    train_hour_valuecnts.index, train_hour_valuecnts
)
plt.show()

display(
    pd.qcut(train_hour_valuecnts, 3, retbins=True)[1]
)
```

```
5      8637
4      9863
6     13133
3     14014
7     22048
```

```
2      22296
1      26173
8      32900
9      35555
10     37806
11     38373
23     42460
13     43145
21     43661
14     44424
20     44694
0      44865
22     45741
15     48058
19     49475
16     50137
12     51934
17     53553
18     55104
Name: Hour, dtype: int64
```




```
array([ 8637., 34670., 44751., 55104.])
```

[21]:
```
# Cleanup for train/test _eda_dates dataframes

# del train_eda_dates
# del test_eda_dates
```

[22]:
```
# Explore 'Category' feature from the raw_train_df

# Note the skewness of the distribution of different crime types

raw_train_df.groupby(by='Category').size().sort_values()
```

[22]:
```
Category
TREA                               6
PORNOGRAPHY/OBSCENE MAT           22
GAMBLING                         146
SEX OFFENSES NON FORCIBLE        148
EXTORTION                        256
BRIBERY                          289
BAD CHECKS                       406
FAMILY OFFENSES                  491
SUICIDE                          508
EMBEZZLEMENT                    1166
LOITERING                       1225
ARSON                           1513
LIQUOR LAWS                     1903
RUNAWAY                         1946
DRIVING UNDER THE INFLUENCE     2268
KIDNAPPING                      2341
RECOVERED VEHICLE               3138
DRUNKENNESS                     4280
DISORDERLY CONDUCT              4320
SEX OFFENSES FORCIBLE           4388
STOLEN PROPERTY                 4540
TRESPASS                        7326
PROSTITUTION                    7484
WEAPON LAWS                     8555
SECONDARY CODES                 9985
FORGERY/COUNTERFEITING         10609
FRAUD                          16679
ROBBERY                        23000
MISSING PERSON                 25989
SUSPICIOUS OCC                 31414
BURGLARY                       36755
WARRANTS                       42214
VANDALISM                      44725
VEHICLE THEFT                  53781
DRUG/NARCOTIC                  53971
ASSAULT                        76876
```

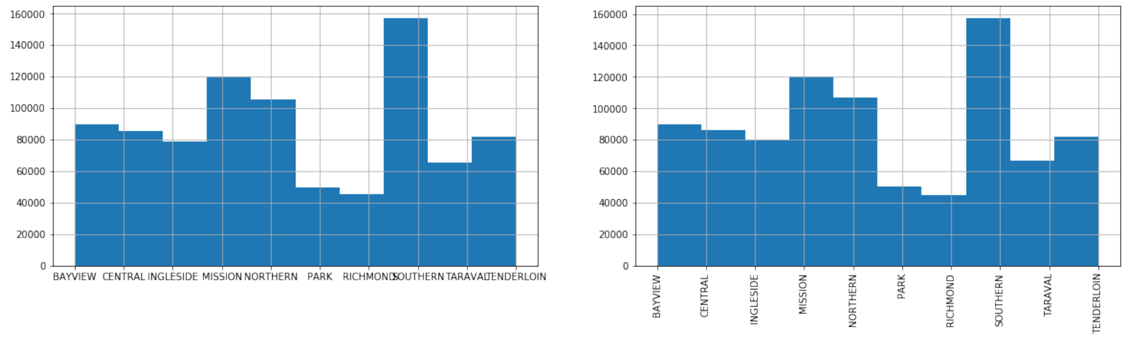
```
NON-CRIMINAL                      92304
OTHER OFFENSES                   126182
LARCENY/THEFT                    174900
dtype: int64
```

[23]:
```
# Explore 'Descript' feature from the raw_train_df

# This feature might be helpful when creating features
# related to "how well police behaved in a X district" => how bad district (?)

# Conclusion: because this feature is not in the test set, drop it when␣
↪creating baseline model.

display(
    raw_train_df[ ['Descript', 'Category'] ].sample(20)
)
```

	Descript	Category
568456	POSS OF PROHIBITED WEAPON	WEAPON LAWS
12057	CASE CLOSURE	NON-CRIMINAL
214054	CASE CLOSURE	NON-CRIMINAL
72240	GRAND THEFT FROM LOCKED AUTO	LARCENY/THEFT
459505	BATTERY	ASSAULT
267258	GRAND THEFT PICKPOCKET	LARCENY/THEFT
849106	AGGRAVATED ASSAULT WITH BODILY FORCE	ASSAULT
135917	GRAND THEFT FROM LOCKED AUTO	LARCENY/THEFT
400921	POSSESSION OF NARCOTICS PARAPHERNALIA	DRUG/NARCOTIC
730731	MALICIOUS MISCHIEF, VANDALISM OF VEHICLES	VANDALISM
638026	GRAND THEFT FROM LOCKED AUTO	LARCENY/THEFT
270033	CREDIT CARD, THEFT BY USE OF	FRAUD
451869	AIDED CASE, MENTAL DISTURBED	NON-CRIMINAL
377247	TRAFFIC VIOLATION ARREST	OTHER OFFENSES
824735	FOUND PERSON	MISSING PERSON
502435	GRAND THEFT FROM LOCKED AUTO	LARCENY/THEFT
798297	BATTERY	ASSAULT
719084	GRAND THEFT FROM LOCKED AUTO	LARCENY/THEFT
441927	CHECKS, POSSESSION WITH INTENT TO PASS	FORGERY/COUNTERFEITING
217137	POSS OF PROHIBITED WEAPON	WEAPON LAWS

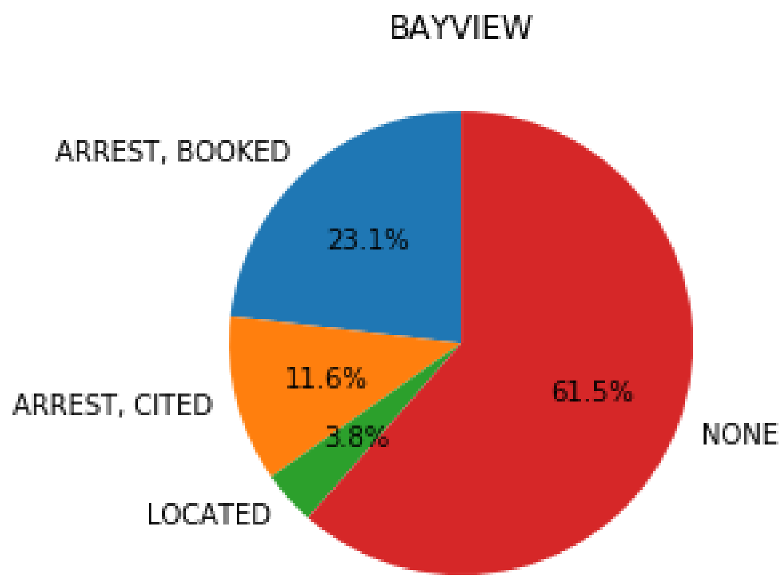
[24]:
```
# Explore 'PdDistrict' feature

