Data-science-projects (/github/mpruna/Data-science-projects/tree/a5470e5e3195750f83945a94e743b36fb6e60090)
/ Montgomery crime analysis (/github/mpruna/Data-science-projects/tree/a5470e5e3195750f83945a94e743b36fb6e60090/Montgomery crime analysis)

Analyzing Montgomery County crimes to find useful patterns

Montgomery is a county in the Maryland US located on the east coast Montgomery County, Maryland (https://en.wikipedia.org/wiki/Montgomery_County,_Maryland).

Improvement of various aspects of social life entitles a proactive and reactive analysis. With this in mind I will be looking to to find patterns by performing time, location and crime classification analysis. For this project I will heavily rely on graph visualization as a picture worths a thousand words.

Here are some conclusion highlights:

1. Time analysis:
   A. Most of the crimes are committed Tuesday
   B. On 24 hour basis most of the crimes are committed between 7 a.m - 11 p.m
   C. October has the highest crime count
2. Classification analysis:
   A. Violent/Non-Violent crimes rates are pretty even 42.8%/57.2%
3. Location analysis:
   A. Cities with highest crime counts are : Silver Spring, Rockville, Gaithersburg
   B. Most of the crimes happen in the street, residence or parking lot
   C. Silver Spring Police District has the highest crime rates

Dataset for this project: here (https://data.montgomerycountymd.gov/Public-Safety/Crime/icn6-v9z3)

In [26]:

```
import pandas as pd
import numpy as np

crimes = pd.read_csv("MontgomeryCountyCrime2013.csv")
crimes.head()
```

Out[26]:

	Incident ID	CR Number	Dispatch Date / Time	Class	Class Description	Police District Name	Block Address	City	State	Zip Code	...	Sector	E
0	200939101	13047006	10/02/2013 07:52:41 PM	511	BURG FORCE-RES/NIGHT	OTHER	25700 MT RADNOR DR	DAMASCUS	MD	20872.0	...	NaN	N
1	200952042	13062965	12/31/2013 09:46:58 PM	1834	CDS-POSS MARIJUANA/HASHISH	GERMANTOWN	GUNNERS BRANCH RD	GERMANTOWN	MD	20874.0	...	M	5
2	200926636	13031483	07/06/2013 09:06:24 AM	1412	VANDALISM-MOTOR VEHICLE	MONTGOMERY VILLAGE	OLDE TOWNE AVE	GAITHERSBURG	MD	20877.0	...	P	(
3	200929538	13035288	07/28/2013 09:13:15 PM	2752	FUGITIVE FROM JUSTICE(OUT OF STATE)	BETHESDA	BEACH DR	CHEVY CHASE	MD	20815.0	...	D	:
4	200930689	13036876	08/06/2013 05:16:17 PM	2812	DRIVING UNDER THE INFLUENCE	BETHESDA	BEACH DR	SILVER SPRING	MD	20815.0	...	D	:

5 rows × 22 columns

Exploring the data

Each row in the dataset represents a crime being commited. Data contains location information, crime classification as well as various timestamps. In examining the dataset our goals would be to:

1. Find the columns that have meaningfull information, have minimum missing values, and also hold granular data.
2. We also need to perform data cleaning/manipulation

In [27]:

```
crimes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23369 entries, 0 to 23368
Data columns (total 22 columns):
Incident ID              23369 non-null int64
CR Number                23369 non-null int64
Dispatch Date / Time     23369 non-null object
Class                    23369 non-null int64
Class Description        23369 non-null object
Police District Name     23369 non-null object
Block Address            23369 non-null object
City                     23369 non-null object
State                    23369 non-null object
Zip Code                 23339 non-null float64
Agency                   23369 non-null object
Place                    23369 non-null object
Sector                   23323 non-null object
Beat                     23361 non-null object
PRA                      23363 non-null float64
Start Date / Time        23369 non-null object
End Date / Time          13191 non-null object
Latitude                 23208 non-null float64
Longitude                23208 non-null float64
Police District Number   23369 non-null object
Location                 23208 non-null object
Address Number           23237 non-null float64
dtypes: float64(5), int64(3), object(14)
memory usage: 3.9+ MB
```

We need to convert `Start Date / Time`; `End Date / Time`; `Dispatch Date/ Time` to datetime format.

Examine the number of (rows,columns) from the dataframe

In [28]:

```
crimes.shape
```

Out[28]:

```
(23369, 22)
```

Examining missing values.

In [29]:

```
crimes.isnull().sum()
```

Out[29]:

```
Incident ID                 0
CR Number                   0
Dispatch Date / Time        0
Class                       0
Class Description           0
Police District Name        0
Block Address               0
City                        0
State                       0
Zip Code                   30
Agency                      0
Place                       0
Sector                     46
Beat                        8
PRA                         6
Start Date / Time           0
End Date / Time         10178
Latitude                  161
Longitude                 161
Police District Number      0
Location                  161
Address Number            132
dtype: int64
```

1. Main takeaway here is that `End Date / Time` has a high number of missing value, for this reason it can't be used for our analysis
2. A deeper look into the collumns that have missing values is needed, also we have to determine which columns will provide usefull insight

In [30]:

```
columns_to_keep=['Zip Code','Sector','Beat','PRA','Latitude','Longitude','Location','Address Number']

for col in columns_to_keep:
    item_null=crimes[col].notnull()
    print(col+"\n",crimes[col][item_null==True].head(),"\n")
```

```
Zip Code
 0    20872.0
1    20874.0
2    20877.0
3    20815.0
4    20815.0
Name: Zip Code, dtype: float64

Sector
 1    M
2    P
3    D
4    D
5    P
Name: Sector, dtype: object

Beat
 1    5M1
2    6P3
3    2D1
4    2D3
5    6P1
Name: Beat, dtype: object

PRA
 1    470.0
2    431.0
3     11.0
4    178.0
5    444.0
Name: PRA, dtype: float64

Latitude
 10    39.105561
13    39.064334
14    39.067335
15    39.017814
16    39.178862
Name: Latitude, dtype: float64

Longitude
 10   -77.144617
13   -76.968985
14   -77.124027
15   -77.047689
16   -77.267406
Name: Longitude, dtype: float64

Location
 10    (39.105560882140779, -77.144617133574968)
13     (39.064334220776551, -76.96898520383327)
14    (39.067334736049553, -77.124027420153752)
15     (39.017814078946948, -77.04768926351224)
16    (39.178862442227761, -77.267405973712243)
Name: Location, dtype: object

Address Number
 0     25700.0
10      600.0
11     9200.0
13     2100.0
14     2200.0
Name: Address Number, dtype: float64
```

Columns analysis:

1. `Zip Code`; `Sector`; `Beat`; `Address Number` can't be used for our analysis as these details are meaningless for a casual reader.
2. Columns like `Dispatch Date / Time`; `Class Description`; `City`; `Start Date / Time`; `Police District Number` will be useful
3. `Latitude` and `Longitude` is not an obvious choice, at least from a comprehensibility standpoint but will be useful for map visualization later on.
4. Also the number of nan values for `End Date / Time` column is quite high so for this reason I will choose `Dispatch Date / Time` instead
5. I will exclude missing latitude, longitude values from our dataset

In [31]:

```
#exclude lat&lon missing values
lat_null=crimes['Latitude'].notnull()
lon_null=crimes['Latitude'].notnull()
crimes=crimes[(lat_null==True) & (lon_null==True)]
```

Time analysis

The aim here is to spot time related patterns in crimes. Time analysis is structured around these questions:

1. What day of the week are the most crimes committed on? (i.e Monday, Tuesday, etc)
2. During what time of day are the most crimes committed?
3. During what month are the most crimes committed?

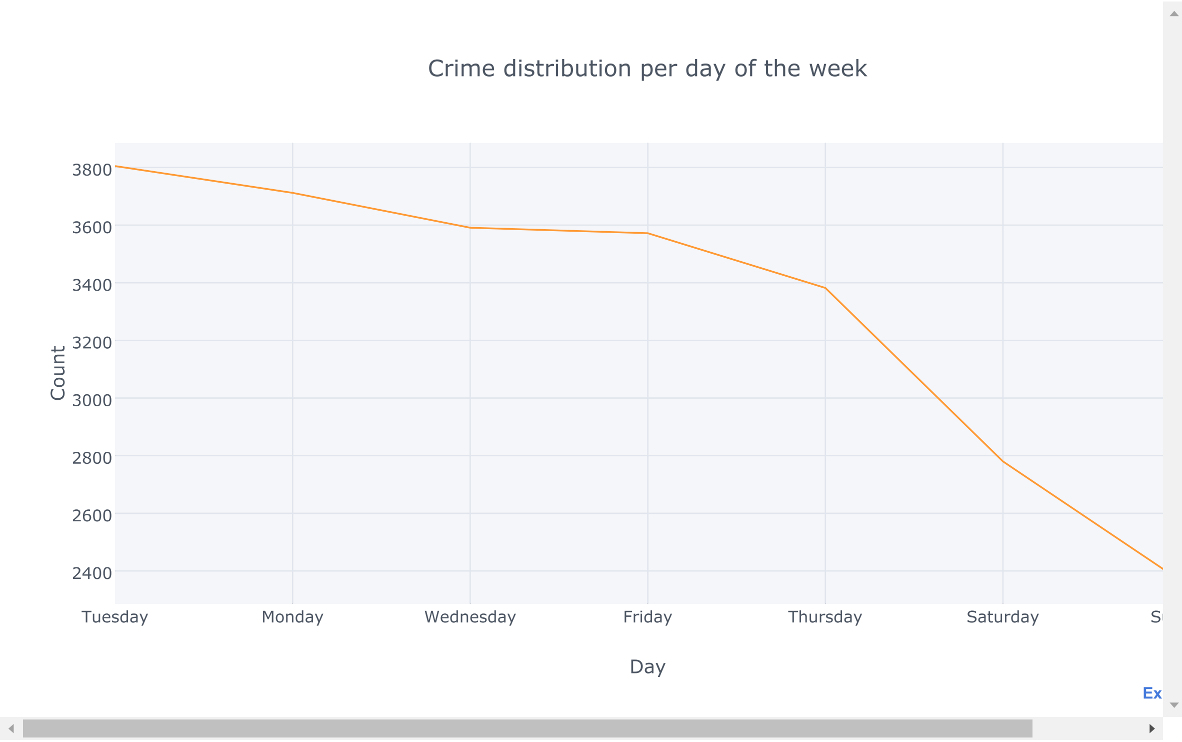
First step would be to convert the `Dispatch Date / Time` column from an object type to a datetime type.
Dispatch Date/Time--The actual date and time a Officer was dispatched

In [32]:

```
# import datetime modules
import datetime as dt

# convert 'Dispatch Date / Time' to datetime format
crimes['Dispatch Date / Time']=pd.to_datetime(crimes['Dispatch Date / Time'])

# get crime counts/weekday;hour;month
crimes['Dispatch Day of the Week']=crimes['Dispatch Date / Time'].dt.weekday_name
crimes['Dispatch Hour']=crimes['Dispatch Date / Time'].dt.hour
crimes['Dispatch Month']=crimes['Dispatch Date / Time'].dt.month
```

In [33]:

```
# import plotting modules

from plotly.offline import download_plotlyjs, init_notebook_mode, iplot
from plotly.graph_objs import *
init_notebook_mode()
import cufflinks as cf

# day of the week crime counts
dow=crimes['Dispatch Day of the Week'].value_counts().copy()

# Converting to a series to a dataframe
# It will be easier for plottling
pd.DataFrame({'Dispatch Day of the Week':dow.index,'Counts':dow}).reset_index(drop=True)

# plotting
dow.iplot(theme='pearl', filename='crime_distrib_per_day', title='Crime distribution per day of the week',
         xTitle='Day', yTitle='Count', world_readable=False)
dow
```



Out[33]:

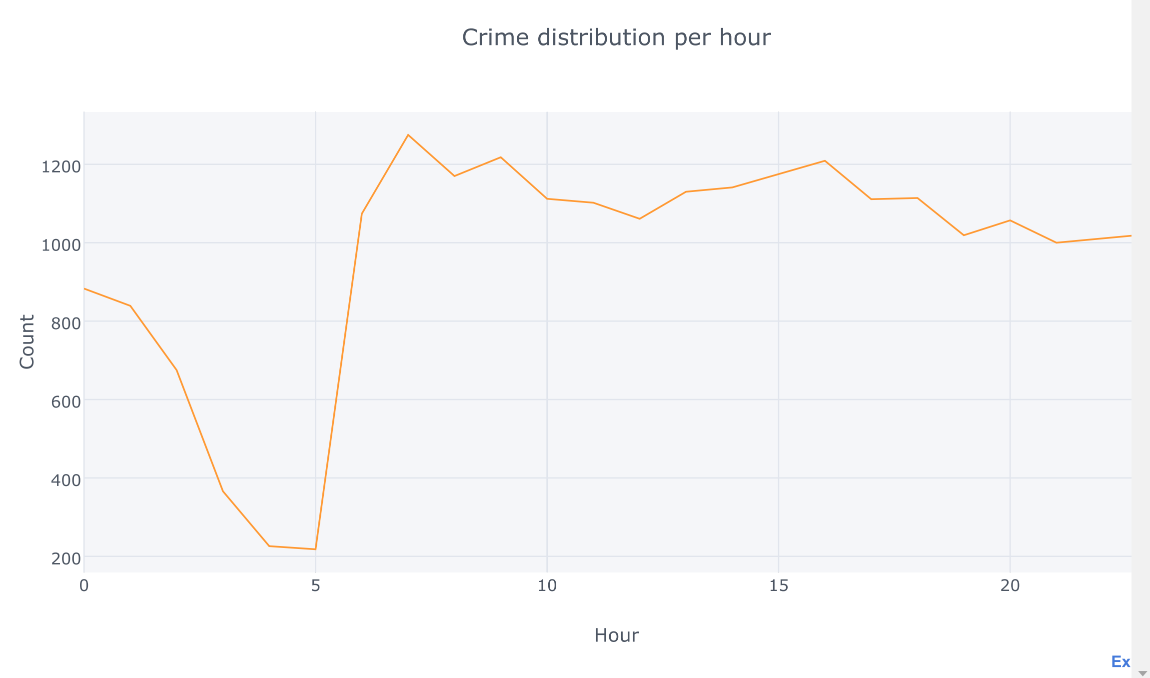
```
Tuesday      3805
Monday       3712
Wednesday    3591
Friday       3572
Thursday     3382
Saturday     2780
Sunday       2366
Name: Dispatch Day of the Week, dtype: int64
```

Day of the week analysis:

1. Generally it seems that less crimes are committed towards the weekend and highest counts are around the beginning of the week
2. Crime peak is Tuesday and lowest crimes are on Sunday.

In [34]:

```
# hour crime counts
tod=crimes['Dispatch Hour'].value_counts()

# Converting to a series to dataframe
pd.DataFrame({'Dispatch Hour':tod.index,'Counts':tod}).reset_index(drop=True)

# Pandas dataframe needs to be sorted by index for our time analysys otherwise the graph will be scrambled.
# This is because df will be sorted by values
tod.sort_index().iplot(theme='pearl', filename='crime_distrib_per_hour', title='Crime distribution per hour',
         xTitle='Hour', yTitle='Count', world_readable=False)

tod
```



Out[34]:

```
7     1275
9     1218
16    1209
15    1176
8     1170
14    1141
13    1130
18    1114
10    1112
17    1111
11    1102
6     1074
12    1061
20    1057
23    1022
19    1019
22    1010
21    1000
0      883
1      839
2      675
3      366
4      226
5      218
Name: Dispatch Hour, dtype: int64
```

Crime analysis by hour:

1. Highest value is a 7 am and lowest is at 5 am.
2. Most of the crimes are committed between 7 am and 11 pm

In [35]:

```
# crime distribution/month
tom=crimes['Dispatch Month'].value_counts()

# convert series to df
pd.DataFrame({'Dispatch Month':tom.index,'Counts':tom}).reset_index(drop=True)

# sort index & plot
tom.sort_index().iplot(theme='pearl', filename='crime_distrib_per_month', title='Crime distribution per month',
         xTitle='Month', yTitle='Count', world_readable=False)

tom
```

Out[35]:

```
10    4045
8     3977
11    3913
9     3898
12    3874
7     3501
Name: Dispatch Month, dtype: int64
```

Month crime analysis:

1. Data is not complete, we only have the statistics from July till the end of the year.
2. July month has the least crimes committed followed by a substantial increase in crimes starting from August.
3. I can attribute this to vacation time. Most probably by the end of August most people return from vacation. Peak is in October

Dispatch time interval analysis

Dispatch time could be a strong indicator as to how does police prioritize crimes.