# Conclusion: 'PdDistrict' should be a good feature - definitely use it in␣
↪modelling

# Training and Test sets
fig, [ax_0, ax_1] = plt.subplots(1, 2, figsize=(20, 5))
raw_train_df['PdDistrict'].sort_values().hist(bins=10, ax=ax_0)
```

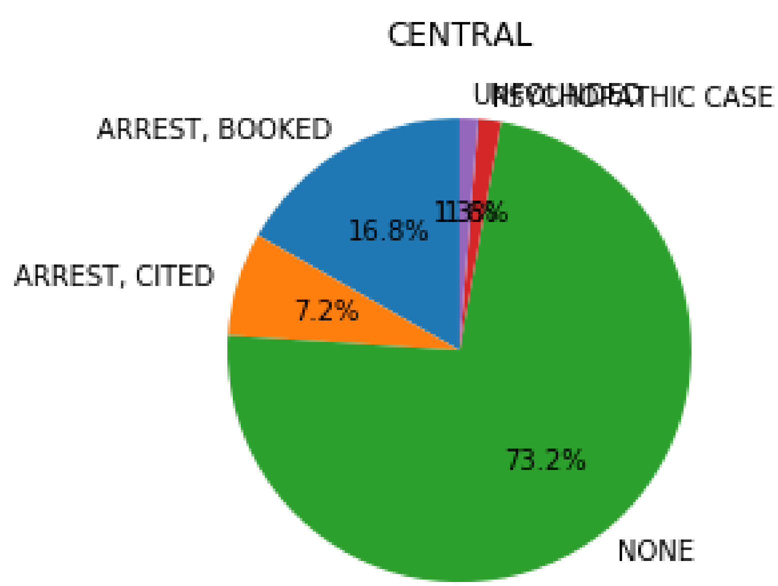
```
plt.xticks(rotation=90)
raw_test_df['PdDistrict'].sort_values().hist(bins=10, ax=ax_1)
plt.xticks(rotation=90)
plt.show()
```

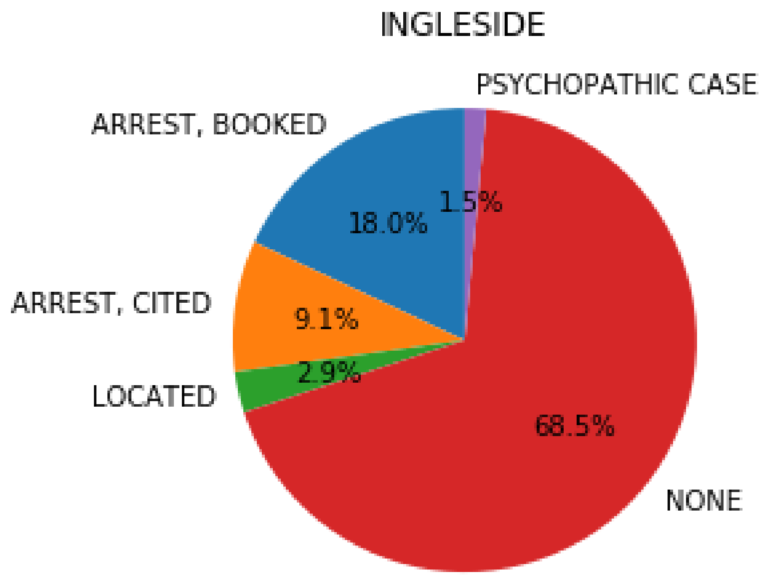


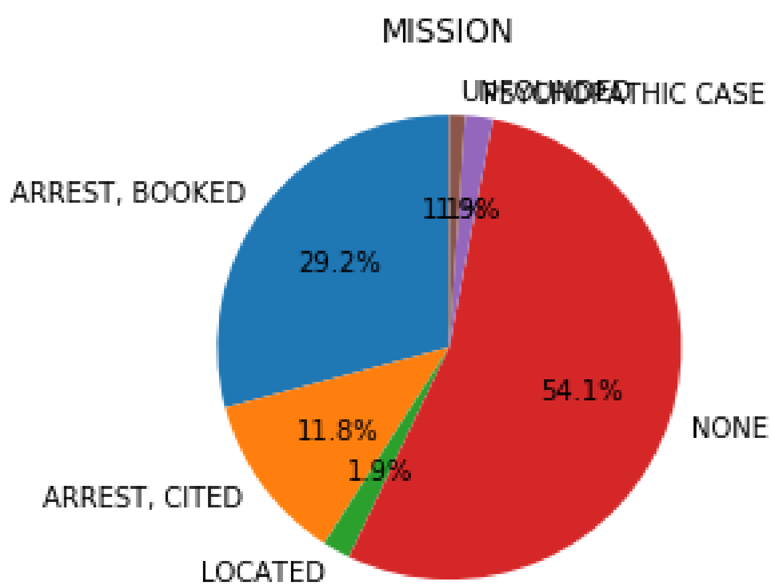
[25]:
```
# Explore 'Resolution' feature

# It is weird that there are so many rows with 'Resolution'='NONE' (526790␣
↪rows)

# We might drop this feature or use it to identify some district as good/bad,
# i.e. lots of arrests & cited - good one; lots of arrests & booked - bad one
# OR
# 'NONE' might (or not) event represent false call

for district_name, district_entries in raw_train_df.groupby(by='PdDistrict'):
    resolutions_in_district = district_entries.groupby(by='Resolution').size()
#     display(
#         resolutions_in_district.sum() -
#         resolutions_in_district[ resolutions_in_district > 1000 ].sum()   #␣
↪losing 3k-5k elements
#     )
    resolutions_in_district = resolutions_in_district[ resolutions_in_district␣
↪> 1000 ]
    plt.pie(
        resolutions_in_district, labels=resolutions_in_district.index,
        autopct='%1.1f%%', startangle=90
    )
    plt.title(district_name)
    plt.show()

display(
    raw_train_df.groupby(by='Resolution').size().sort_values()
)
```

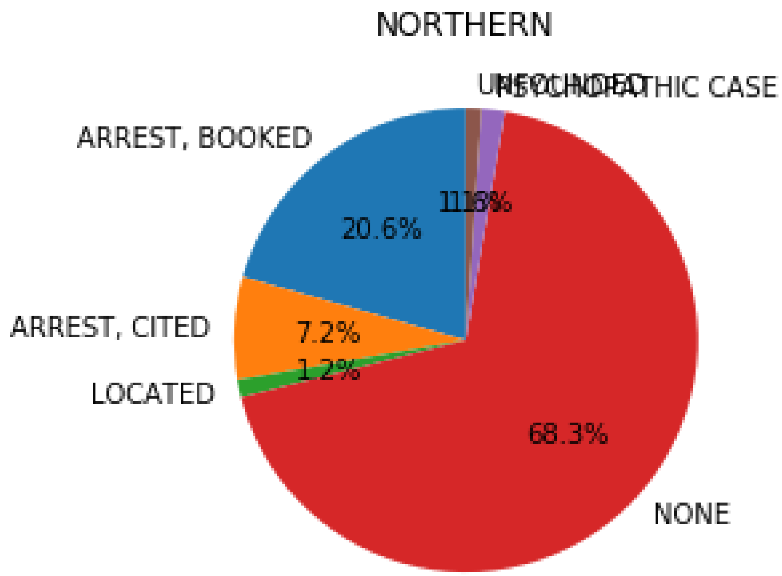
```
display(
    raw_train_df[ ['Resolution', 'Category', 'PdDistrict'] ].sample(20)
)
```

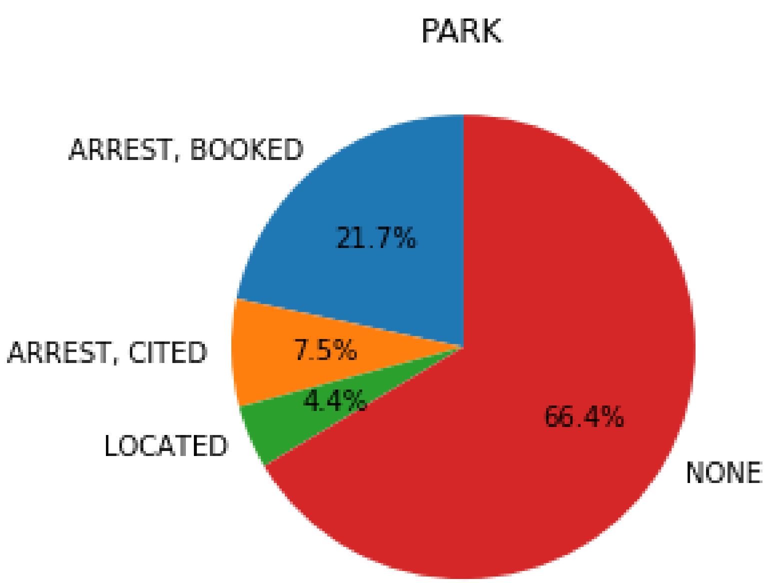


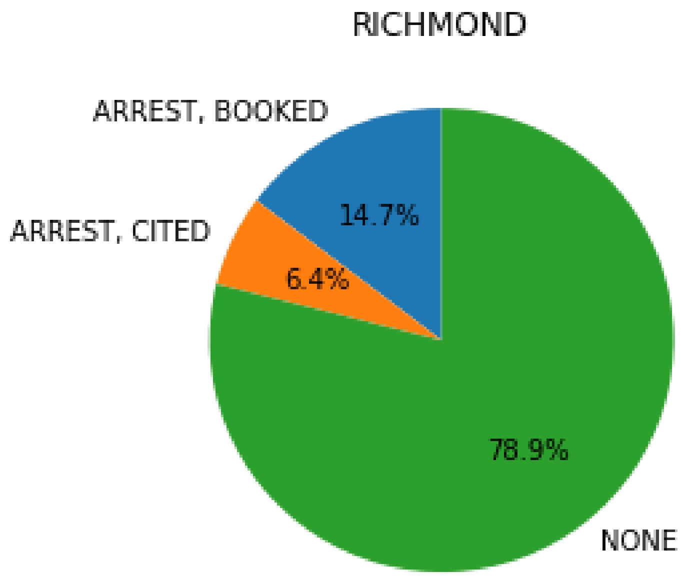


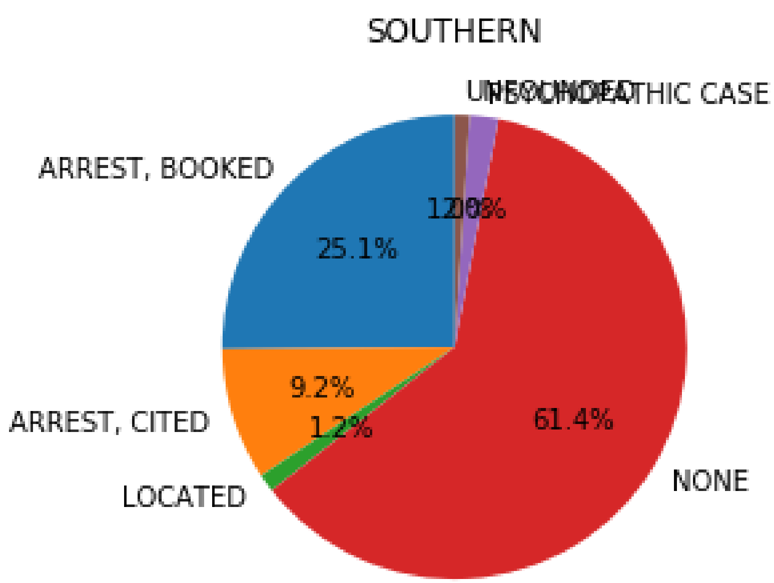

INGLESIDE
PSYCHOPATHIC CASE
ARREST, BOOKED
1.5%
18.0%
ARREST, CITED
9.1%
2.9%
LOCATED
68.5%
NONE

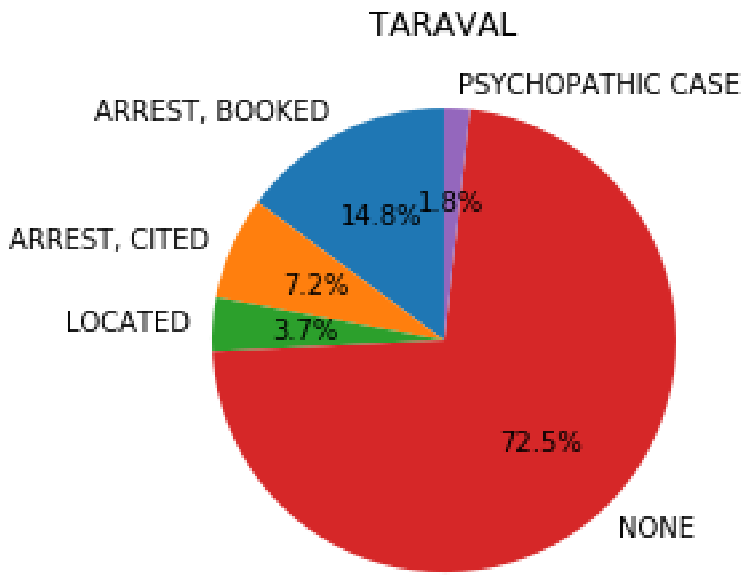

MISSION
ARREST, BOOKED
29.2%
54.1%
NONE
11.8%
1.9%
ARREST, CITED
LOCATED

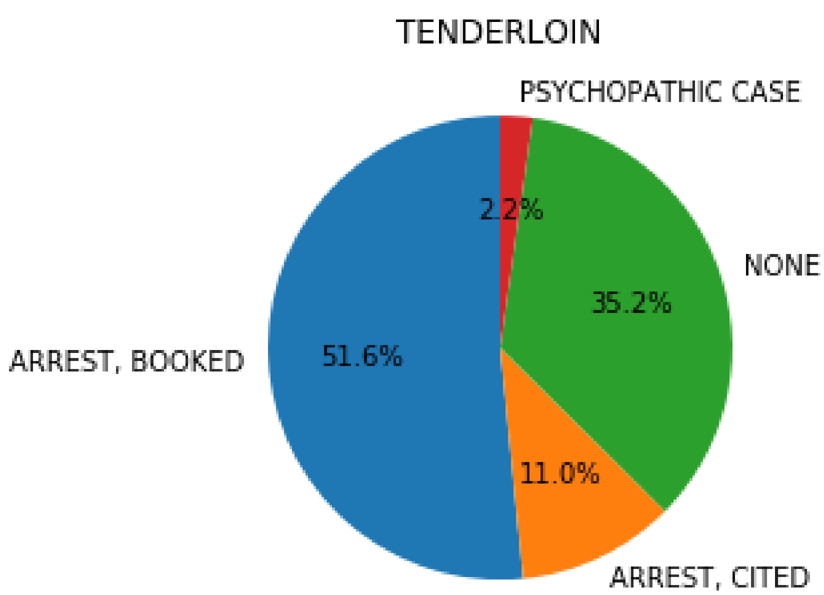

NORTHERN
PSYCHOPATHIC CASE
ARREST, BOOKED
20.6%
ARREST, CITED
7.2%
1.2%
LOCATED
68.3%
NONE


PARK
ARREST, BOOKED
21.7%
ARREST, CITED
7.5%
4.4%
LOCATED
66.4%
NONE


RICHMOND
ARREST, BOOKED
14.7%
ARREST, CITED
6.4%
78.9%
NONE


SOUTHERN
ARREST, BOOKED
25.1%
9.2%
ARREST, CITED
1.2%
LOCATED
61.4%
NONE






```
Resolution
PROSECUTED FOR LESSER OFFENSE                51
CLEARED-CONTACT JUVENILE FOR MORE INFO      217
JUVENILE DIVERTED                           355
```