1. To do this we need to see the general time difference between `Dispatch Date / Time` and `Start Date / Time`
2. This will be useful to determine and categorize the dispatch time intervals
3. Also it would be interesting to see if there is any difference in dispatch time based on the type of incidents

In [36]:

```
# convert to datetime
crimes['Start Date / Time']=pd.to_datetime(crimes['Start Date / Time'])

# get difference between 'Dispatch Date / Time' 'Start Date / Time'
crimes['Date diff']=crimes['Dispatch Date / Time'] -crimes['Start Date / Time']
pd.DataFrame(crimes.groupby(['Start Date / Time','Dispatch Date / Time'])['Date diff'].value_counts().head())
```

Out[36]:

			Date diff
Start Date / Time	**Dispatch Date / Time**	**Date diff**	
1974-10-30 00:00:00	**2013-11-07 11:41:23**	**14253 days 11:41:23**	1
1977-02-11 00:00:00	**2013-08-14 17:05:22**	**13333 days 17:05:22**	1
1980-08-24 00:00:00	**2013-12-23 10:28:40**	**12174 days 10:28:40**	1
1985-01-01 06:00:00	**2013-09-25 20:27:18**	**10494 days 14:27:18**	1
1993-08-27 00:00:00	**2013-11-18 17:45:43**	**7388 days 17:45:43**	1

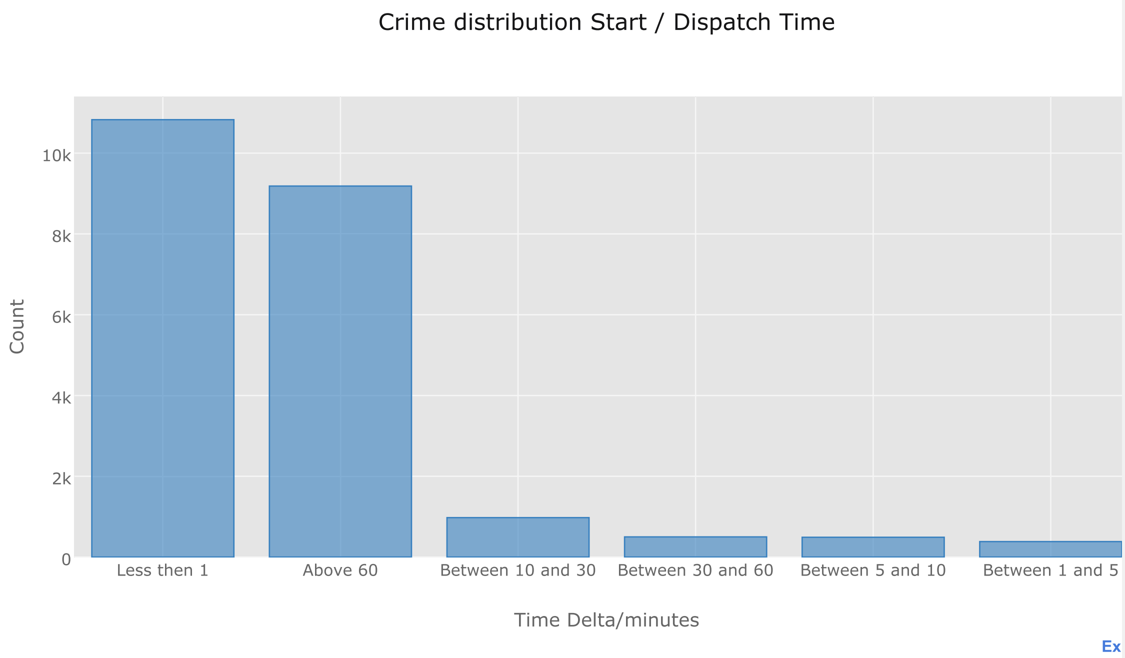
1. As we can see start time column is not accurate because a lot of dates that have the start time before 2013:
2. The data sample we are analyzing provides a summary of incidents from 07/2013 till 12/2013
3. So we will have to ignore those rows from out analysis

In [37]:

```
# Exclude records where start time was before 2013/07/01
crimes_slice=crimes[(crimes['Start Date / Time'] >= dt.date(2013,7,1))].copy()

# group Start/Dispatch Time on minutes intervals [<1,(1,5),(5,10),(10,30),(30,60),>60]
crimes_slice.loc[crimes_slice['Date diff']<dt.timedelta(minutes=1),'Less then 1']='True'
crimes_slice.loc[((crimes_slice['Date diff']>dt.timedelta(minutes=1))& (crimes_slice['Date diff']<dt.timedelta(minutes=5))),'Between
crimes_slice.loc[((crimes_slice['Date diff']>dt.timedelta(minutes=5)) & (crimes_slice['Date diff']<dt.timedelta(minutes=10))),'Betwee
crimes_slice.loc[((crimes_slice['Date diff']>dt.timedelta(minutes=10)) & (crimes_slice['Date diff']<dt.timedelta(minutes=30))),'Betwe
crimes_slice.loc[((crimes_slice['Date diff']>dt.timedelta(minutes=30)) & (crimes_slice['Date diff']<dt.timedelta(minutes=60))),'Betwe
crimes_slice.loc[crimes_slice['Date diff']>dt.timedelta(minutes=60),'Above 60']='True'
```

In [38]:

```
# Slicing the df on the time interval collumns & get the counts
td=crimes_slice.loc[:,['Less then 1',
                    'Between 1 and 5',
                    'Between 5 and 10',
                    'Between 10 and 30',
                    'Between 30 and 60',
                    'Above 60']].apply(pd.Series.value_counts)


#.iloc[0]  select that row positionally using iloc, which gives you a Series with the columns as the new index and values,
# then sorting by values
td.iloc[0].sort_values(ascending=False).iplot(kind='bar', title='Crime distribution Start / Dispatch Time',
         xTitle='Time Delta/minutes', yTitle='Count',color='rgba(55, 128, 191, 1.0)',filename='cufflinks/bar-chart-row')

td.iloc[0].sort_values(ascending=False)
```



Out[38]:

```
Less then 1          10828
Above 60              9185
Between 10 and 30      977
Between 30 and 60      500
Between 5 and 10       491
Between 1 and 5        384
Name: True, dtype: int64
```

Dispatch time interval breakdown:

1. The majority of the dispatch time happened quite fast(less then a minute). However this could be misleading as there isn't a clear description of what this column represent. I would tend to believe that `Dispatch Date / Time` represents the time when an officer was sent to a crime location not when it arrived at the crime scene.
2. Next would be the response time above 60 minutes.
3. Also it's important to remember that this is only a slice of the dataset as some start time might have not been accurate.
4. Researching the internet I saw various sources and it seems that response time varies between 6 and 15 minutes for critical issues. However I could not find an universal SLA for dispatching police officers.
5. Also another point worth taking into consideration that not all incidents require the same attention and some of the could be resolved over the phone.
6. The same analysis would be interesting when diving further into our analysis and classifying the crimes in violent/non violent ones.

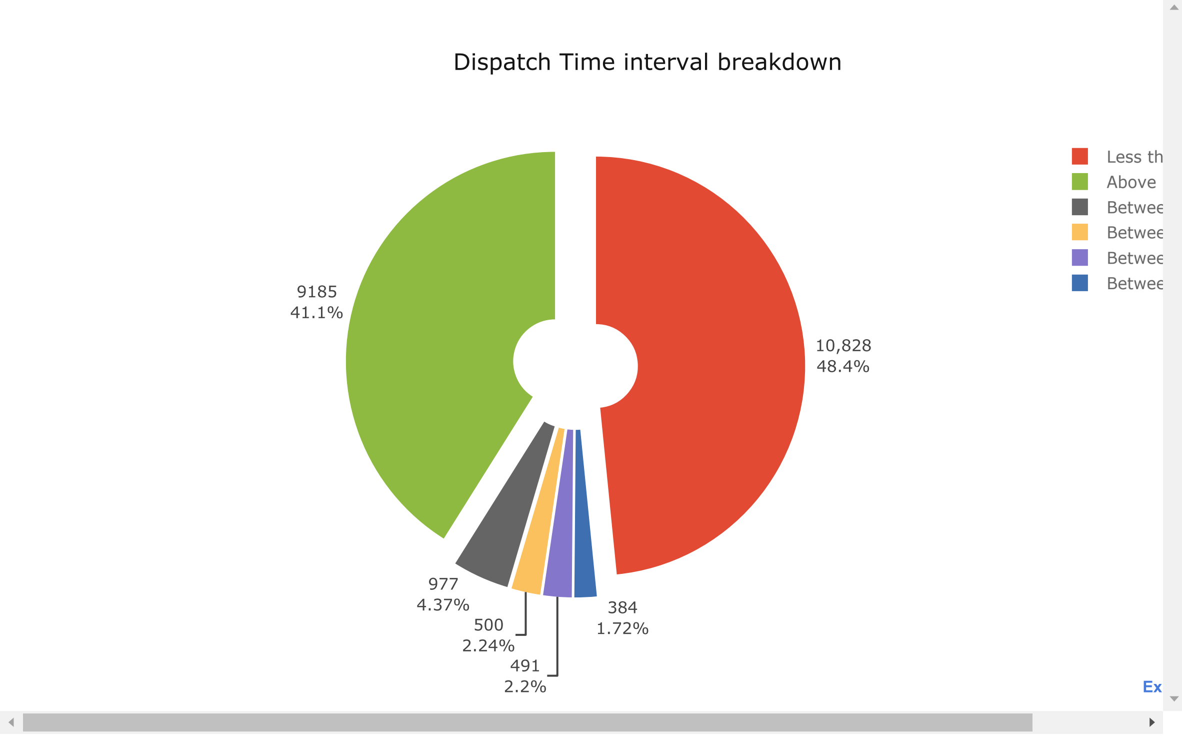
Dispatch Time Delta analysis

1. The bar chart above is not to revealing in term of quantifying the timedeltas
2. If we want to do that we will need to use a pie chart. For simplicity I will have to convert the series to a dataframe format for a pie chart with cufflinks. See below: https://plot.ly/pandas/pie-charts/ (https://plot.ly/pandas/pie-charts/)

   pie format

In [39]:

```
# convert series to dataframe
td_pie=td.iloc[0].to_frame()

# reset index, if column is in index it can't be used to plot
td_pie=td_pie.reset_index()

# rename columns, this step is not necessary
td_pie.columns=['Time Deltas','Values']

# plot
td_pie.iplot(kind='pie',labels='Time Deltas',values='Values',pull=.1,hole=.2,title='Dispatch Time interval breakdown',
                    textposition='outside',textinfo='value+percent')
```



Dispatch time intervals percentage breakdown:

1. dispatch time < 1 min 48.4%
2. dispatch time > 1 min & < 5 min 1.72%
3. dispatch time > 5 min & < 10 min 2.2%
4. dispatch time > 10 min & < 30 min 4.37%
5. dispatch time > 30 min & < 60 min 2.24%
6. dispatch time > 60 min 41%

Analyzing crime locations:

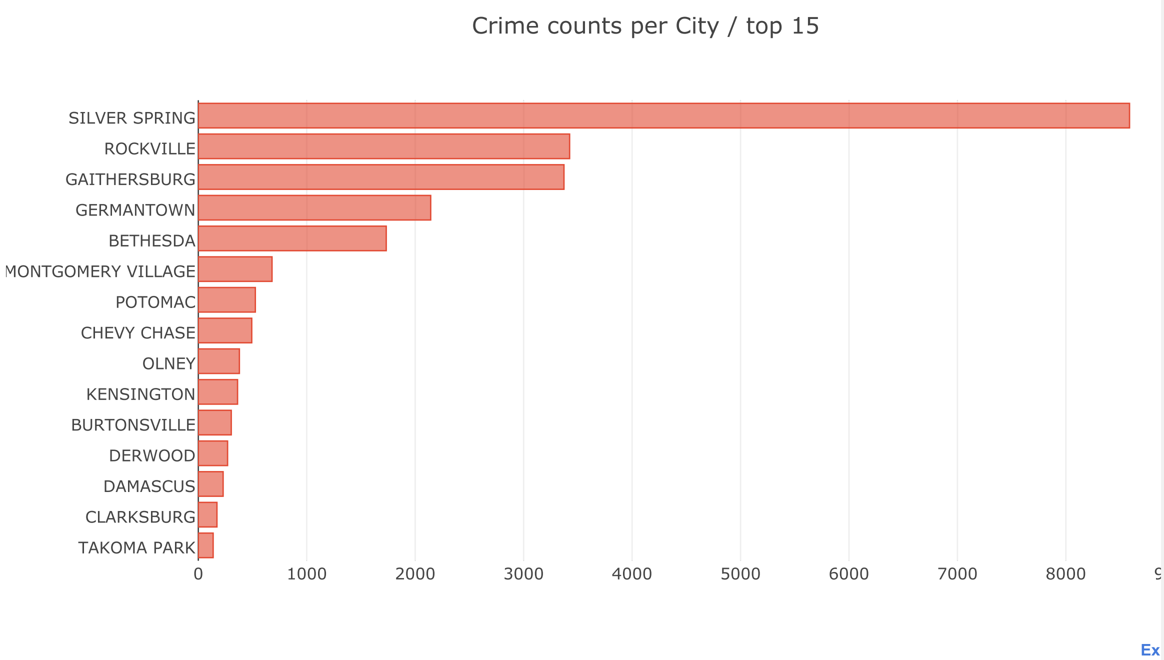
1. Will use the following criteria for choosing columns:
    A. Granularity: Small areas shouldn't be used, because only a few crimes were committed inside them, which makes it hard to analyze and compare
    B. Comprehensibility: Need to analyze data that is compelling for the casual reader
    C. Missing values: If a column has a lot of missing values, that means that the conclusions you draw are less valid, because you don't know if the missing data is systematic

1. Columns used:
    A. Police District Number | Major Police Boundary corresponding to Police District Names i.e (Rockville,Weaton etc.)
    B. City | City

Crime distribution by City

In [40]:

```
# crime counts per City, transform to df; sort

crimes_city=crimes['City'].value_counts().to_frame().sort_values(by='City')

# graph layout
layout=dict(autosize= True,
            title = 'Crime counts per City / top 15',
            xaxis=dict(domain=[0.08, 1]),
            )

# plot
crimes_city.tail(15).iplot(kind='barh',barmode='normal', bargap=.8, filename='cufflinbarh',layout=layout)
crimes_city.sort_values(by='City',ascending=False).head(10)
```



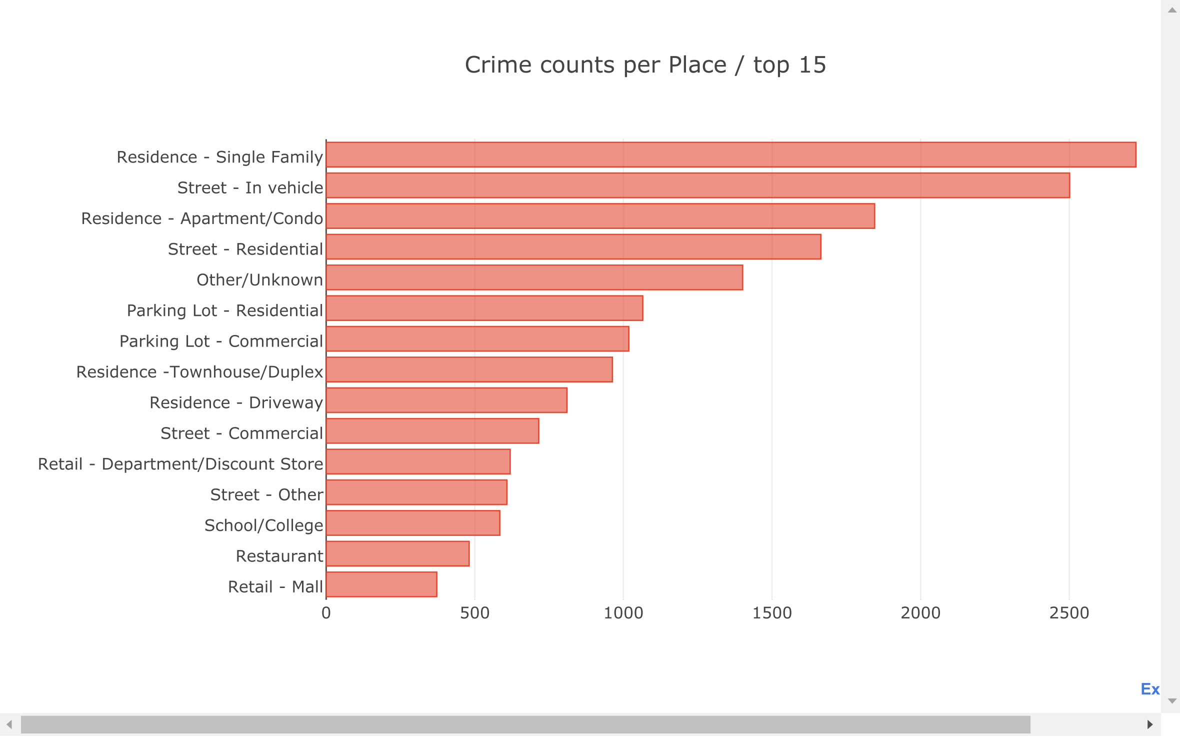
Out[40]:

	City
SILVER SPRING	8587
ROCKVILLE	3424
GAITHERSBURG	3372
GERMANTOWN	2143
BETHESDA	1733
MONTGOMERY VILLAGE	680
POTOMAC	526
CHEVY CHASE	493
OLNEY	379
KENSINGTON	362

Crime distribution by location

In [41]:

```
# get crime counts/place; transform to df
crimes_place=crimes['Place'].value_counts().to_frame().sort_values(by='Place')

# graph layout
layout=dict(
            title = 'Crime counts per Place / top 15',
            xaxis=dict(domain=[0.2, 1]),
            yaxis=dict(domain=[0.1, 0,66])
)

# plot
crimes_place.tail(15).iplot(kind='barh',barmode='normal', bargap=.4, filename='cufflinbarh',layout=layout)
crimes_place.sort_values(by='Place',ascending=False).head(10)
```