```
JUVENILE ADMONISHED                          1455
EXCEPTIONAL CLEARANCE                        1530
PROSECUTED BY OUTSIDE AGENCY                 2504
JUVENILE CITED                               3332
NOT PROSECUTED                               3714
DISTRICT ATTORNEY REFUSES TO PROSECUTE       3934
COMPLAINANT REFUSES TO PROSECUTE             3976
JUVENILE BOOKED                              5564
UNFOUNDED                                    9585
PSYCHOPATHIC CASE                           14534
LOCATED                                     17101
ARREST, CITED                               77004
ARREST, BOOKED                             206403
NONE                                       526790
dtype: int64
```

```
                              Resolution        Category  PdDistrict
413322                              NONE    NON-CRIMINAL    SOUTHERN
18120                     ARREST, BOOKED        WARRANTS     BAYVIEW
168581                              NONE         ROBBERY     BAYVIEW
29097                     ARREST, BOOKED  OTHER OFFENSES  TENDERLOIN
156087                              NONE  MISSING PERSON        PARK
841721                    ARREST, BOOKED   DRUG/NARCOTIC    SOUTHERN
797316  COMPLAINANT REFUSES TO PROSECUTE  SUSPICIOUS OCC     TARAVAL
775088                              NONE   LARCENY/THEFT     CENTRAL
642731                              NONE         ASSAULT     CENTRAL
300598                    ARREST, BOOKED   DRUG/NARCOTIC  TENDERLOIN
417491                              NONE    NON-CRIMINAL    NORTHERN
687332                    ARREST, BOOKED  OTHER OFFENSES     BAYVIEW
347970                              NONE  SUSPICIOUS OCC   INGLESIDE
292070                              NONE   LARCENY/THEFT  TENDERLOIN
533642                              NONE       VANDALISM        PARK
396260                              NONE   VEHICLE THEFT    NORTHERN
396016                              NONE    NON-CRIMINAL    NORTHERN
871913                              NONE   LARCENY/THEFT    SOUTHERN
769067                              NONE        BURGLARY     CENTRAL
653166                    ARREST, BOOKED        WARRANTS  TENDERLOIN
```

[26]:
```
# Explore 'Address' feature

# Note: almost alll top Addresses by crime rate have 'Block' in their␣
↪description.
# Note: intersections also identify top addresses by crime rate

# Conclusion: transform 'Address' into 'IsBlock' and 'IsIntersection'
```

```python
# Note: 800 Block of BRYANT ST has top crime rate in train and test set

# Conclusion: add 'Is BryantSt800Blk' feature

display(
    len(raw_train_df['Address'].unique()), len(raw_test_df['Address'].unique())
)

display(
    raw_train_df['Address'].value_counts().head(3),
    raw_train_df['Address'].value_counts().tail(3)
)
display(
    raw_test_df['Address'].value_counts().head(3),
    raw_test_df['Address'].value_counts().tail(3),
)
```

```
23228
```

```
23184
```

```
800 Block of BRYANT ST      26533
800 Block of MARKET ST       6581
2000 Block of MISSION ST     5097
Name: Address, dtype: int64
```

```
MIDDLEFIELD DR / EUCALYPTUS DR    1
SHIELDS ST / ORIZABA AV           1
PANORAMA DR / LONGVIEW CT         1
Name: Address, dtype: int64
```

```
800 Block of BRYANT ST      26984
800 Block of MARKET ST       6883
2000 Block of MISSION ST     4955
Name: Address, dtype: int64
```

```
2200 Block of GREAT HWY        1
100 Block of TUBBS ST          1
THORNTON AV / BRIDGEVIEW DR    1
Name: Address, dtype: int64
```

[27]:
```
# Explore crimes on street corners (street is like 'street 1 / street 2')

# Almost 1/4 of crimes happens on such intersections: this might be a decent␣
↪feature

street_corner_crimes = raw_train_df['Address'].apply(
    lambda x: 1 if '/' in x else 0
)

display(
    street_corner_crimes.value_counts()
)
```

```
0    617231
1    260818
Name: Address, dtype: int64
```

[28]:
```
# Intermediate DF for fixing features

fixd_train_df = raw_train_df.copy()

fixd_test_df = raw_test_df.copy()

display(
    fixd_train_df.shape, fixd_test_df.shape
)
```

```
(878049, 9)
```

```
(884262, 7)
```

[29]:
```
# Fix rows with unusual Longtitude and Latitude == fix ['X', 'Y'] features␣
↪values

# For rows that have Y=90.0 but where similiar ADDRESSES have valid coordinates:
# Replace coordinates with the same coordinates

train_invalid_rows = raw_train_df[ raw_train_df['Y'] == 90.0 ]
test_invalid_rows = raw_test_df[ raw_test_df['Y'] == 90.0 ]

def _ugly_fix_invalid_coords_inplace(invalid_rows_df, tofix_df):
    """note: used global variable concat_no_invalid_rows"""
    concat_no_invalid_rows = raw_concat_traintest_df[␣
↪raw_concat_traintest_df['Y'] != 90.0 ]
```

```
    for row_idx, row in invalid_rows_df.iterrows():
        addr_occurences_in_concat = concat_no_invalid_rows[
            concat_no_invalid_rows['Address'] == row['Address']
        ]
        if addr_occurences_in_concat.shape[0]:
            # Fix longtitude
            tofix_df.iloc[row_idx, tofix_df.columns.get_loc('X')] =␣
 ↪addr_occurences_in_concat['X'].iloc[0]
            # Fix latitude
            tofix_df.iloc[row_idx, tofix_df.columns.get_loc('Y')] =␣
 ↪addr_occurences_in_concat['Y'].iloc[0]

_ugly_fix_invalid_coords_inplace(train_invalid_rows, fixd_train_df)  # 67␣
 ↪invalid rows -> 61 invalid rows
_ugly_fix_invalid_coords_inplace(test_invalid_rows, fixd_test_df)  # 76 invalid␣
 ↪rows -> 65 invalid rows

# Otherwise: replace with most common value.

def _ugly_fix_invalid_coords_inplace_2(tofix_df):
    tofix_df.loc[ tofix_df['Y'] == 90.0, 'X' ] = tofix_df['X'].mode()[0]  #␣
 ↪note: because we use 'Y'=90, do X first
    tofix_df.loc[ tofix_df['Y'] == 90.0, 'Y' ] = tofix_df['Y'].mode()[0]

_ugly_fix_invalid_coords_inplace_2(fixd_train_df)
_ugly_fix_invalid_coords_inplace_2(fixd_test_df)
```

[30]:
```
overview_invalid_long_lat(fixd_train_df)  # should be 0

overview_invalid_long_lat(fixd_test_df)  # should be 0
```

```
'Found longtitude invalid values: (0, 9)'
```

```
'Found latitude invalid values: (0, 9)'
```

```
'Found longtitude invalid values: (0, 7)'
```

```
'Found latitude invalid values: (0, 7)'
```

[31]:
```
# Fix duplicated rows in train and test sets

display(
    'Duplicated items in train set: {0}'.format(fixd_train_df.duplicated().
 ↪sum()),  # 2323 items
```

```
    'Duplicated items in test set: {0}'.format(fixd_test_df.duplicated().sum())␣
 ↪ # 0 items
)

fixd_train_df = fixd_train_df.drop_duplicates()
```

```
'Duplicated items in train set: 2323'
```

```
'Duplicated items in test set: 0'
```

[32]:
```
display(
    'Duplicated items in train set: {0}'.format(fixd_train_df.duplicated().
 ↪sum()),  # should be 0
    'Duplicated items in test set: {0}'.format(fixd_test_df.duplicated().sum())␣
 ↪ # should be 0
)
```

```
'Duplicated items in train set: 0'
```

```
'Duplicated items in test set: 0'
```

[33]:
```
# Intermediate array for performing feature engineering / features dropping

feateng_train_df = fixd_train_df.copy()

feateng_test_df = fixd_test_df.copy()

# Cleanup old intermediate DFs
# del fixd_train_df
# del fixd_test_df
```

[34]:
```
def date_col_to_datetime_inplace(dataset_df, date_col_name='Dates'):
    dataset_df[date_col_name] = pd.to_datetime(dataset_df[date_col_name])


def add_date_features_inplace(dataset_df, date_col_name='Dates'):
    # Time
    dataset_df['Hour'] = dataset_df[date_col_name].dt.hour
    dataset_df['Minute'] = dataset_df[date_col_name].dt.minute
#     dataset_df['IsQuietTime'] = 0
#     dataset_df.loc[ (dataset_df['Hour'] >= 1) & (dataset_df['Hour'] <= 6),␣
 ↪'IsQuietTime' ] = 1
#     dataset_df['IsDangerousTime'] = 0
```

```
#     dataset_df.loc[ (dataset_df['Hour'] >= 15) & (dataset_df['Hour'] <= 19),␣
↪'IsDangerousTime' ] = 1
#     dataset_df['IsMidnight'] = 0
#     dataset_df.loc[ (dataset_df['Hour'] == 0), 'IsMidnight' ] = 1
#     dataset_df['IsLunchTime'] = 0
#     dataset_df.loc[ (dataset_df['Hour'] == 12), 'IsLunchTime' ] = 1
    # Date: general
    dataset_df['n_days_passed'] = dataset_df[date_col_name] -␣
↪dataset_df[date_col_name].min()
    dataset_df['n_days_passed'] = dataset_df['n_days_passed'].apply( lambda x:␣
↪x.days )
    dataset_df['Day'] = dataset_df[date_col_name].dt.day
    dataset_df['Month'] = dataset_df[date_col_name].dt.month
    dataset_df['Year'] = dataset_df[date_col_name].dt.year
    # Date: other
    dataset_df['DayOfWeek'] = dataset_df[date_col_name].dt.weekday  # Overwrite␣
↪raw 'DayOfWeek' feature
#     dataset_df['WeekOfYear'] = dataset_df[date_col_name].dt.weekofyear
#     dataset_df['IsWeekend'] = 0
#     dataset_df.loc[ dataset_df['DayOfWeek'] >= 5, 'IsWeekend' ] = 1
    # Certain "unusual risk" days
#     dataset_df['IsMiddleOfWeek'] = 0
#     dataset_df.loc[ dataset_df['DayOfWeek'] == 2, 'IsMiddleOfWeek' ] = 1
#     dataset_df['IsFriday'] = 0  # highest rate of crime
#     dataset_df.loc[ dataset_df['DayOfWeek'] == 4, 'IsFriday' ] = 1
#     dataset_df['IsSunday'] = 0  # if crime happened even on Sundays - very␣
↪gangerous one  # lower rate of crime
#     dataset_df.loc[ dataset_df['DayOfWeek'] == 6, 'IsSunday' ] = 1
```

[35]:
```
date_col_to_datetime_inplace(feateng_train_df)
# display(feateng_train_df.dtypes)  # Dates: datetime64[ns]

date_col_to_datetime_inplace(feateng_test_df)
# display(feateng_test_df.dtypes)  # Dates: datetime64[ns]
```

[36]:
```
add_date_features_inplace(feateng_train_df)

add_date_features_inplace(feateng_test_df)
```

[37]:
```
# Explore newly created features

# display(
#     "IsQuietTime", feateng_train_df['IsQuietTime'].value_counts()
# )

# display(
#     "IsDangerousTime", feateng_train_df['IsDangerousTime'].value_counts()  #␣
↪might be a bad one
```

```
# )