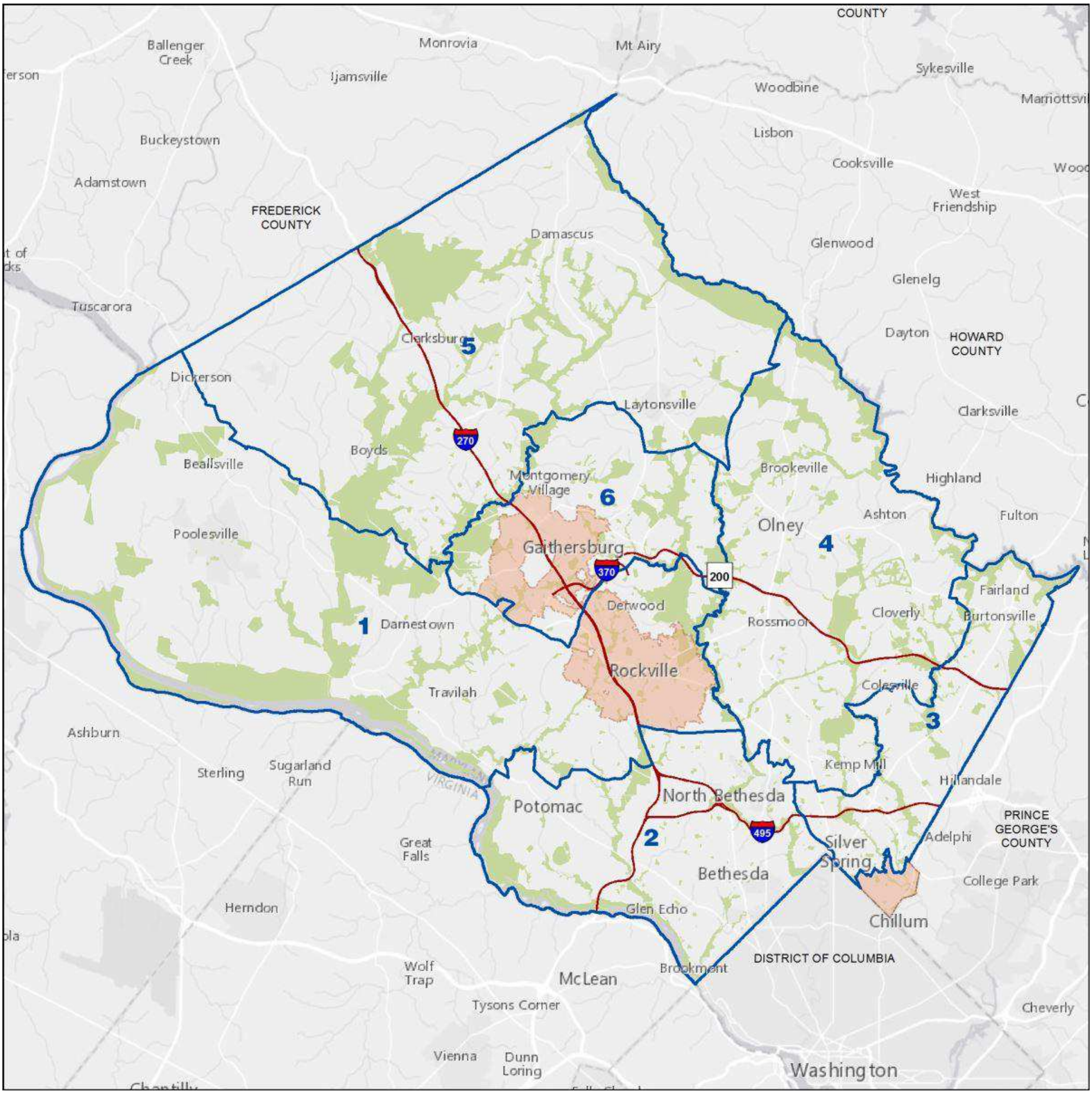


Out[41]:

	Place
Residence - Single Family	2724
Street - In vehicle	2501
Residence - Apartment/Condo	1845
Street - Residential	1664
Other/Unknown	1401
Parking Lot - Residential	1065
Parking Lot - Commercial	1018
Residence -Townhouse/Duplex	963
Residence - Driveway	810
Street - Commercial	715

Analize crimes based on Police District Location

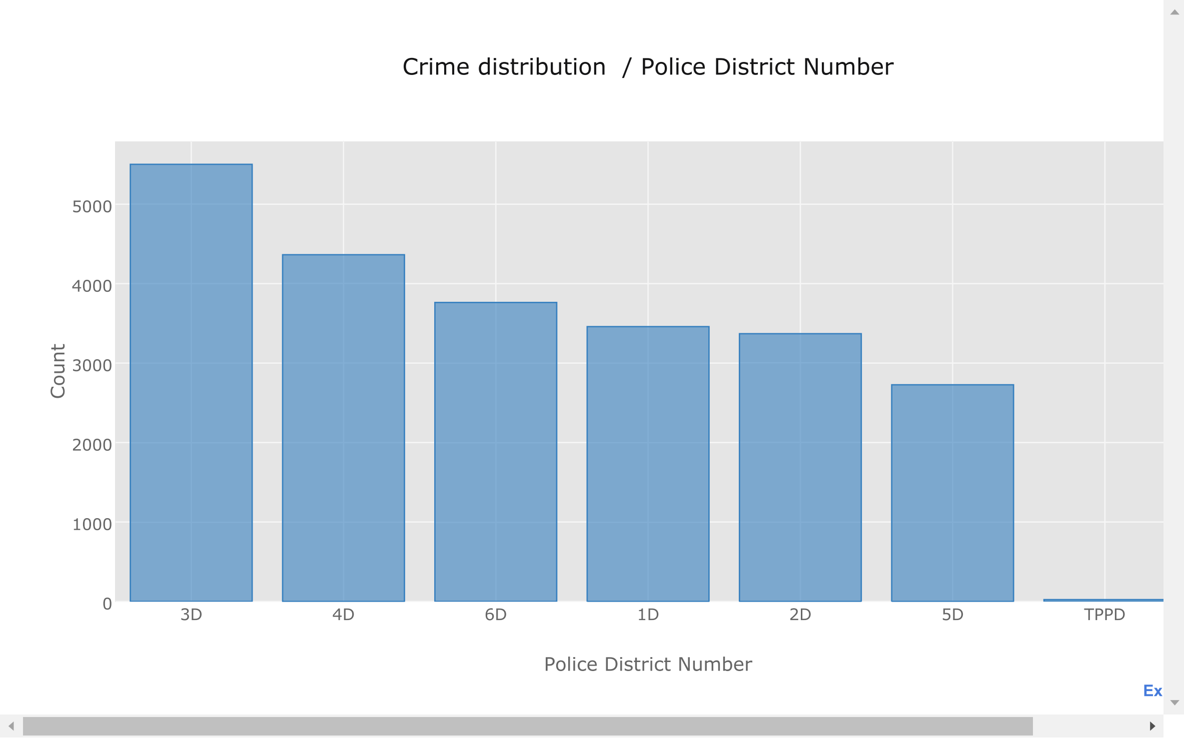
Although Police District does not mean much for the casual reader it could provide useful insight into our data analysis as the area are quite big an would provide great granularity.
Let's visualize Police District Location:



PD summary:
1D | Rockville
2D | Bethesda
3D | Silver Spring
4D | Wheaton
5D | Germantown
6D | Gaitersburg

In [42]:

```
# plot crime distribution/ PD
crimes['Police District Number'].value_counts().iplot(kind='bar',title='Crime distribution  / Police District Number',
       xTitle='Police District Number', yTitle='Count',color='rgba(55, 128, 191, 1.0)', filename='crime_distribution_per_district')
```



Crime by PD:

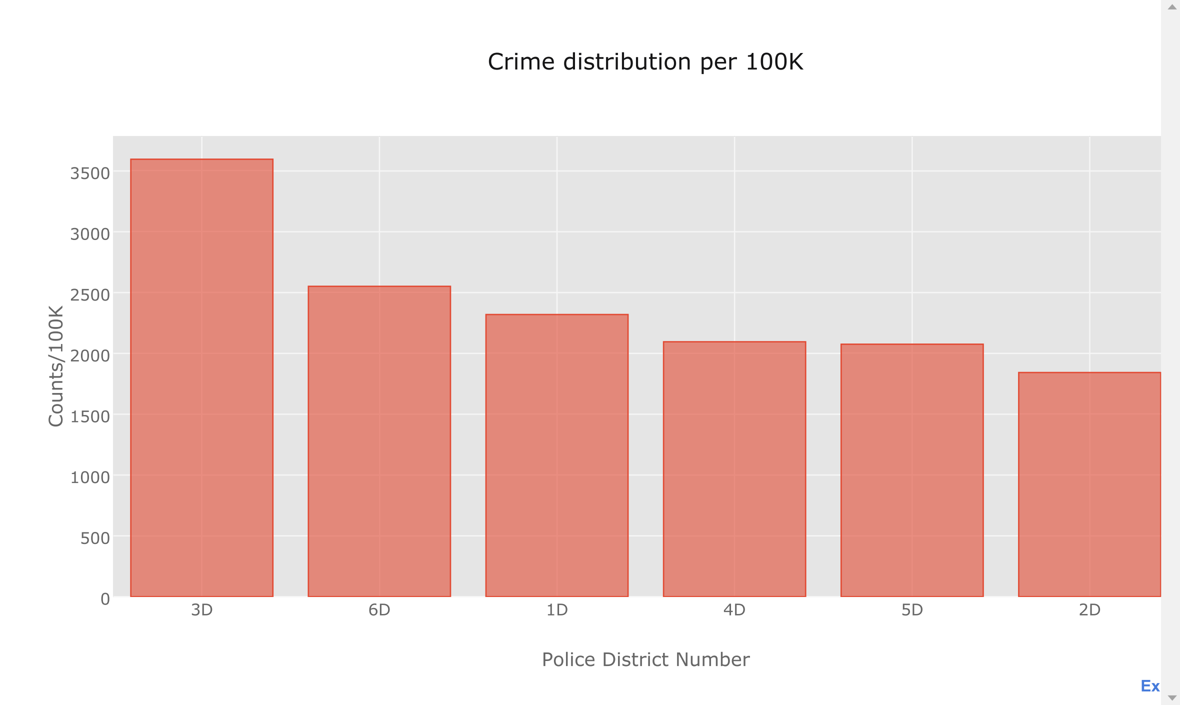
Silver Spring has the highest crime rates, and the lowest crime rates are in Germantown. Because of the lack of data we will exclude the last district i.e TPPD from our analysis
This is the order of the crimes counts / per district from the highest to lowest:

1. `Silver Spring; Wheaton; Gaithersburg; Rockville; Bethesda; Germantown`

But this would not be an accurate analysis unless we take into account the census data for these districts.
For this I will use the census data for Montgomery county Police Department. See below link at the end there is census data per each district.
https://www.wau.edu/wp-content/uploads/2012/09/MCPCrimeReport2014.compressed.pdf (https://www.wau.edu/wp-content/uploads/2012/09/MCPCrimeReport2014.compressed.pdf)

In [43]:

```
# Constructing a pandas dataframe with Police District Number & Population
census = pd.DataFrame([{'Police District Number':'1D','Population':'149118'},
                      {'Police District Number':'2D','Population':'182883'},
                      {'Police District Number':'3D','Population':'152991'},
                      {'Police District Number':'4D','Population':'208263'},
                      {'Police District Number':'5D','Population':'131391'},
                      {'Police District Number':'6D','Population':'147486'}]
                     )

# crime counts / Police District Number
cpd=crimes['Police District Number'].value_counts()

# Change the series to a df object with PD as index
cpd=pd.DataFrame({'Police District Number':cpd.index,'Counts':cpd}).reset_index(drop=True)

# Merge the 2 dfs
result=pd.merge(census, cpd, on='Police District Number')

# Compute the crime counts pe 100k,
result['Counts/100k']=100000*result['Counts']/pd.to_numeric(result['Population'], errors='coerce')

# Set Police district number as index,
# this is needed to have on the X axis the Police District Number, then get the value_count()
result.set_index('Police District Number',inplace=True)
result['Counts/100k'].sort_values(ascending=False).iplot(kind='bar',title='Crime distribution per 100K',xTitle='Police District Numbe
```



Crime by PD/100k:

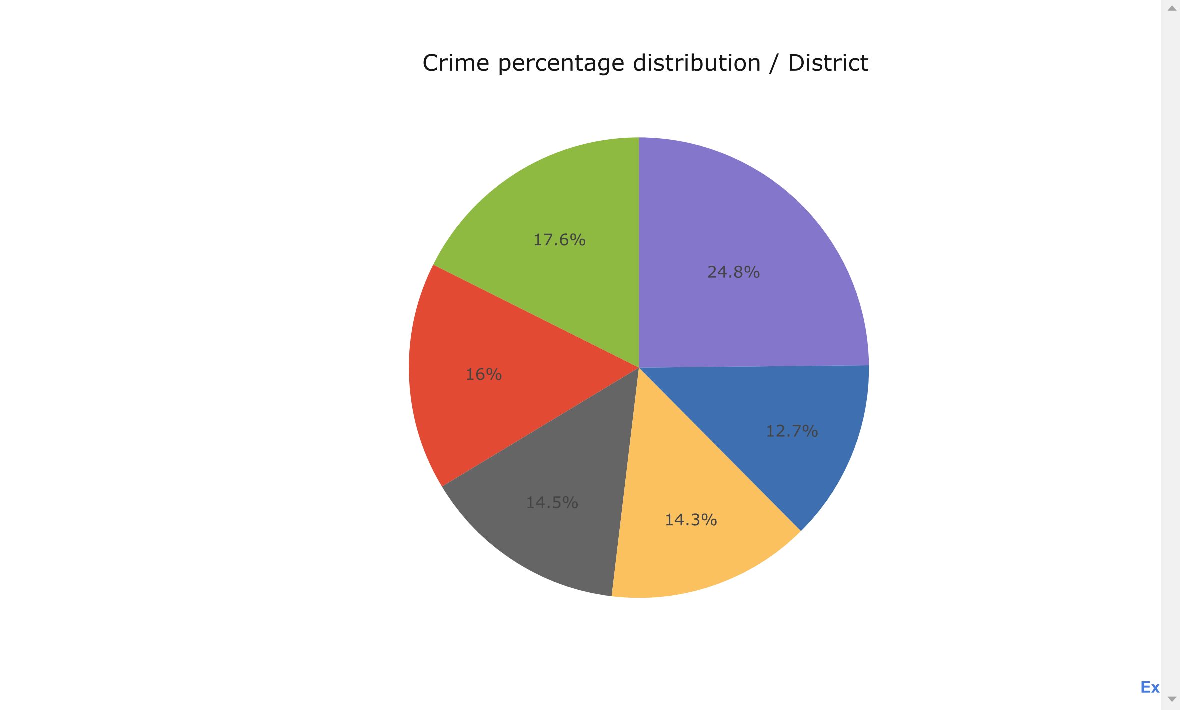
Silver Spring is still the district with the highest crime rates, followed by Gaithersburg and at the end Bethesda.
This is the order from highest to the lowest:

1. Silver Spring; Gaithersburg; Rockville; Wheaton; Germantown; Bethesda

Crime distribution/PD vizualization pie

In [44]:

```
#Getting the crime percentage per district
result['Percentage']=result['Counts']*100/pd.to_numeric(result['Population'], errors='coerce')

#Need to reset the index otherwise I can't plot a pandas df and use one column if that column is in the index
result.reset_index().iplot(kind='pie',labels='Police District Number',values='Percentage',title="Crime percentage distribution / Dist
result
```



Out[44]:

	Population	Counts	Counts/100k	Percentage
Police District Number				
1D	149118	3459	2319.639480	2.319639
2D	182883	3370	1842.708180	1.842708
3D	152991	5502	3596.289978	3.596290
4D	208263	4364	2095.427416	2.095427
5D	131391	2727	2075.484622	2.075485
6D	147486	3763	2551.428610	2.551429

Crime percentage breakdown/PD:

- 3D - Silver Spring 24.8%
- 6D - Gaitersburg 17.6%
- 1D - Rockville 16%
- 4D - Wheaton 14.5%
- 5D - Germantown 14.3%
- 2D - Bethesda 12.7%

Crime analysis by City & PD

In [45]:

```
# group crimes based on Police District Number / City; get the value_counts
pd_city_series=crimes.loc[:,['Police District Number','City']].groupby('Police District Number')['City'].apply(lambda s: s.value_cour

# get the names of cities that appear on different PDs
# convert series to frame; reset index; rename the columns
pd_city=pd_city_series.to_frame()
pd_city.reset_index(inplace=True)
pd_city.columns=['Police District Number','City','Counts']
pd_city_series
```

Out[45]:

```
Police District Number
1D                        ROCKVILLE            2375
                          GAITHERSBURG          398
                          POTOMAC               365
                          DERWOOD               154
                          POOLESVILLE           104
                          GERMANTOWN             37
                          DICKERSON              19
                          BOYDS                   4
                          BEALLSVILLE             2
                          SILVER SPRING           1
2D                        BETHESDA             1733
                          ROCKVILLE             531
                          CHEVY CHASE           492
                          KENSINGTON            274
                          POTOMAC               161
                          SILVER SPRING         157
                          CABIN JOHN             18
                          GLEN ECHO               4
3D                        SILVER SPRING        5075
                          BURTONSVILLE          304
                          TAKOMA PARK           118
                          SPENCERVILLE            3
                          LAUREL                  1
                          CHEVY CHASE             1
4D                        SILVER SPRING        3350
                          ROCKVILLE             410
                          OLNEY                 375
                          KENSINGTON             88
                          BROOKEVILLE            68
                          SANDY SPRING           42
                          ASHTON                 19
                          BRINKLOW                5
                          SPENCERVILLE            5
                          GAITHERSBURG            2
5D                        GERMANTOWN           2103
                          DAMASCUS              229
                          CLARKSBURG            172
                          GAITHERSBURG          117
                          BOYDS                  86
                          DICKERSON               7
                          BARNESVILLE             4
                          MOUNT AIRY              3
                          DERWOOD                 2
                          BROOKEVILLE             2
                          POOLESVILLE             1
                          MONTGOMERY VILLAGE      1
6D                        GAITHERSBURG         2855
                          MONTGOMERY VILLAGE    679
                          DERWOOD               114
                          ROCKVILLE             108
                          OLNEY                   4
                          GERMANTOWN              3
TPPD                      TAKOMA PARK            19
                          SILVER SPRING           4
Name: City, dtype: int64
```

One thing that stick out is that some cities appear on several districts. This could be due to crime locations being close to multiple PDs.
Let's do a map visualization

In [46]:

```
# get duplicated cities
pd_city_dup=pd_city[pd_city.duplicated('City')==True]['City'].unique()
pd_city_dup
```

Out[46]:

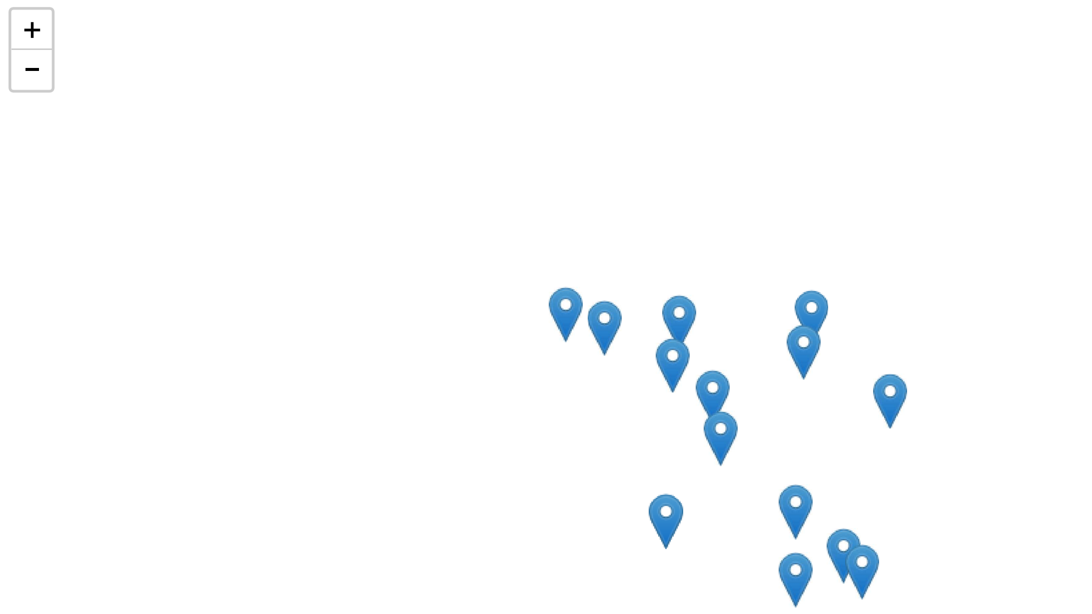
```
array(['ROCKVILLE', 'POTOMAC', 'SILVER SPRING', 'CHEVY CHASE',
       'KENSINGTON', 'SPENCERVILLE', 'GAITHERSBURG', 'GERMANTOWN', 'BOYDS',
       'DICKERSON', 'DERWOOD', 'BROOKEVILLE', 'POOLESVILLE',
       'MONTGOMERY VILLAGE', 'OLNEY', 'TAKOMA PARK'], dtype=object)
```

City map visualization

In [47]:

```
import folium

map_1 = folium.Map(location=[39.154743, -77.240515],
                   zoom_start=9.7,
                   tiles='Stamen Terrain')
folium.Marker([39.0838889, -77.1530556], popup='ROCKVILLE').add_to(map_1)
folium.Marker([39.0180556, -77.2088889], popup='POTOMAC').add_to(map_1)
folium.Marker([38.9905556, -77.0263889], popup='SILVER SPRING').add_to(map_1)
folium.Marker([38.9712215, -77.0763667], popup='CHEVY CHASE').add_to(map_1)
folium.Marker([39.0256651, -77.0763669], popup='KENSINGTON').add_to(map_1)
folium.Marker([39.1142747, -76.9783097], popup='SPENCERVILLE').add_to(map_1)
folium.Marker([39.1433333, -77.2016667], popup='GAITHERSBURG').add_to(map_1)
folium.Marker([39.1730556, -77.2719444], popup='GERMANTOWN').add_to(map_1)
folium.Marker([39.1837171, -77.3127623], popup='BOYDS').add_to(map_1)
folium.Marker([39.11733,  -77.1610916], popup='DERWOOD').add_to(map_1)
folium.Marker([39.1806623, -77.0591452], popup='BROOKEVILLE').add_to(map_1)
folium.Marker([39.0180556, -77.2088889], popup='POOLESVILLE').add_to(map_1)
folium.Marker([39.1766667, -77.1955556], popup='MONTGOMERY VILLAGE').add_to(map_1)
folium.Marker([39.1530556, -77.0672222], popup='OLNEY').add_to(map_1)
folium.Marker([38.9777778, -77.0077778], popup='TAKOMA PARK').add_to(map_1)

map_1
```

Out[47]:



Leaflet (http://leafletjs.com) | Map tiles by Stamen Design (http://stamen.com), under CC BY 3.0 (http://creativecommons.org/licenses/by/3.0). Data by OpenStreetMap (http://openstreetmap.org), under CC BY SA (http://creativecommons.org/licenses/by-sa/3.0).

Because it's not clear why a city name appears on multiple PDs, I will focus the analysis on other areas

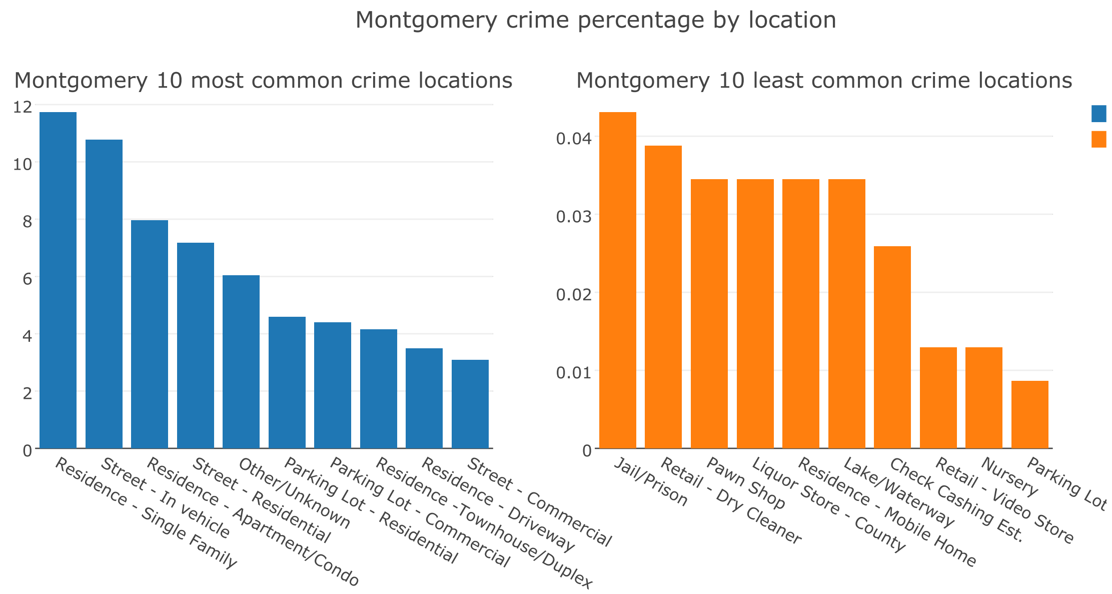
Analyze crime by location:

1. determine what are the most and least common crime locations(i.e residence,street) for Montgomery county and for each PD.
2. get 10 most common/least common locations

In [48]:

```
# import graph modules
import plotly.graph_objs as go
from plotly import tools

# Will get the total number of crimes committed
total=crimes['Place'].count()

# Get the value_counts() for all the places
place=crimes['Place'].value_counts()

# Convert the series to a df
place=pd.DataFrame({'Place':place.index,'Counts':place}).reset_index(drop=True)

# Get crime percentege on a given location.
place['Percent']=place['Counts']*100/total

# 10 most/least common
trace1=go.Bar(x=place['Place'].head(10),y=place['Percent'].head(10),name='most common')
trace2=go.Bar(x=place['Place'].tail(10),y=place['Percent'].tail(10),name='least common')

fig = tools.make_subplots(rows=1, cols=2, subplot_titles=('Montgomery 10 most common crime locations',
                                                          'Montgomery 10 least common crime locations'))

fig.append_trace(trace1,1,1)
fig.append_trace(trace2,1,2)

fig['layout'].update(height=500,width=1000,autosize=True,margin=go.Margin(b=135),title='Montgomery crime percentage by location')
iplot(fig, filename='stacked-subplots-shared-xaxes')
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
```



10 most common locations:

1. Residence (Single Family; Apartment; Townhouse/Duplex; Driveway)
2. Street(in Vehicle; Residential; Commercial)
3. Parking Lot (Commercial; Residential)

10 least common locations:

1. Jail
2. Retail(Dry Cleaner; Video Store)
3. Liquor Store
4. Lake
5. Pawn Shop
6. Residence(Mobile Home)
7. Nursery
8. Parking Lot(Park & Ride)

10 most/least common crime locations on PDs

In [49]:

```
# First I will create a df where I will group the df by 'Police District Number' chain it to 'Place' column and get
# the value_counts(), and create a new column called Counts
crimes_district_location=pd.DataFrame({'Counts' : crimes.groupby(['Police District Number'])['Place'].value_counts()}).reset_index()

def find_pcent(row):
    # Get the total crimes per district
    p_d_val_counts = crimes['Police District Number'].value_counts()

    # get a list of unique police district numbers and iterate over it
    for p_d in crimes['Police District Number'].unique():
        # for each Police District Number get the percentage
        if row['Police District Number'] == p_d:
            return (row['Counts'] / p_d_val_counts[p_d]) * 100

# Use an apply function to compute the percentage for each PD / location
crimes_district_location['%'] = crimes_district_location.apply(find_pcent, axis = 1)
crimes_district_location.head()
```

Out[49]:

	Police District Number	Place	Counts	%
0	1D	Residence - Single Family	509	14.715236
1	1D	Other/Unknown	303	8.759757
2	1D	Street - In vehicle	287	8.297196
3	1D	Street - Residential	201	5.810928
4	1D	Residence - Driveway	183	5.290546

Plotting functions

1. to make the code modular I decided to implement 3 plotting functions
2. These are the benefits:
   A. reduce the overall code
   B. put emphasis on the plot not the plotting code
   C. focus more on the actual data behind the plot

1. plot_bar_staked:

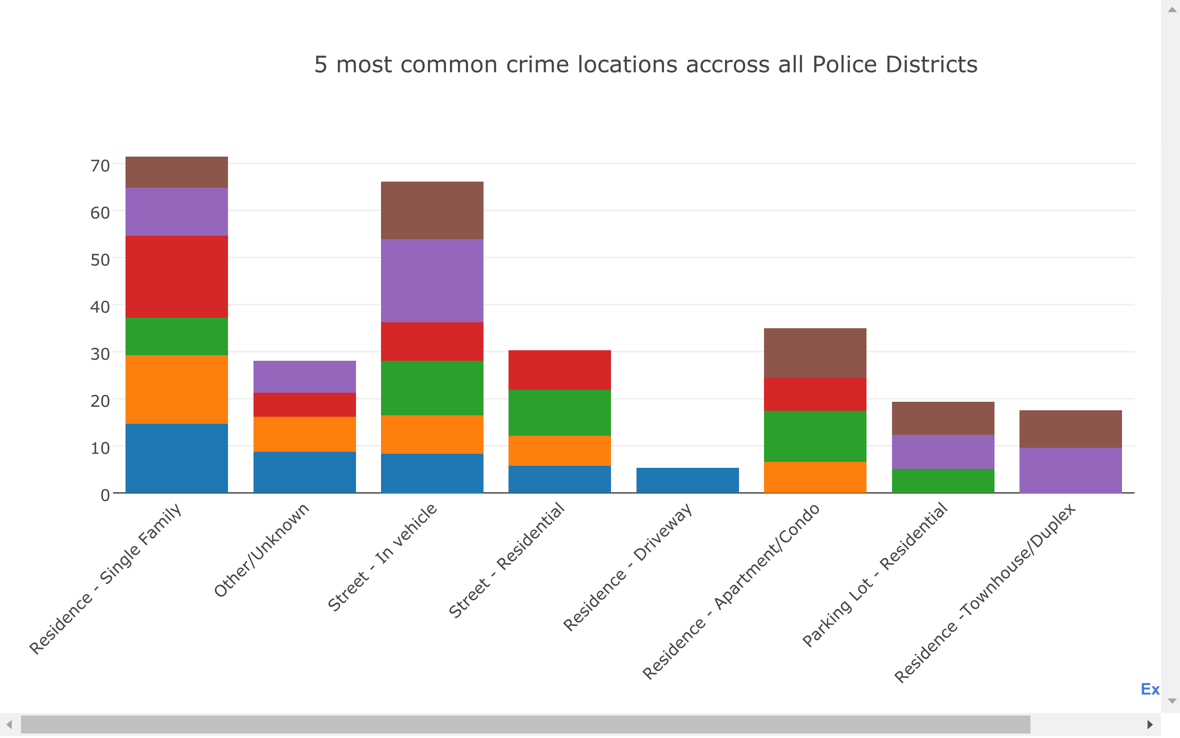
   A. Official doc: https://plot.ly/python/bar-charts/ (https://plot.ly/python/bar-charts/)
   B. plotting will be done based on a data list
   C. data is comprised of traces
   D. a trace is made out of:
      a. labels (x on the axis)
      b. values (y on the axis)
      c. name of the particular trace
   E. particular to this code we will plot data based on top/least values
   F. so that means we will have separate data and traces for each case

In [50]:

```
def plot_bar_staked(df,part):
    data_h=[]
    data_t=[]
    names=['PD_1D','PD_2D','PD_3D','PD_4D','PD_5D','PD_6D']
    for pd,n in zip(['1D','2D','3D','4D','5D','6D'],names):
        data_h+=['trace'+str(pd)]
        data_t+=['trace'+str(pd)]
        labels_h=crimes_district_location[crimes_district_location['Police District Number']==pd]['Place'].head()
        labels_t=crimes_district_location[crimes_district_location['Police District Number']==pd]['Place'].tail()
        values_h=crimes_district_location[crimes_district_location['Police District Number']==pd]['%'].head()
        values_t=crimes_district_location[crimes_district_location['Police District Number']==pd]['%'].tail()
        data_h[len(data_h)-1]=go.Bar(x=labels_h,y=values_h,name=n)
        data_t[len(data_t)-1]=go.Bar(x=labels_t,y=values_t,name=n)

    if part=='head':
        return data_h
    elif part=='tail':
        return data_t
```

In [51]:

```
# plotting most common locations
# 1) need to pass to the plotting function data for the analysis in this case 'crimes_district_location;
# 2) need to specify if are are looking for top/least common locations in this case top most i.e part='head'

df=crimes_district_location
part='head'
data=plot_bar_staked(df,part)

# create figure layout
layout = go.Layout(
    xaxis=dict(tickangle=-45),
    barmode='stack',
    autosize=True,
    margin=go.Margin(b=160),
    title='5 most common crime locations accross all Police Districts'
)

fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='angled-text-bar')
```

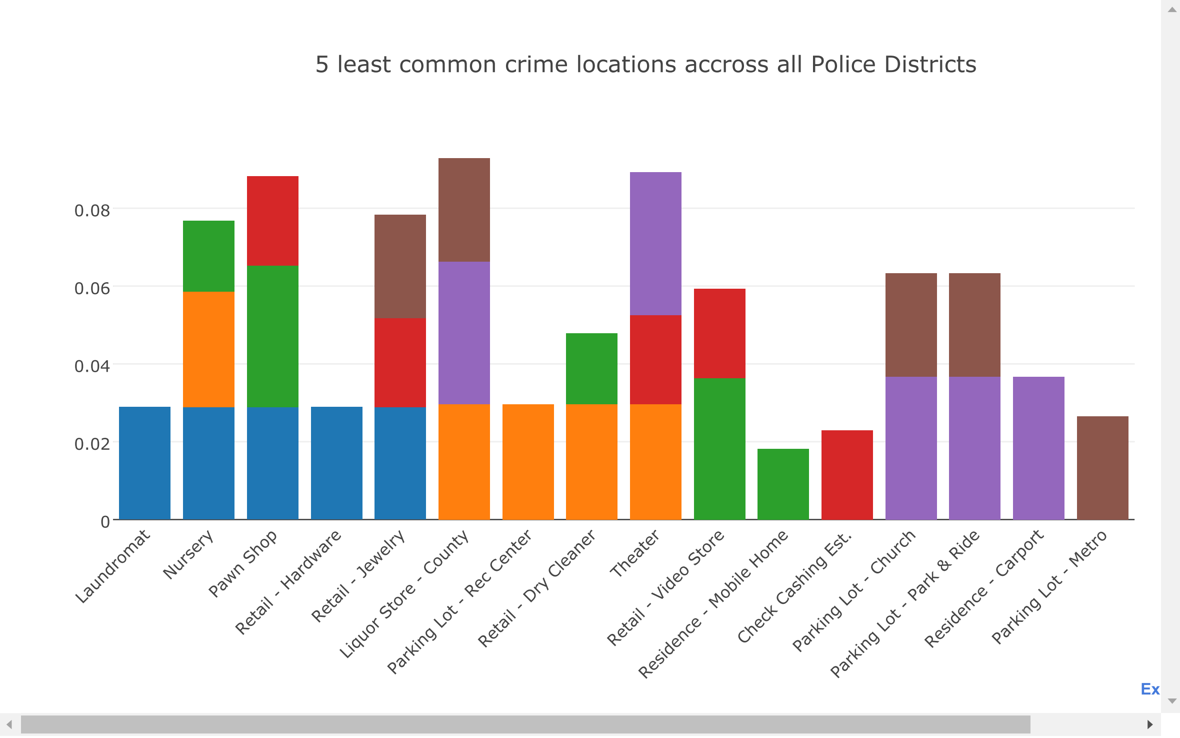


Most common crime locations:

1. Residence - Single Family
2. Street - In vehicle
3. Residence - Apartment/Condo
4. Street - Residential
5. Other
6. Parking Lot
7. Residence - Townhouse/Duplex
8. Residence - Driveway

In [52]:

```
# create plot trace for each district; get 5 least common locations
df=crimes_district_location
part='tail'
data=plot_bar_staked(df,part)

#create fig layout
layout = go.Layout(
    xaxis=dict(tickangle=-45),
    barmode='stack',
    autosize=True,
    margin=go.Margin(b=140),
    title='5 least common crime locations accross all Police Districts'
)

fig = go.Figure(data=data, layout=layout)
iplot(fig, filename='angled-text-bar')
```



Least common crime locations:

- Liquer Store
- Theater
- Pawn Shop
- Retail (Jewelery;Video Store; Dry Cleaner;Hardware)
- Nursery
- Parking Lot (Church;Park & Ride;Rec Center; Metro)
- Residence (Mobile Home; Carpot)
- Loundromat
- Check Cashing

Analize violent/non-violent crimes:

According to the UCR(Uniform Crime Reporting) definition violent crimes are the following:
"The descending order of UCR violent crimes are murder and nonnegligent manslaughter, forcible rape, robbery, and aggravated assault, followed by the property crimes of burglary, larceny-theft, and motor vehicle theft. Although arson is also a property crime, the Hierarchy Rule does not apply to the offense of arson. In cases in which an arson occurs in conjunction with another violent or property crime, both crimes are reported, the arson and the additional crime.".