# display(
#     "IsMidnight", feateng_train_df['IsMidnight'].value_counts()
# )

# display(
#     "IsLunchTime", feateng_train_df['IsLunchTime'].value_counts()
# )

# display(
#     "IsWeekend", feateng_train_df['IsWeekend'].value_counts()
# )

# display(
#     "IsMiddleOfWeek", feateng_train_df['IsMiddleOfWeek'].value_counts()
# )

# display(
#     "IsFriday", feateng_train_df['IsFriday'].value_counts()
# )

# display(
#     "IsSunday", feateng_train_df['IsSunday'].value_counts()
# )
```

[38]:
```
def add_address_features_inplace(dataset_df, addr_col_name='Address'):
    dataset_df['IsBlock'] = dataset_df[addr_col_name].str.contains('block',␣
 ↪case=False)

#     dataset_df['IsIntersection'] = 0
#     intersection_addresses = dataset_df[addr_col_name].str.contains('/',␣
 ↪case=False, regex=False)
#     isintersection_col_idx = dataset_df.columns.get_loc('IsIntersection')
#     dataset_df.iloc[ intersection_addresses[intersection_addresses].index,␣
 ↪isintersection_col_idx ] = 1

#     dataset_df['IsBryantSt800Blk'] = 0
#     bryantst = "800 Block of BRYANT ST"
#     bryantst_addresses = dataset_df[addr_col_name].str.contains(bryantst,␣
 ↪case=False, regex=False)
#     isbryantst_col_idx = dataset_df.columns.get_loc('IsBryantSt800Blk')
#     dataset_df.iloc[ bryantst_addresses[bryantst_addresses].index,␣
 ↪isbryantst_col_idx ] = 1
```

[39]:
```
add_address_features_inplace(feateng_train_df)

add_address_features_inplace(feateng_test_df)
```

[40]:
```
# Display Pearson correlation

display(
    feateng_train_df.corr()
)
```

```
              DayOfWeek         X         Y      Hour    Minute  n_days_passed       Day      M
DayOfWeek      1.000000  0.008231  0.013497 -0.021014 -0.014083       0.015066  0.010622  0.01
X              0.008231  1.000000  0.154168  0.002279  0.057871       0.002137  0.002144 -0.00
Y              0.013497  0.154168  1.000000 -0.010809  0.013604       0.024728  0.004183  0.00
Hour          -0.021014  0.002279 -0.010809  1.000000  0.010104      -0.006310  0.015512 -0.00
Minute        -0.014083  0.057871  0.013604  0.010104  1.000000       0.018708  0.009680 -0.00
n_days_passed  0.015066  0.002137  0.024728 -0.006310  0.018708       1.000000 -0.002012  0.03
Day            0.010622  0.002144  0.004183  0.015512  0.009680      -0.002012  1.000000  0.01
Month          0.010766 -0.000188  0.003941 -0.001786 -0.008210       0.030573  0.016912  1.00
Year           0.014135  0.002137  0.024374 -0.006266  0.019276       0.996870 -0.009961 -0.04
IsBlock       -0.013532 -0.038688 -0.052363 -0.043849 -0.051452       0.027707 -0.007845  0.00
```

[41]:
```
# Display the skew

display(
    feateng_train_df.skew()
)

display(
    feateng_test_df.skew()
)
```

```
DayOfWeek       -0.005626
X               -1.203543
Y               -0.722581
Hour            -0.513167
Minute           0.360971
n_days_passed   -0.006408
Day              0.017377
Month            0.022287
Year             0.012475
IsBlock         -0.886653
dtype: float64
```

```
Id               3.742898e-16
DayOfWeek       -2.027639e-04
X               -1.206417e+00
Y               -7.195169e-01
Hour            -5.113385e-01
```

```
Minute            3.608941e-01
n_days_passed     1.574413e-03
Day               6.055429e-03
Month             4.871328e-02
Year              1.794998e-02
IsBlock          -8.913023e-01
dtype: float64
```

[42]:
```
# Intermediate DFs : dropping features

dropfeat_train_df = feateng_train_df.copy()
dropfeat_test_df = feateng_test_df.copy()

# Cleanup
# del feateng_train_df
# del feateng_test_df
```

[43]:
```
# y_train
dropfeat_train_category = dropfeat_train_df['Category']
# X_train
dropfeat_train_df = dropfeat_train_df.drop(
    ['Dates', 'Category', 'Descript', 'Resolution', 'Address'],
    axis=1
)

# X_test
# 'id' is saved in raw_test_df DataFrame
dropfeat_test_df = dropfeat_test_df.drop(
    ['Id', 'Dates', 'Address'],
    axis=1
)
```

[44]:
```
display(
    dropfeat_train_category.head(),
    dropfeat_train_df.head(),
    dropfeat_train_df.head()
)  # should hold same dimensions
```

```
0          WARRANTS
1    OTHER OFFENSES
2    OTHER OFFENSES
3     LARCENY/THEFT
4     LARCENY/THEFT
Name: Category, dtype: object
```

```
   DayOfWeek PdDistrict           X          Y  Hour  Minute  n_days_passed  Day  Month  Year
0          2   NORTHERN -122.425892  37.774599    23      53           4510   13      5  2015
```

```
1          2   NORTHERN -122.425892  37.774599    23      53            4510   13         5  2015
2          2   NORTHERN -122.424363  37.800414    23      33            4510   13         5  2015
3          2   NORTHERN -122.426995  37.800873    23      30            4510   13         5  2015
4          2       PARK -122.438738  37.771541    23      30            4510   13         5  2015

   DayOfWeek PdDistrict           X          Y  Hour  Minute  n_days_passed  Day  Month  Year
0          2   NORTHERN -122.425892  37.774599    23      53            4510   13      5  2015
1          2   NORTHERN -122.425892  37.774599    23      53            4510   13      5  2015
2          2   NORTHERN -122.424363  37.800414    23      33            4510   13      5  2015
3          2   NORTHERN -122.426995  37.800873    23      30            4510   13      5  2015
4          2       PARK -122.438738  37.771541    23      30            4510   13      5  2015
```

[45]:
```
# Intermediate DFs for feature encoding before applying data to model

featenc_category_series = dropfeat_train_category.copy()

featenc_train_df = dropfeat_train_df.copy()
featenc_test_df = dropfeat_test_df.copy()