The goal is to:

```
1. get a percentage breakdown between violent and non-violent crimes
2. percentage breakdown for violent subcategories
3. analyze dispatch time for above mentioned cases
```

Columns used:
Class | Four digit code identifying the crime type of the incident
Class Description | Common name description of the incident class type

Official reference: https://ucr.fbi.gov/crime-in-the-u.s/2011/crime-in-the-u.s.-2011/violent-crime/violent-crime (https://ucr.fbi.gov/crime-in-the-u.s/2011/crime-in-the-u.s.-2011/violent-crime/violent-crime) alt text

Violent crime class analysis

In [53]:

```
# Set pandas to display absolutly all the rows. Usually pandas prints just the beginning and the last part of the df
import pandas as pd
pd.set_option('display.max_rows', None)
pd.options.display.float_format = '{:20,.2f}'.format

# Slice the df on'Class' and 'Class Description'
violent_crimes=crimes.loc[:,['Class','Class Description']]

# get unique class values & sort ascending
print(violent_crimes.sort_values(by='Class').drop_duplicates())

# Reset pandas display max rows to default value
pd.reset_option('display.max_rows')
```

```
       Class                       Class Description
17839    111                        HOMICIDE-FIREARM
12606    115                          HOMICIDE-OTHER
4379     211                              RAPE-FORCE
5508     212                  RAPE - ATTEMPT - FORCE
16701    311                    ROB FIREARM - STREET
12576    312                ROB FIREARM - COMMERCIAL
6594     313               ROB FIREARM - GAS/SVC STA
19925    314               ROB FIREARM - CONV. STORE
16228    315               ROB FIREARM - RESIDENTIAL
14217    316     ROB FIREARM - FINANCIAL INSTITUTION
4358     317                     ROB FIREARM - OTHER
1593     318                     ROB FIREARM CARJACK
13358    321             ROB KNIFE/CUT INST - STREET
11292    322               ROB KNIFE/CUT INST - COMM
15268    324             ROB KNIFE/CUT - CONV. STORE
9334     325        ROB KNIFE/CUT INST - RESIDENTIAL
7613     327               ROB KNIFE/CUT INS - OTHER
4818     328                   ROB KNIFE CUT CARJACK
1897     331               ROB OTHER WEAPON - STREET
2446     332                 ROB OTHER WEAPON - COMM
3577     333          ROB OTHER WEAPON GAS/SVC  STA
18246    335                  ROB OTHER WEAPON - RES
5743     337                ROB OTHER WEAPON - OTHER
17234    341                   ROB STNG ARM - STREET
23352    342                     ROB STNG ARM - COMM
17190    343               ROB STN ARM - GAS/SVC STA
4505     344               ROB STN ARM - CONV. STORE
6613     345                      ROB STNG ARM - RES
20149    346    ROB STNG ARM - FINANCIAL INSTITUTION
6051     347                    ROB STNG ARM - OTHER
7319     348                  ROB STRONG ARM CARJACK
18945    411               AGG ASSLT FIREARM CITIZEN
20447    412                  AGG ASSLT FIREARM P.O.
1513     414         AGG  ASSLT FIREARM OTHR DOMESTIC
9273     421              AGG ASSLT CUT/STAB CITIZEN
17195    422                AGG  ASSLT CUT/STAB P.O.
1901     423       AGG ASSLT CUT/STAB SPOUSE/PARTNER
3217     424        AGG  ASSLT CUT/STAB OTHR DOMESTC
5961     431             AGG ASSLT OTHER WPN CITIZEN
4833     432                AGG ASSLT OTHER WPN P.O.
2097     433      AGG ASSLT OTHER WPN SPOUSE/PARTNER
11946    434         AGG ASSLT OTHER WPN OTHR DOMEST
21986    435          AGG ASSLT OTHER WPN ON ELDERLY
8435     441                 AGG ASSLT BEAT/INJ CTZN
20487    442                AGG  ASSLT BEAT/INJ P.O.
21938    443      AGG  ASSLT BEAT/INJ SPOUSE/PARTNER
2811     444           AGG ASSLT BEAT/INJ OTHR DOMES
8306     445              AGG ASSLT BEAT/INJ ELDERLY
21904    511                   BURG FORCE-RES/NIGHT
9745     512                     BURG FORCE-RES/DAY
14572    513                BURG FORCE-RES/TIME UNK
3127     514                  BURG FORCE-COMM/NIGHT
15144    515                    BURG FORCE-COMM/DAY
12840    516               BURG FORCE-COMM/TIME UNK
22045    517                   BURG FORCE-SCH/NIGHT
18201    518                     BURG FORCE-SCH/DAY
1830     519                BURG FORCE-SCH/TIME UNK
15471    521               BURG NO FORCE - RES/NIGHT
10578    522                 BURG NO FORCE - RES/DAY
8428     523            BURG NO FORCE - RES/TIME UNK
11792    524              BURG NO FORCE - COMM/NIGHT
11910    525                BURG NO FORCE - COMM/DAY
9541     526           BURG NO FORCE - COMM/TIME UNK
2102     527               BURG NO FORCE - SCH/NIGHT
14843    529            BURG NO FORCE - SCH/TIME UNK
7530     531           BURG FORCE - ATTEMPT- RES/NIGHT
4185     532             BURG FORCE - ATTEMPT-RES/DAY
4943     533        BURG FORCE - ATTEMPT-RES/TIME UNK
15926    534           BURG FORCE - ATTEMPT-COMM/NIGHT
12727    535           BURG FORCE - ATTEMPT -COMM/DAY
11748    536      BURG FORCE - ATTEMPT - COM/TIME UNK
5709     537         BURG FORCE - ATTEMPT - SCH/NIGHT
3329     611           LARCENY PICK POCKET OVER $200
7961     612          LARCENY PURSE SNATCH OVER $200
2138     613           LARCENY SHOPLIFTING OVER $200
```

13156	614	LARCENY FROM AUTO OVER $200
18548	615	LARCENY AUTO PART OVER $200
10569	616	LARCENY BICYCLE OVER $200
6738	617	LARCENY FROM BUILDING OVER $200
4701	618	LARCENY COIN MACH OVER $200
12520	619	LARCENY OTHER OVER $200
17215	621	LARCENY PICK POCKET $50 - $199
13648	622	LARCENY PURSE SNATCH $50 - $199
9895	623	LARCENY SHOPLIFTING $50 - $199
9445	624	LARCENY FROM AUTO $50 - $199
16347	625	LARCENY AUTO PART $50-$199
21280	626	LARCENY BICYCLE $50 - $199
19661	627	LARCENY FROM BUILDING $50-$199
260	628	LARCENY COIN MACH $50-$199
10985	629	LARCENY OTHER $50 - $199
20779	631	LARCENY PICK POCKET UNDER $50
9827	632	LARCENY PURSE SNATCH UNDER $50
5353	633	LARCENY SHOPLIFTING UNDER $50
12069	634	LARCENY FROM AUTO UNDER $50
5899	635	LARCENY AUTO PART UNDER $50
16023	636	LARCENY BICYCLE UNDER $50
6963	637	LARCENY FROM BLDG UNDER $50
3267	638	LARCENY COIN MACH UNDER $50
21043	639	LARCENY OTHER UNDER $50
18488	711	AUTO THEFT - PASSENGER VEHICLE
421	712	AUTO THEFT - TRUCKS & BUSES
3272	713	AUTO THEFT - OTHER VEHICLES
6421	811	ASSAULT & BATTERY - CITIZEN
8986	812	ASSAULT & BATTERY - POLICE OFFICER
18270	813	ASSAULT & BATTERY SPOUSE/PARTNER
17441	814	ASSAULT & BATTERY OTHER DOMESTIC
6393	815	ASSAULT & BATTERY - ELDERLY
5579	821	SIMPLE ASSAULT - CITIZEN
12510	822	SIMPLE ASSAULT - PO
505	823	SIMPLE ASSAULT SPOUSE/PARTNER
8893	824	SIMPLE ASSAULT OTHER DOMESTIC
12380	825	SIMPLE ASSAULT - ELDERLY
9977	911	ARSON- OCCUPIED STRUCTURE
17604	912	ARSON - UNOCCUPIED STRUCT/OTH
1268	913	ARSON MOTOR VEHICLE
17420	922	ARSON ATTEMPT UNOCCUPIED STRUCTURE/OTH PROP
4679	932	ARSON UNDER INVEST - UNOCCUP/OTHER
7086	933	ARSON UNDER INVEST - VEHICLE
20893	1011	FORGERY/CNTRFT-CRDT CARDS
18597	1012	FORGERY/CNTRFT-CHECKS
21268	1013	FORGERY/CNTRFT - IDENTITY THEFT
1171	1014	FORGERY/CNTRFT-ALL OTHER
17435	1111	BAD CHECKS-MERCHANDISE/ $300+
4891	1112	BAD CHECKS-LABOR/SERVICE $300+
23310	1113	BAD CHECKS-CASH/OTHER $300+
21935	1121	BAD CHECKS-MERCHANDISE/UNDER $300
9435	1122	BAD CHECKS-LABOR/SERVICE UNDER $300
9914	1123	BAD CHECKS-CASH/OTHER UNDER $300
14185	1211	EMBEZZLEMENT $300 OR MORE
14784	1212	EMBEZZLE LARC AFT TRUST ABOVE $300
19266	1213	EMBEZZLE CONFIDENCE GAMES-$300+
6164	1214	EMBEZZLE/THEFT-$300+ OTHER
773	1221	EMBEZZLE UNDER $300
19689	1222	EMBEZZLE LARC AFTER TRUST UNDER $300
3325	1223	EMBEZZLE CONFIDENCE GAMES UNDER $300
12483	1224	EMBEZZLE/THEFT-UNDER $300 OTHER
15288	1311	STOLEN PROP-POSSES/BUY/RECEIVE
11601	1411	VANDALISM-DWELLING
8596	1412	VANDALISM-MOTOR VEHICLE
13128	1413	VANDALISM-COMMERCIAL EST
17030	1414	VANDALISM-SCHOOL
20964	1415	VANDALISM-CHURCH/TEMPLE
13244	1416	VANDAL-CONSTR/SITE/EQUIP
1978	1417	VANDALISM-OTHER
11381	1421	VANDALISM GRAFFITI DWELLING
6848	1422	VANDALISM GRAFFITI MOTOR VEH
22637	1423	VANDALISM GRAFFITI COMMERC EST
39	1424	VANDALISM GRAFFITI SCHOOL
10847	1425	VANDALISM GRAFFITI CHURCH/TEMP
22693	1427	VANDALISM GRAFFITI OTHER
11878	1431	VANDALISM POSSESSION GRAFFITI MATERIAL
9725	1511	WEAPON CONCEALED HANDGUN
9301	1513	WEAPON CONCEALED OTHER WEAPON
953	1521	WEAPON POSSESSION HANDGUN
12128	1522	WEAPON POSSESSION OTHER FIREARM
12166	1523	WEAPON POSSESSION OTHER WEAPON
5845	1531	WEAPON DISCHARGING HANDGUN
4501	1532	WEAPON DISCHARGING OTHER FIREARM
13295	1541	WEAPON TRAFFICIKING HANDGUN
4725	1613	PROSTITUTION/VICE-DISORDERLY HOUS
11559	1614	PROSTITUTION/VICE-SOLICIT/PANDER
12081	1711	SEX OFFENSE - SEX ASSAULT
17823	1712	SEX OFFENSE- INDECENT EXPOSURE
16141	1713	SEX OFFENSE - INDECENT PHONE CALL
17413	1714	SEX OFFENSE - PEEPING TOM
16740	1715	SEX OFFENDER-REGISTRATION VIOLATION
1646	1716	SEX OFFENSE - 4TH DEGREE SEX OFFENSE

1646	1716	SEX OFFENSE - 4TH DEGREE SEX OFFENSE
7939	1718	SEX OFFENSE - ALL OTHER SEX OFFENSE
12683	1812	CDS-MANU-HALLUC/LSD/PCP/ETC
14094	1814	CDS-MANU-MARIJUANA/HASHISH
7390	1815	CDS-MANU-SYNTH DEMEROL/METHADO
176	1817	CDS-MANU INHALANT/GLUE/AEROSOL
5557	1818	CDS-MANU DRUG OVERDOSE NOT FATAL
8794	1821	CDS-SELL-OPIUM & DERIVATIVES
7677	1822	CDS-SELL-HALLUC/LSD/PCP ETC
12991	1823	CDS-SELL-COCAINE& DERIVATIVES
11246	1824	CDS-SELL-MARIJUANA/HASHISH
21247	1825	CDS-SELL-SYNTH DEMEROL/METHADO
18431	1826	CDS-SELL-BARBITURATES/AMPHETAM
6378	1831	CDS-POSS- OPIUM & DERIVATIVES
14780	1832	CDS-POSS HALLUC/LSD/PCP ETC
9769	1833	CDS-POSS COCAINE& DERIVATIVES
18707	1834	CDS-POSS MARIJUANA/HASHISH
9206	1835	CDS-POSS SYNTH DEMEROL/METHADO
17082	1836	CDS-POSS BARBITURATES/AMPHETAM
378	1837	CDS-POSS INHALANT/GLUE/AEROSOL
21526	1838	CDS-POSS DRUG OVERDOSE NOT FATAL
18509	1841	CDS-USE OPIUM & DERIVATIVE
11477	1842	CDS-USE HALLUC/LSD/PCP/ETC.
7276	1843	CDS-USE COCAINE& DERIVATIVES
8583	1844	CDS-USE MARIJUANA/HASHISH
6189	1845	CDS-USE SYNTH DEMEROL/METADONE
19942	1848	CDS-USE DRUG OVERDOSE NOT FATAL
19553	1851	CDS RX FORG OPIUM & DERIVATIVE
16945	1855	CDS RX-FORGERY SYNTH DEMEROL/METHA
3656	1856	CDS RX FORGERY BARBITURATES/AMPHET
22732	1857	CDS RX FORGERY INHALANT/GLUE/AEROS
13024	1861	CDS IMPLEMNT-OPIUM&DERIVATIVE
18944	1862	CDS IMPLMENT-HALLUC/LSD/PCP/ET
5330	1863	CDS IMPLEMNT-COCAINE& DERIVATI
9603	1864	CDS IMPLMNT-MARIJUANA/HASHISH
11108	1865	CDS IMPLMNT-SYNTH DEMEROL/METH
4490	1866	CDS IMPLMNT-BARBITUR/AMPHETAMI
14823	1868	CDS IMPLMNT-OVERDOSE/NOT FATAL
5695	2012	FAMILY OFFENSE-NEGLECT/CHILD
9617	2013	FAMILY OFFENSE - ABUSE/CHILD
11621	2014	FAMILY OFFENSE - CHILD UNDER 12 TAKEN BY PARENT
2834	2015	FAMILY OFFENSE - ELDER ABUSE
7242	2016	FAMILY OFFENSE - ELDER NEGLECT
7968	2111	JUVENILE RUNAWAY
6305	2113	JUVENILE OTHER/RUNAWAY-OTHER JURISDICTION
15228	2114	JUVENILE OFFENSE OTHER
22426	2211	LIQUOR - UNDERAGE DRINKING PARTY
11422	2212	LIQUOR - FURNISHING LIQUOR UNDER 21
13228	2213	LIQUOR - UNLAWFUL POSS UNDER 21
8292	2215	LIQUOR - HOURS SALE LIQUOR VIOLATION
351	2216	LIQUOR - DRINK IN PUB OVER 21
14454	2412	DISCONDUCT-DISORDERLY HOUSE(NOT SEX)
7612	2413	DISORDERLY CONDUCT
4534	2611	SUICIDE - FIREARM
17037	2612	SUICIDE - CUT/STAB
18251	2613	SUICIDE - POISON/OVERDOSE
20706	2614	SUICIDE - HANGING
16470	2615	SUICIDE - ASPHYXIATION
9853	2616	SUICIDE - OTHER
12677	2621	SUICIDE-ATTEMPT-FIREARM
5370	2622	SUICIDE-ATTEMPT-CUT/STAB
18569	2623	SUICIDE-ATTEMPT-POISON/OVERDOSE
17742	2624	SUICIDE-ATTEMPT-HANGING
14057	2626	SUICIDE-ATTEMPT-OTHER
5408	2711	FAIL TO RETURN RENTAL PROPERTY
22646	2712	HOME IMPROVEMENT VIOLATION
3015	2713	IMPERSONATING POLICE OFFICER
12842	2715	BLACKMAIL/EXTORTION
35	2716	BOMB THREAT
18861	2717	EXPLOSIVE DEVICE
14311	2719	FAIL PAY BOARD/LDG/FOOD/TAXI/SERVIC
7093	2722	FALSE STATEMNT/REPORT OF CRIME
7194	2724	FIREWORKS
20687	2725	ESCAPEE
9630	2726	KIDNAPPING
8527	2727	LITTERING/TRASH DUMPING
13480	2728	LOITERING
17767	2731	PORNOGRAPHY
4229	2732	WELFARE FRAUD
21594	2733	RENTAL CAR VIOLATION
1218	2734	ROGUE AND VAGABOND
11564	2735	SOLICITING/TRADE W/O LIC
8009	2736	UNAUTH. USE OF MOTOR VEHICLE
13321	2737	TRESPASSING
2451	2738	THREATENING/ANNOYING PHONE CALL
5074	2739	PANHANDLING
4360	2741	HARASSMENT/STALKING
22473	2742	EX PARTE/PROTECT. ORDER VIOL.
10205	2751	FUGITIVE FROM OTHER MD JURISDICTION
5659	2752	FUGITIVE FROM JUSTICE(OUT OF STATE)
14692	2791	ALL OTHER NON-TRAFFIC CRIM OFFENSES
10750	2811	ABANDONED AUTO
10455	2812	DRIVING UNDER THE INFLUENCE

```
16455   2812                      DRIVING UNDER THE INFLUENCE
9733    2813                                   TRAFFIC HAZARD
2454    2814                                 PARKING OFFENSES
3919    2815                 VIOLATION OF ALCOHOL RESTRICTION
18908   2911                      SUDDEN DEATH ACC NON TRAFF
8061    2912                           SUDDEN DEATH DROWNING
16438   2913                            SUDDEN DEATH NATURAL
13524   2914                       SUDDEN DEATH UNDETERMINED
5946    2934                                           DRUNK
17936   2935                                      FIRE OTHER
21368   2936                                      ILL PERSON
19482   2937                             INJURY- NONTRAFFIC
11611   2938                                 POL INFORMATION
17458   2939                          HOMELAND SECURITY EVENT
13984   2941                                   LOST PROPERTY
2162    2942                                MENTAL TRANSPORT
17713   2943                                  MISSING PERSON
18627   2945                   RECOVERED PROPERTY - FIREARM
2763    2946                   RECOVERED PROPERTY/MONT. CO.
15119   2947                        RECOVERED PROPERTY/OTHER
20489   2949                   SANE COLLECTION NON-STRANGER
20171   2951                                  FAMILY TROUBLE
23343   2952                  SUSPICIOUS SIT/PERSON/VEHICLE
11965   2991                            OTHER MISCELLANEOUS
796     3213                        ANIMAL OFFENSE - HOT CAR
22375   3220                 ANIMAL OFFENSE - AGG ANIMAL/BITE
5831    3240                          ANIMAL NUISANCE/BARKING
```

1. Crimes identified by Class between 111 and 933 could be classified as violent.
2. Crimes from 911-933 fall into arson category. Arson as a single crime would not classify as violent but I tend to believe that here there is a combination of crimes which could be classified as violent.
3. For the time being I will classify crimes between 111-933 as violent

Setup violent crime classification

In [54]:

```
crimes.loc[((crimes['Class']>=111) &(crimes['Class']<=933)),'Violent Crimes']='Violent'
crimes.loc[(crimes['Class']>933),'Violent Crimes']='Non-Violent'

#print first 5 rows
crimes.loc[:,['Class','Class Description','Violent Crimes']].head()
```

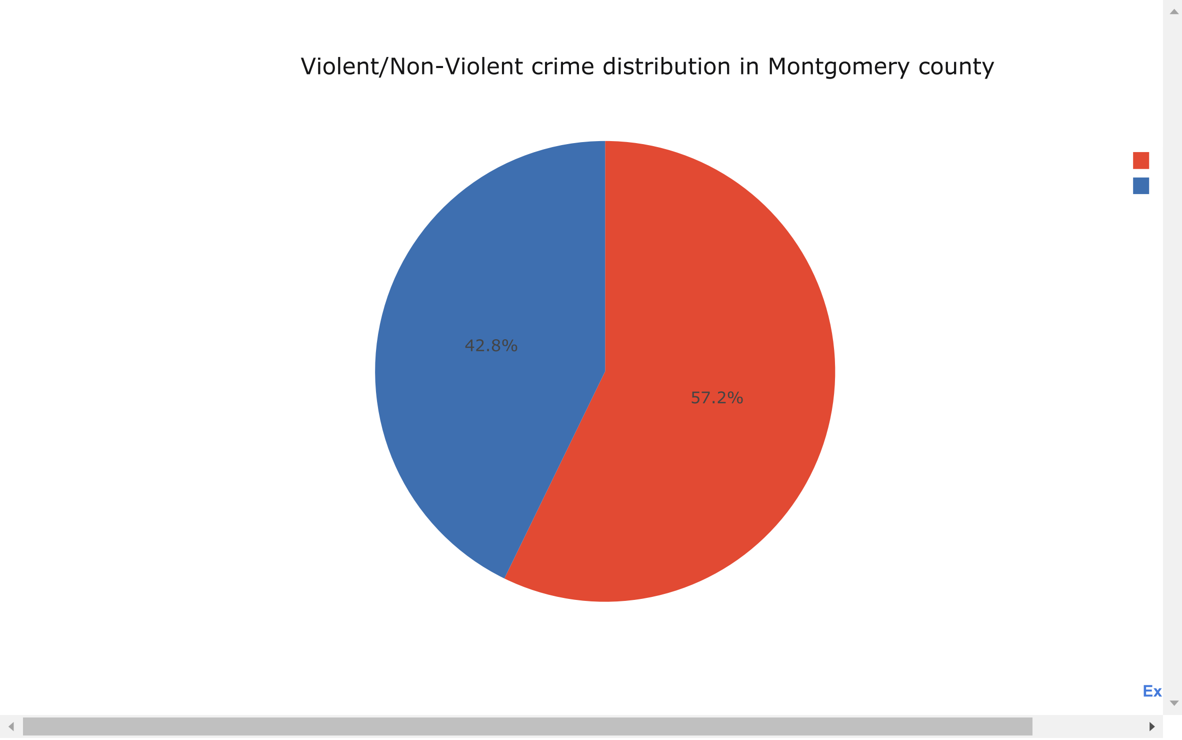
Out[54]:

	Class	Class Description	Violent Crimes
10	2938	POL INFORMATION	Non-Violent
13	821	SIMPLE ASSAULT - CITIZEN	Violent
14	634	LARCENY FROM AUTO UNDER $50	Violent
15	2623	SUICIDE-ATTEMPT-POISON/OVERDOSE	Non-Violent
16	1712	SEX OFFENSE- INDECENT EXPOSURE	Non-Violent

Violent/Non-Violent crime distribution

In [55]:

```
# Getting violent/non violent crime counts
violent_count=crimes['Violent Crimes'].value_counts()

# used the same process as for the time analysis pie chart
violent_count=violent_count.to_frame()
violent_count.reset_index(inplace=True)
violent_count.rename(columns={'index':'labels','Violent Crimes':'values'}, inplace=True)

# plot
violent_count.iplot(kind='pie', labels='labels',values='values',title='Violent/Non-Violent crime distribution in Montgomery county')
violent_count
```



Out[55]:

	labels	values
0	Non-Violent	13279
1	Violent	9929

Violent/Non-Violent breakdown:

- 42.8% violent crimes; 57.2% non-violent
- This is somewhat a little bit of a surprise as violent/non-violent values are quite close. I would have expected much lower values for non-violent crimes.

Pie Plotting function:

- pie_plot:
  - Ref: https://plot.ly/python/pie-charts/ (https://plot.ly/python/pie-charts/)
  - this type of plot needs a figure and a layout
  - both figure and layout are dictionaries
  - figure is comprised of:
    - domains grid for a particular axis x,y values are between [0,1]
    - labels list
    - values list
  - particular for this code I used zip to iterate over the items mentioned above at the same time & append the items to data

- make_annotations
  - annotations are part of the layout dictionary
  - annotations are as well dictionaries
  - I decided I didn't want an entire function for the layout but will need one for annotations
  - make_annotations requites:
    - size - font size fo the text
    - text - text associated with the pie chart
    - x - positioning of the pie charts on x axis
    - y - positioning on y axis

In [56]:

```
def pie_plot(domains,labels,values):
    data=[]
    original = [domains,labels,values]
    zipped=tuple([list(tup) for tup in zip(*original)])
    for i in range(0,len(zipped)):
        data.append({
            'labels': zipped[i][1],
            'values': zipped[i][2],
            'type': 'pie',
            'name': 'Starry Night',
            'domain': zipped[i][0],
            'hoverinfo':'label+percent+name',
            'hole': .4,
            'pull': .2
            })
    return data

def make_annotations(size,text,x,y):
    if len(text)==len(x)==len(y):
        data=[]
        size=[size]*len(x)
        for i,j,a,b in zip(size,text,x,y):
            data.append({
                "font":{"size":i},
                "showarrow": False,
                "text":j,
                "x":a,
                "y":b
            })
        return data
    else:
        print('Length mismatch')
```

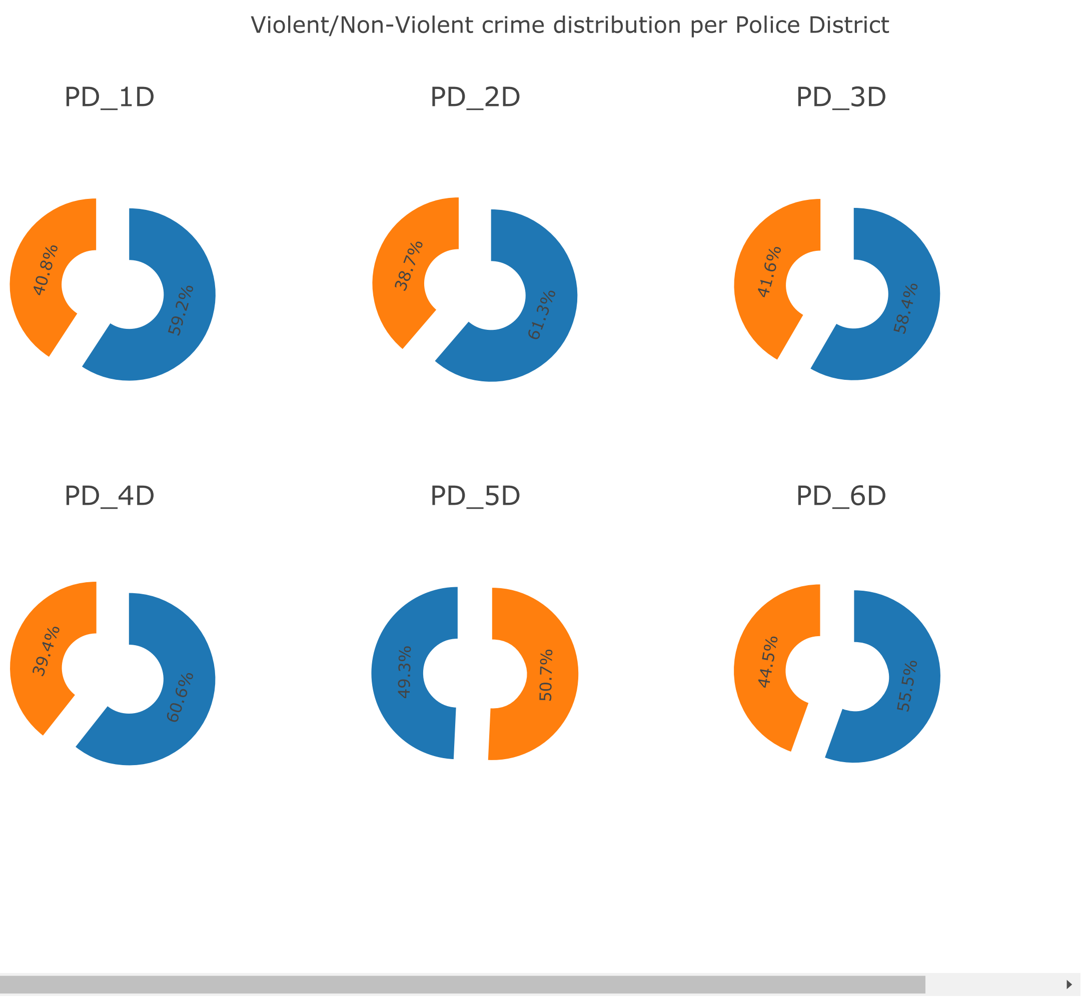
Violent/Non-Violent crime distribution per PD

In [57]:

```
# Slice the dataframe based on Police District Number & Violent Crimes
vc_pd=crimes.loc[:,['Police District Number','Violent Crimes']]

labels=[]
values=[]

# loopt through each PD
for pd in ['1D','2D','3D','4D','5D','6D']:

    # get the violent/non-violent crime counts for each PD
    item=vc_pd[vc_pd['Police District Number']==pd]['Violent Crimes'].value_counts()

    # create 2 list from the above step; labels & values
    l=item.index.tolist()
    v=item.values.tolist()
    labels.append(l)
    values.append(v)
```

In [58]:

```
domains=[({'x': [0, .20],'y': [0, .49]}),
         ({'x': [.35, .55],'y': [0, .49]}),
         ({'x': [.70, .90],'y': [0, .49]}),
         ({'x': [0, .20],'y': [.50, 3.93]}),
         ({'x': [.35, .55],'y': [.50, 3.93]}),
         ({'x': [.70, .90],'y': [.50, 3.93]})]

data=[]
data=pie_plot(domains,labels,values)

size=20
text=['PD_1D','PD_2D','PD_3D','PD_4D','PD_5D','PD_6D']
x=[0.05,0.45,0.85,0.05,0.45,0.85]
y=[1.03,1.03,1.03,0.48,0.48,0.48]
anno=make_annotations(size,text,x,y)

layout = dict(height = 750,
              width = 1000,
              autosize = False,
              title = 'Violent/Non-Violent crime distribution per Police District',
              annotations= anno
              )

fig = dict(data=data, layout=layout)
iplot(fig, filename='pie-subplots')
```



PD violent/non-violent breakdown:

1D | Rockville:

1. violent 40.8%
2. non-violent 59.2%

2D | Bethesda:

1. violent 37.8%
2. non-violent 61.3%

3D | Silver Spring:

1. violent 41.6 %
2. non-violent 58.4%

4D | Wheaton:

1. violent 39.4%
2. non-violent 60.6%

5D | Germantown:

1. violent 50.7%
2. non-violent 49.3 %

6D | Gaitersburg:

1. violent 44.5%
2. non-violent 55.5%

1. Violent crimes range from 38.7-44.5%
2. Non-violent crimes range from 55.4-61.3%
3. The only surprise is for PD_5D violent crimes outnumber the non-violent crimes;(violent/non-violent) distribution is 50.7%/49.3%
4. highest violent percentage PD_5D 50.7%
5. lowest violent percentage PD_2D 38.7%
6. highet non-violent percentage PD_2D 61.3%
7. lowest non-violent percentage PD_5D 49.3%

Analyze violent/non-violent crime distribution per PD & Place

1. Get top 5 places where violent/non-violent crimes are committed
2. For each PD create a slice of the dataframe where based on PD number and Crime categorization
3. sort values in descending order

In [59]:

```
import pandas as pd

# Create a slice of the df based on 'Police District Number','Place','Violent Crimes'
crimes_pd_place=crimes.loc[:,['Police District Number','Place','Violent Crimes']]

# get violent/non-violent crime counts per PD & Place
# 1) Create a new df by grouping the slice based on 'Police District Number','Place'
# 2) Chain the df to 'Violent Crimes',get value_counts(),reset the index

pd_place_vc=pd.DataFrame({'Counts' : crimes_pd_place.groupby(['Police District Number','Place'])['Violent Crimes'].value_counts()}).r
labels_v=[]
values_v=[]

labels_nv=[]
values_nv=[]

# loop throught the PDs
for pd in ['1D','2D','3D','4D','5D','6D']:
    # get the violent/non-violent crime counts for each PD
    item_v=pd_place_vc[(pd_place_vc['Police District Number']==pd)& (pd_place_vc['Violent Crimes']=='Violent')].sort_values(by='Count
    item_nv=pd_place_vc[(pd_place_vc['Police District Number']==pd)& (pd_place_vc['Violent Crimes']=='Non-Violent')].sort_values(by='

    # create labels,values lists for each case i.e violent/non-violent
    labels_v.append(item_v['Place'])
    values_v.append(item_v['Counts'])

    labels_nv.append(item_nv['Place'])
    values_nv.append(item_nv['Counts'])
```

Plotting most common places where violent crimes are commited /PD

Grupped bar subplots plotting functions:

1. plot_subplots:

   A. Ref: https://plot.ly/python/subplots/ (https://plot.ly/python/subplots/)
   B. this require a figure & layout
   C. the figure is comprised of individual traces and (row; column) specification
   D. each trace is made out a x<-->label and y<-->values
   E. particular to this function I will use zip to iterate over labels,value,names at the same time & construct the traces
   F. make the figure layout
   G. append each trace to the figure

1. make_titles:
   A. this will create a list with titles
   B. titles are made out a base string + another string typically this would be the PDs

In [60]:

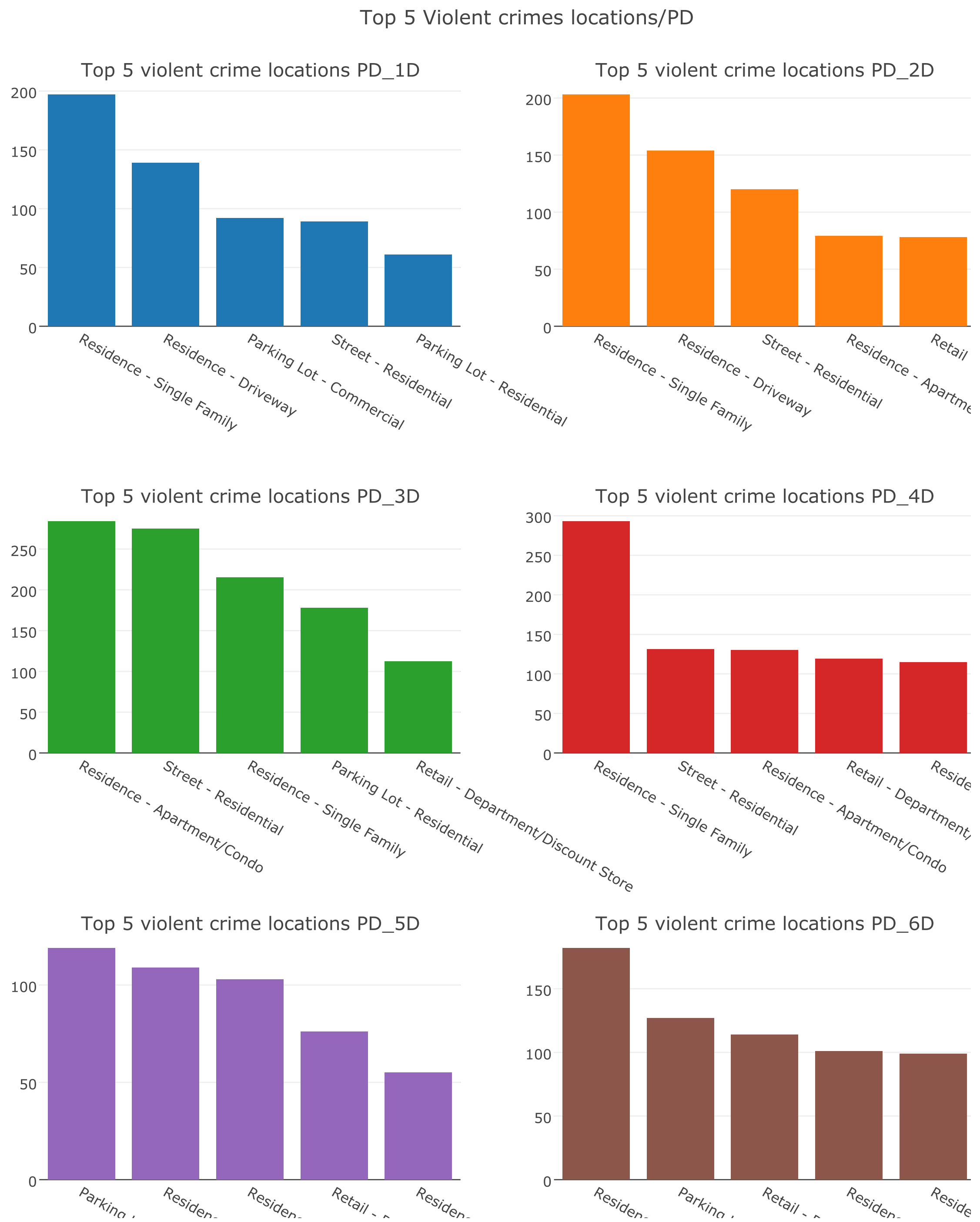
```
def plot_subplots(labels,values,row,col,names,titles):
    if len(labels)==len(values)==len(names)==len(titles):

        original = [labels,values,names]
        zipped=tuple([list(tup) for tup in zip(*original)])

        trace=[]
        for i in range(0,len(labels)):
            trace+=['trace'+str(i+1)]
            trace[i]=go.Bar(x=zipped[i][0],y=zipped[i][1],name=zipped[i][2])

        fig = tools.make_subplots(
            rows=row, cols=col, subplot_titles=(titles)
            )

        layout_row=[]
        layout_col=[]

        for i in range(1,row+1):
            for j in (1,col):
                layout_row.append(j)
                layout_col.append(i)

        for i,j,k in zip(trace,layout_col,layout_row):
            fig.append_trace(i,j,k)

        return fig
    else:
        return print('Length mismatch')
        exit()


def make_titles(names,base_name):
    titles=[]
    for n in names:
        titles+=[str(base_name)+' '+str(n)]
    return titles
```

Plot crime locations distribution/PD

In [61]:

```
# setup labels & values
labels=labels_v
values=values_v

# strings used to construct titles
names=['PD_1D','PD_2D','PD_3D','PD_4D','PD_5D','PD_6D']
base_name='Top 5 violent crime locations'

#construct titles list
titles=make_titles(names,base_name)

row=3;
col=2

#create figure layout
fig=plot_subplots(labels,values,row,col,names,titles)

#plot
fig['layout'].update(height=1200,width=1000, margin=go.Margin(b=155), title='Top 5 Violent crimes locations/PD')
iplot(fig, filename='stacked-subplots-shared-xaxes')
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
```



Top 5 violent crime locations:

1. The actual places where these crimes are committed are more specific i.e Residence Single Family but I will not focus on that as I fell it will not bring an substantial added value to the analysis
2. The order of this locations differ from each PD, however I would not focus on that either
3. Most common place where crimes are committed are Residence;Street;Parking Lot;Department/Retail store

In [62]:

```
#The same process as above will follow

labels=labels_nv
values=values_nv

base_name='Top 5 non-violent crime locations'
titles=make_titles(names,base_name)

row=3;
col=2

fig=plot_subplots(labels,values,row,col,names,titles)

fig['layout'].update(height=1200,width=1000, margin=go.Margin(b=130), title='Top 5 Non-Violent crimes locations/PD')
iplot(fig, filename='stacked-subplots-shared-xaxes')
```

```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
```

Top 5 non-violent crime locations:

1. Most common non-violent crime locations are Residence;Street;Parking Lot;Department/Retail Store
2. These are the same locations as for violent crimes

Analize violent crimes by main categories

1. The goal here is to know what is the percentage of a particular violent crime sub category.
2. Classification structure:
   A. create a function that is applied against the 'Class' columns i.e(Homicide,Rape...etc)
   B. depending on the values the function will return the overall category
   C. return the results in a new column

In [63]:

```
def class2nd(x):
    if ((x>100) & (x<=199)):
        #print(x)
        return 'HOMICIDE'
    elif ((x>200) & (x<=299)):
        return 'RAPE'
    elif ((x>300) & (x<=399)):
        return 'ROBBERY'
    elif ((x>400)& (x<=499)):
        return 'AGG ASSAULT'
    elif ((x>500)&(x<=599)):
        return 'BURGLARY'
    elif ((x>600)&(x<=699)):
        return 'LARCENY'
    elif ((x>700)&(x<=799)):
        return 'AUTO THEFT'
    elif ((x>800)&(x<=815)):
        return 'ASSAULT & BATTERY'
    elif ((x>815)&(x<=825)):
        return 'ASSAULT'
    elif ((x>900)&(x<999)):
        return 'ARSON'

crimes['Class Main Cathegory']=crimes['Class'].apply(class2nd)
```

Get violent crimes count by main class category

In [64]:

```
import pandas as pd

# 1) create a new df end excluce non-violent crimes
# 2) group by PD and get the counts for each violent crime main cathegory

violent_main=pd.DataFrame({'Counts': crimes[crimes['Class Main Cathegory'].notnull()==True].groupby(['Police District Number'])['Clas

#print preview
violent_main
```

Out[64]:

	Police District Number	Class Main Cathegory	Counts
0	1D	LARCENY	845
1	1D	BURGLARY	220
2	1D	ASSAULT	94
3	1D	ASSAULT & BATTERY	87
4	1D	AUTO THEFT	56
5	1D	ROBBERY	40
6	1D	AGG ASSAULT	12
7	1D	RAPE	5
8	1D	ARSON	3
9	2D	LARCENY	1307
10	2D	BURGLARY	188
11	2D	ASSAULT & BATTERY	89
12	2D	ASSAULT	55
13	2D	AUTO THEFT	32
14	2D	ROBBERY	30
15	2D	AGG ASSAULT	5
16	2D	ARSON	2
17	2D	RAPE	2
18	3D	LARCENY	1475
19	3D	BURGLARY	343
20	3D	ASSAULT & BATTERY	187
21	3D	ROBBERY	131
22	3D	AUTO THEFT	129
23	3D	ASSAULT	109
24	3D	AGG ASSAULT	52
25	3D	RAPE	16
26	3D	ARSON	6
27	3D	HOMICIDE	1
28	4D	LARCENY	1057
29	4D	BURGLARY	209
...	...	...	...
31	4D	ASSAULT	120
32	4D	AUTO THEFT	69
33	4D	ROBBERY	62
34	4D	AGG ASSAULT	48
35	4D	ARSON	9
36	4D	RAPE	4
37	4D	HOMICIDE	1
38	5D	LARCENY	634
39	5D	BURGLARY	128
40	5D	ASSAULT & BATTERY	106
41	5D	ASSAULT	75
42	5D	AUTO THEFT	54
43	5D	ROBBERY	29
44	5D	AGG ASSAULT	17
45	5D	ARSON	6
46	5D	RAPE	6
47	5D	HOMICIDE	1
48	6D	LARCENY	945

	Police District Number	Class Main Cathegory	Counts
49	6D	BURGLARY	207
50	6D	ASSAULT & BATTERY	148
51	6D	ASSAULT	103
52	6D	AUTO THEFT	65
53	6D	ROBBERY	56
54	6D	AGG ASSAULT	33
55	6D	RAPE	8
56	6D	ARSON	1
57	6D	HOMICIDE	1
58	TPPD	LARCENY	3
59	TPPD	ARSON	1
60	TPPD	ASSAULT	1

61 rows × 3 columns

In [65]:

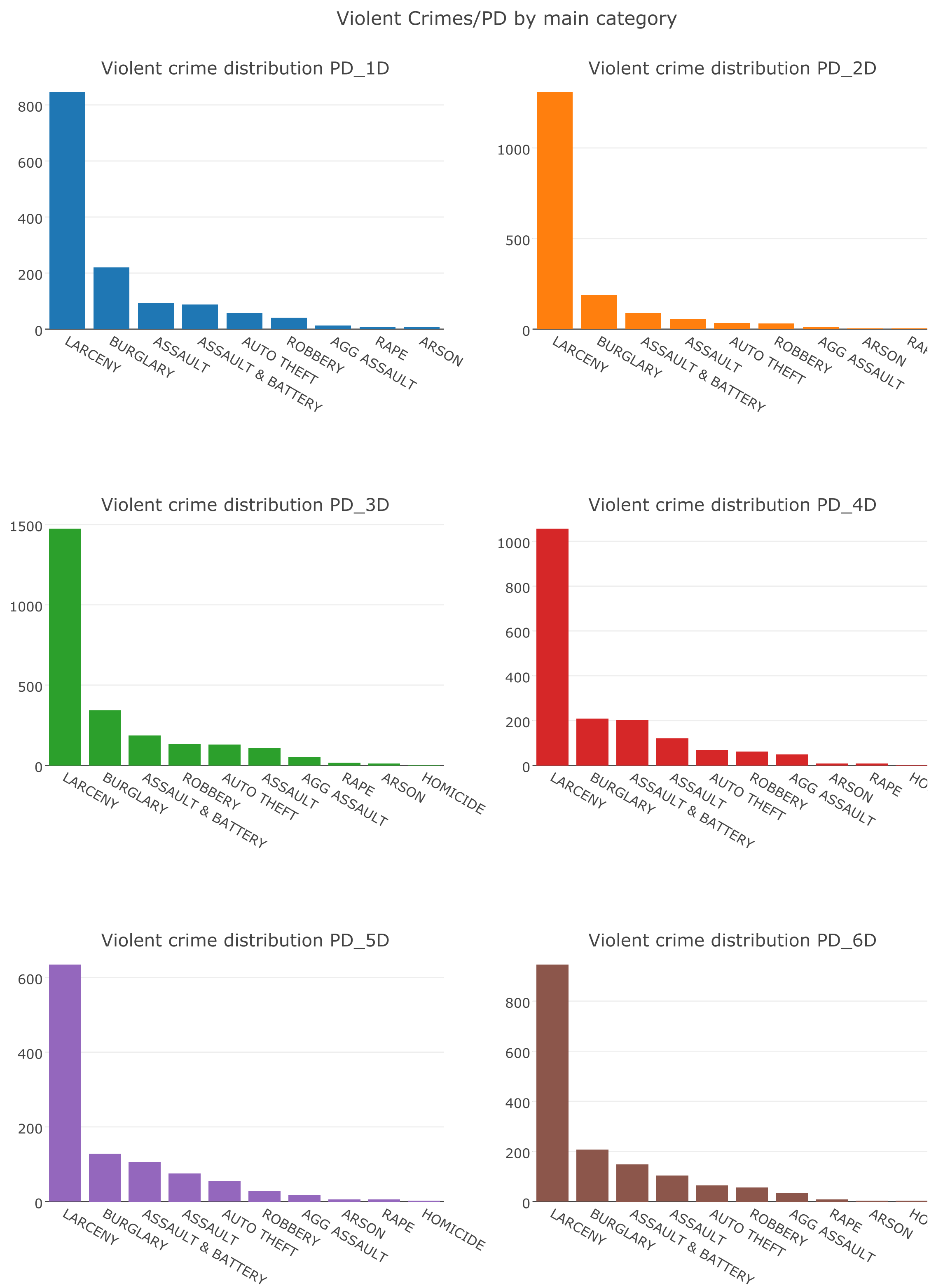
```
labels_v=[]
values_v=[]

# loop through each PD
for pd in ['1D','2D','3D','4D','5D','6D']:
    # for each PD get the main cathegory & counts
    l=violent_main[violent_main['Police District Number']==pd]['Class Main Cathegory']
    v=violent_main[violent_main['Police District Number']==pd]['Counts']

    # create labels & values list
    labels_v.append(l)
    values_v.append(v)
```

Plot main violent crimes categories distribution on PDs

In [75]:

```
labels=labels_v
values=values_v

base_name='Violent crime distribution'
titles=make_titles(names,base_name)

row=3
col=2
# plot
fig=plot_subplots(labels,values,row,col,names,titles)

fig['layout'].update(height=1200,width=1000, title='Violent Crimes/PD by main category')
iplot(fig, filename='stacked-subplots-shared-xaxes')
```

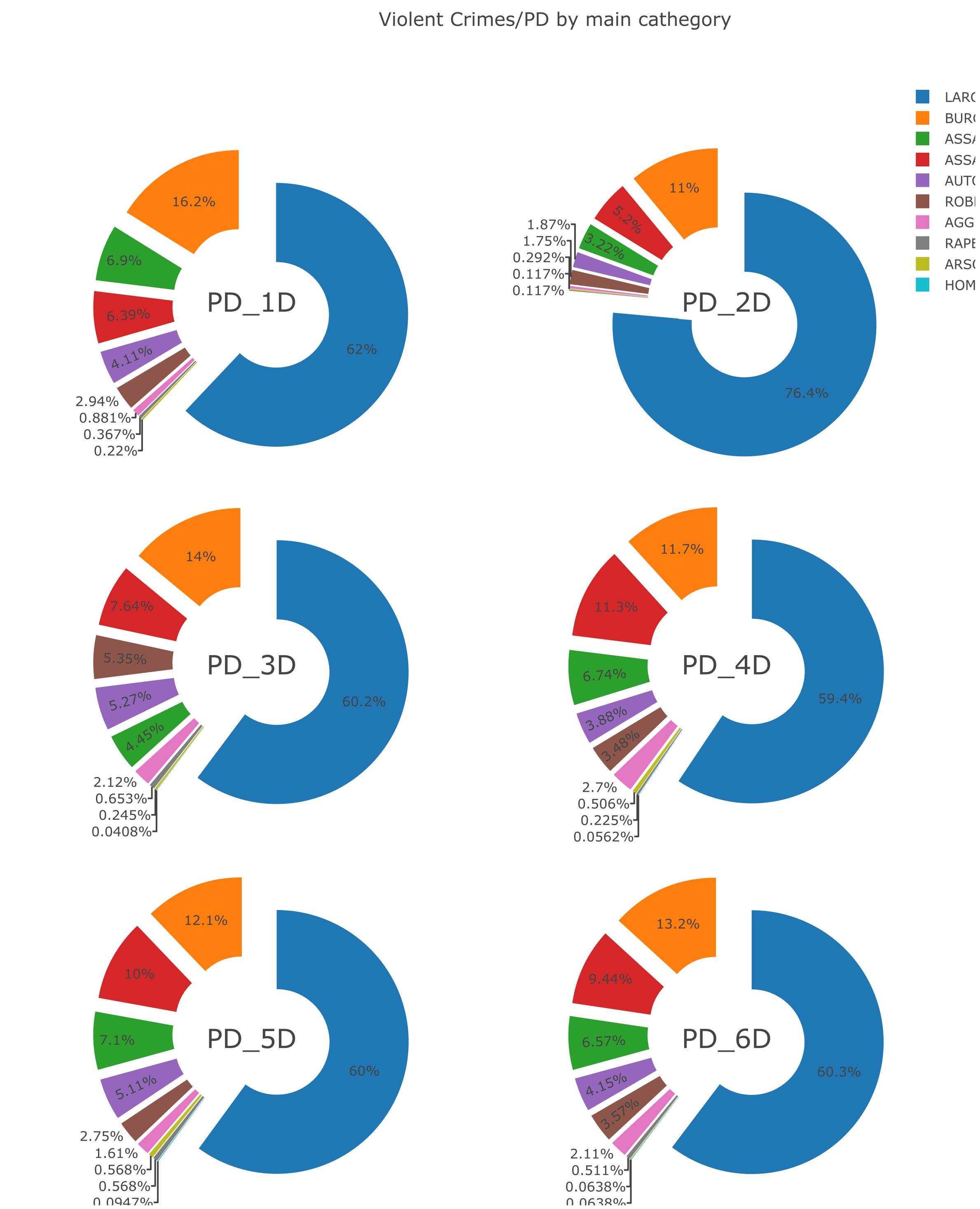
```
This is the format of your plot grid:
[ (1,1) x1,y1 ]  [ (1,2) x2,y2 ]
[ (2,1) x3,y3 ]  [ (2,2) x4,y4 ]
[ (3,1) x5,y5 ]  [ (3,2) x6,y6 ]
```



Main violent crimes categories distribution on PDs - Pie Chart

In [67]:

```
domains=[({'x':[0, 0.40], 'y':[0.60,1.0]}), # for the cell (1,1)
         ({'x':[0.60, 1], 'y':[0.60,1]}),#cell (1,2)
         ({'x':[0, 0.40], 'y':[0.30, 0.66]}), #cell (2,1)
         ({'x':[0.60, 1], 'y':[0.30, 0.66]}),#cell (2,2)
         ({'x':[0, 0.40], 'y':[0,0.30]}),#cell (3,1)
         ({'x': [0.60, 1], 'y':[0,0.30]})]#cell (3,1)

data=pie_plot(domains,labels,values)

size=24
x=[0.14,0.86,0.14,0.86,0.14,0.86]
y=[0.82,0.82,0.48,0.48,0.13,0.13]

anno=make_annotations(size,text,x,y)
layout = dict(height = 1200,
              width = 1000,
              autosize = False,
              title = 'Violent Crimes/PD by main category',
              annotations=anno
             )

fig = dict(data=data, layout=layout)
iplot(fig, filename='pie-subplots')
```



Violent crimes sub categories breakdown:

For the most part, the order of the violent crimes in terms of percentage from high to low is the following(did not focus on the exact order):

```
1. Larceny
2. Burglary
3. ASSAULT & BATTERY
4. ASSAULT
5. AUTO THEFT
6. ROBBERY
7. AGGRAVATED ASSAULT
8. RAPE; ARSON; HOMICIDE
```

1. Larceny:
   A. ranges from 59,4%-76,4%
   B. highest percent PD_2D
   C. lowest percent PD_4D
2. Burglary:
   A. ranges from 11%-16,2%
   B. highest percent PD_1D
   C. lowest percent PD_2D
3. ASSAULT & BATTERY:
   A. ranges from 5.2%-11,4%
   B. highest percent PD_4D
   C. lowest percent PD_2D
4. ASSAULT:
   A. ranges from 3.22%-7,1%
   B. highest percent PD_5D
   C. lowest percent PD_2D
5. AUTO THEFT:
   A. ranges from 1.87%-5,27%
   B. highest percent PD_3D
   C. lowest percent PD_2D
6. ROBBERY:
   A. ranges from 1.75%-5,35%
   B. highest percent PD_3D
   C. lowest percent PD_2D
7. AGGRAVATE ASSAULT:
   A. ranges from 0.292%-2,12%
   B. highest percent PD_3D
   C. lowest percent PD_2D

Time analysis based on violent/non-violent classification

Will attempt to see if there is any difference in dispatch time based on violent/non-violent crime classification.
Will also break down the dispatch time based on violent crime sub categories.
The assumption is that the more violent the crime is the faster is the dispatch time.