# Cleanup old intermediate DFs
# del dropfeat_train_df
# del dropfeat_test_df
```

[46]:
```
# Encode 'PdDistrict'

distr_enc = LabelEncoder()
featenc_train_df['PdDistrict'] = distr_enc.fit_transform(␣
↪featenc_train_df['PdDistrict'] )
featenc_test_df['PdDistrict'] = distr_enc.transform(␣
↪featenc_test_df['PdDistrict'] )
```

[47]:
```
# Encode 'IsBlock'

featenc_train_df['IsBlock'] = featenc_train_df['IsBlock'].apply(
    lambda x: int( x )
)
featenc_test_df['IsBlock'] = featenc_test_df['IsBlock'].apply(
    lambda x: int( x )
)
```

[48]:
```
# Encode 'Category' in training set

cat_enc = LabelEncoder()
featenc_category_series = cat_enc.fit_transform( featenc_category_series )
```

[49]:
```
# Intermediate DFs for baseline model

baseline_category_series = featenc_category_series.copy()
```

```
baseline_train_df = featenc_train_df.copy()
baseline_test_df = featenc_test_df.copy()

# Cleanup old intermediate DFs
# del featenc_train_df
# del featenc_train_df
```

[51]:
```
# train/validation sets

X_tr, X_val, y_tr, y_val = train_test_split(
    baseline_train_df, baseline_category_series
)

y_tr_categories_cnt = pd.unique( y_tr ).shape[0]  # 39
y_val_categories_cnt = pd.unique( y_val ).shape[0]  # 39
raw_y_train_categories_cnt = pd.unique( raw_train_df['Category'] ).shape[0]  #
 ↪39

display(
    y_tr_categories_cnt,
    y_val_categories_cnt,
    raw_y_train_categories_cnt
)
```

```
39

39

39
```

[52]:
```
# Build the baseline model

lgbm_clf_model = LGBMClassifier(
    objective='multiclass',
    num_class=y_tr_categories_cnt
)

lgbm_clf_model.fit( X_tr, y_tr )
```

[52]:
```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gain=0.0,
               n_estimators=100, n_jobs=-1, num_class=39, num_leaves=31,
               objective='multiclass', random_state=None, reg_alpha=0.0,
               reg_lambda=0.0, silent=True, subsample=1.0,
```

```
                    subsample_for_bin=200000, subsample_freq=0)
```

[53]:
```
# Show features importances

baseline_lgbm_feat_imp = pd.DataFrame(
    sorted( zip( lgbm_clf_model.feature_importances_, X_tr.columns ),␣
 ↪reverse=True ),
    columns=['Importance Value', 'Feature']
)

display(
    baseline_lgbm_feat_imp
)
```

```
    Importance Value        Feature
0              21769              X
1              21766              Y
2              17717  n_days_passed
3              15369         Minute
4              13636           Hour
5               8714            Day
6               5766          Month
7               5120      DayOfWeek
8               3409     PdDistrict
9               3039        IsBlock
10               695           Year
```

[54]:
```
# Evaluate baseline method

baseline_y_pred = lgbm_clf_model.predict_proba(X_val)

display(
    log_loss(
        y_val,
        baseline_y_pred
    )
)
```

```
3.316081926146441
```

[65]:
```
# Try out kNN

knn_model = KNeighborsClassifier(
    n_neighbors=100,
    n_jobs=-1
)
```

```
knn_model.fit(X_tr, y_tr)
print('done fitting')

predictions = knn_model.predict_proba(X_val)
print('done predictions')

display(
    log_loss(
        y_val,
        predictions
    )
)
```

```
done fitting
done predictions
```

```
4.024410873165233
```

[83]:
```
# Submit model predictions

def create_submission_file(file_path, model, predictions):
    submission_df = pd.DataFrame( {'Id': raw_test_df['Id']} )

    for category_name in cat_enc.inverse_transform(model.classes_):  # note: .
↪classes_ for encoder model!!!
        submission_df[category_name] = 0

    for row_num, pred_str in enumerate(cat_enc.inverse_transform(predictions)):
        submission_df[pred_str][row_num] = 1
        if row_num % 100000 == 0:  # dbg purposes
            print(row_num)

    submission_df.to_csv(file_path, index=False)
```

[67]:
```
knn_model = KNeighborsClassifier(n_neighbors=100, n_jobs=-1)
knn_model.fit( baseline_train_df, baseline_category_series )
knn_predictions = knn_model.predict(baseline_test_df)
```

[84]:
```
create_submission_file('knn_submission.csv', knn_model, knn_predictions)
```

```
0
100000
200000
300000
400000
500000
```

```
600000
700000
800000
```

[85]:
```
lgbm_clf_model = LGBMClassifier(
    objective='multiclass',
    num_class=y_tr_categories_cnt
)

lgbm_clf_model.fit( baseline_train_df, baseline_category_series )
lgbm_predictions = lgbm_clf_model.predict( baseline_test_df )
```

[86]:
```
create_submission_file('lgbm_submission.csv', lgbm_clf_model, lgbm_predictions)
```

```
0
100000
200000
300000
400000
500000
600000
700000
800000
```

[21]:
```
# todo: work with 'Address'

# todo: identify RATING for each 'Address' depending on # of crimes on that␣
↪street.
```

[ ]:
```
# todo: work with 'PdDistrict'
```

[ ]:
```
# todo: apply Prophet to identify SEASONAL patterns!!!
```

[ ]:
```
# todo: qcut for hour to check if IsRush/IsQuiet features are worth it
```

[ ]:
```
# todo: isWinter/Summer/...  - "season" feature
```