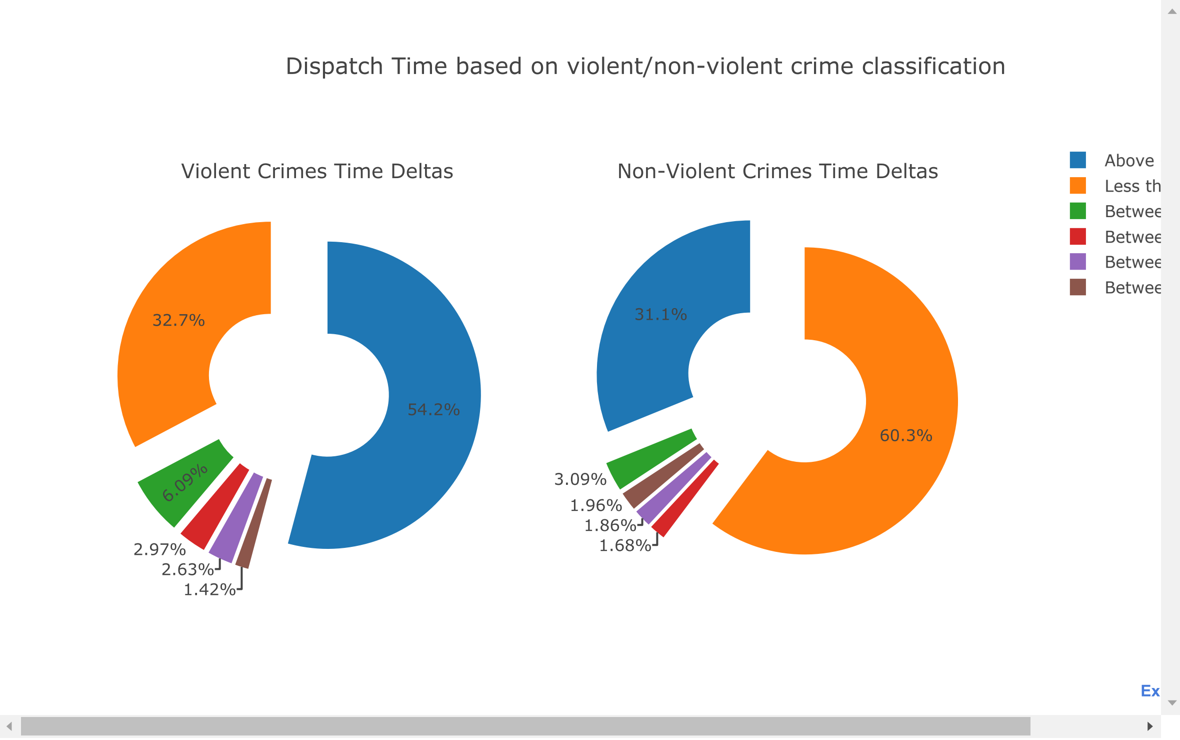
In [68]:

```
import pandas as pd

#create a new df by filtering out the dates before 01.07.2013 & looking at 'Violent Crimes' and 'Date diff' columns
new_td=crimes[crimes['Start Date / Time']>=dt.datetime(2013,7,1)].loc[:,['Violent Crimes','Date diff']].copy()

# function defining dispatch time classification
def time_class(x):
    # assesing time intervals
    if x < pd.Timedelta('1 minute'):
        return 'Less then 1 minute'
    elif ((x>=(pd.Timedelta('1 minute'))) & (x < pd.Timedelta('5 minute'))):
        return 'Between 1 and 5'
    elif ((x>=(pd.Timedelta('5 minute'))) & (x < pd.Timedelta('10 minute'))):
        return 'Between 5 and 10'
    elif ((x>=(pd.Timedelta('10 minute'))) & (x < pd.Timedelta('30 minute'))):
        return 'Between 10 and 30'
    elif ((x>=(pd.Timedelta('30 minute'))) & (x < pd.Timedelta('60 minute'))):
        return 'Between 30 and 60'
    elif x>pd.Timedelta('60 minute'):
        return 'Above 60'

def time_analysis_v_nv(df):
    labels=[]
    values=[]
    # loop through violent/non-violent classification
    for classif in ['Violent','Non-Violent']:

        #create a new df & apply time classification funtcion then get the counts
        td=new_td[new_td['Violent Crimes']==classif].copy()
        td['Time Deltas']=td['Date diff'].apply(time_class)
        td=td['Time Deltas'].value_counts()

        # convert the series to two label & values lists
        l=td.index.tolist()
        v=td.values.tolist()

        labels.append(l)
        values.append(v)

    return (labels,values)
```

Plotting dispatch time interval based on violent/non-violent classification

In [69]:

```
# get data list for plotting a pie chart
domains=[({"x": [0, .40]}),({"x": [.52, .92]})]
[labels,values]=time_analysis_v_nv(df=new_td)
data=pie_plot(domains,labels,values)

# creating the layout
layout = dict(autosize = True,
              margin=go.Margin(b=50),
              title="Dispatch Time based on violent/non-violent crime classification",
              annotations= [{"font": {"size": 15},"showarrow": False,"text": "Violent Crimes Time Deltas","x": 0.07, "y": .97},
                            {"font": {"size": 15},"showarrow": False,"text": "Non-Violent Crimes Time Deltas","x": 0.90, "y": .97}]
              )
# plot
figxx = dict(data=data, layout=layout)
iplot(figxx, filename='pie-subplots')
```



Violent/Non-Violent crime dispatch time breakdown:

- less then 1 minute ---> 82.7% decrease from non-violent to violent
- between 1 minute and 5 minutes --->27.55 % decrease from non-violent to violent
- between 5 minutes and 10 minutes --->41.39% increase from non-violent to violent
- between 10 minutes and 30 minutes --->97.08% increase from non-violent to violent
- between 30 minutes and 60 minutes --->76.78% increase from non-violent to violent
- above 60 minutes --->74.27% increase from non-violent to violent

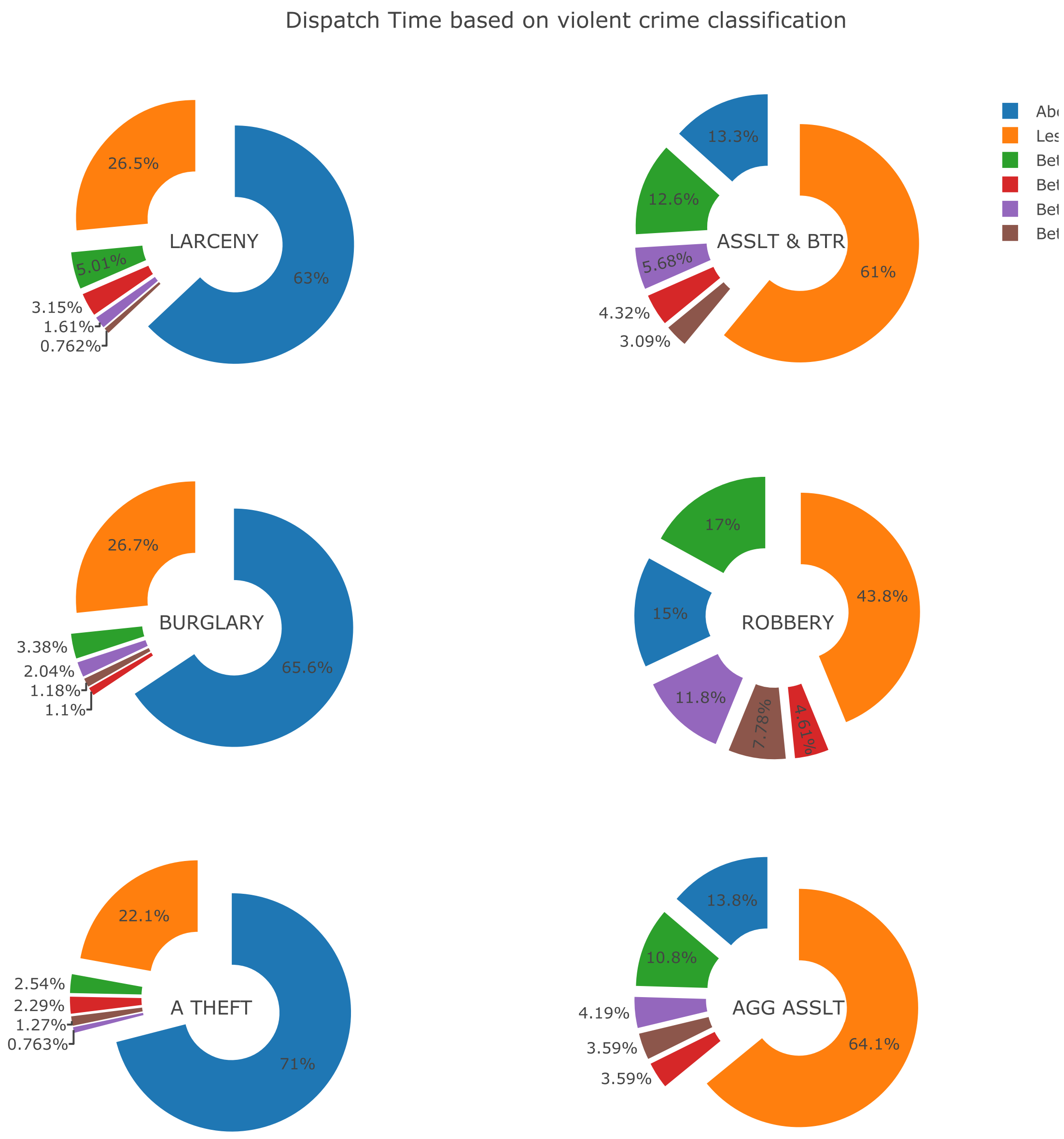
Time delta analysis based on violent class

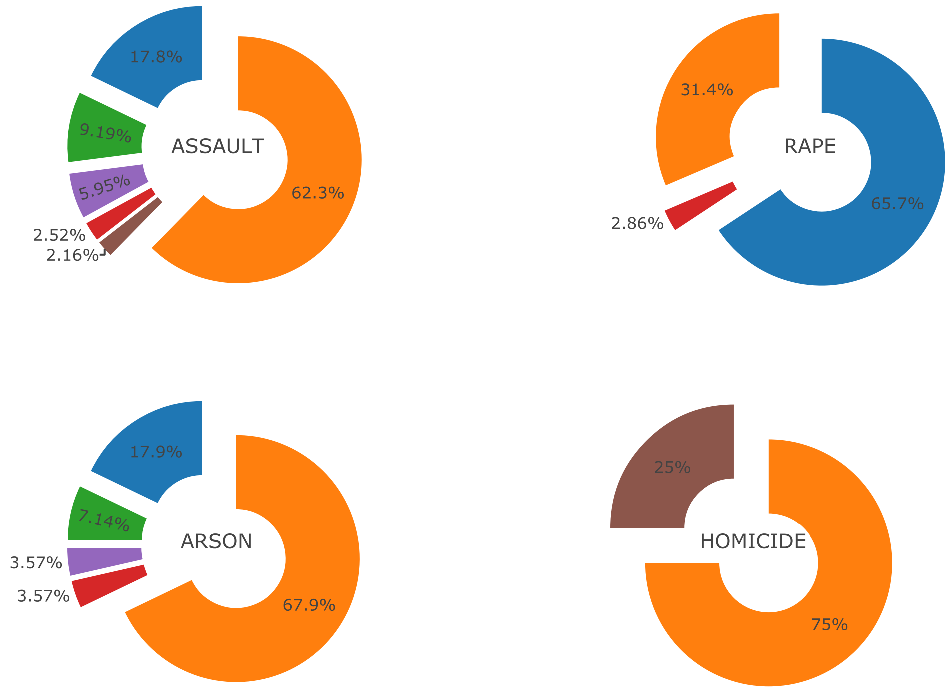
In [70]:

```
# create a data frame for violent crimes
td_v=crimes[((crimes['Violent Crimes']=='Violent')& (crimes['Start Date / Time']>=dt.datetime(2013,7,1)))].loc[:,["Date diff",'Class

data_l=[]
data_v=[]
item_d=[]

#loop through violent crimes categories
for crime in td_v['Class Main Cathegory'].unique():
    #construct 2 lists for plotting labels & values
    data_l+=[str(crime)+'_l']
    data_v+=[str(crime)+'_v']

    # for each violent crime type get dispatch time deltas
    item=td_v[td_v['Class Main Cathegory']==crime]['Date diff']
    item=item.apply(time_class).value_counts()

    # append lables & values to their corresponding lists
    data_l[len(data_l)-1]=item.index
    data_v[len(data_v)-1]=item.values
```

In [71]:

```
domains=[({'x':[0, 0.40], 'y':[0.85, 1]}),    # for the cell (1,1)
        ({'x':[0.60, 1], 'y':[0.85, 1]}),    # cell (1,2)
        ({'x':[0, 0.40], 'y':[0.65, 0.80]}), # cell (2,1)
        ({'x':[0.60, 1], 'y':[0.65, 0.80]}), # cell (2,2)
        ({'x':[0, 0.40], 'y':[0.45, 0.60]}), # cell (3,1)
        ({'x':[0.60, 1], 'y':[0.45, 0.60]}), # cell (3,2)
        ({'x':[0, 0.40], 'y':[0.25,  0.40]}),# cell (4,1)
        ({'x':[0.60 ,1], 'y':[0.25,  0.40]}),# cell (4,1)
        ({'x':[0, 0.40], 'y':[0.05,  0.20]}),# cell (5,1)
        ({'x':[0.5, 1], 'y':[0.05,  0.20]})] # cell (5,1)

labels=data_l
values=data_v

data=pie_plot(domains,labels,values)
text=td_v['Class Main Cathegory'].unique()
text[1]='ASSLT & BTR'
text[4]='A THEFT'
text[5]='AGG ASSLT'
size=15
x=[0.15,0.875,0.14,0.865,0.15,0.875,0.15,0.84,0.16,0.81]
y=[0.93,0.93,0.73,0.73,0.52,0.52,0.32,0.32,0.12,0.12]

anno=make_annotations(size,text,x,y)
layout = dict(height = 1600,
              width = 1000,
              autosize = True,
              margin=go.Margin(b=20),
              title="Dispatch Time based on violent crime classification",
              annotations=anno
             )
figxx = dict(data=data, layout=layout)
iplot(figxx, filename='pie-subplots')
```





Violent crimes dispatch time breakdown:

1. LARCENY:
    A. biggest dispatch time category: above 60 minutes 63%
    B. smallest dispatch time category: between 1 and 5 minutes 0.762%

1. ASSAULT & BATTERY:
    A. biggest dispatch time category: less then 1 minute 61%
    B. smallest dispatch time category: between 1 and 5 minutes 3.09%

1. BURGLARY:
    A. biggest dispatch time category: above 60 minutes 65.6%
    B. smallest dispatch time category: between 30 and 60 minutes 1.1%

1. ROBBERY:
    A. biggest dispatch time category: less then 1 minute 43.8%
    B. smallest dispatch time category: between 30 and 60 minutes 4.61%

1. AUTO THEFT:
    A. biggest dispatch time category: above 60 minutes 71%
    B. smallest dispatch time category: between 5 and 10 minutes 0.763%

1. AGGRAVATE ASSAULT:
    A. biggest dispatch time category: less then 1 minute 64.1%
    B. smallest dispatch time category: between 1 and 5 minutes 3.59%

1. ASSAULT:
    A. biggest dispatch time category: less then 1 minute 62.3%
    B. smallest dispatch time category: between 1 and 5 minutes 2.16%

1. RAPE:
    A. biggest dispatch time category: above 60 minutes 65.7%
    B. smallest dispatch time category: between 30 and 60 minutes 2.86%

1. ARSON:
    A. biggest dispatch time category: less then 1 minutes 67.9%
    B. smallest dispatch time category: between 30 and 60 minutes 3.57%

1. HOMICIDE:
    A. biggest dispatch time category: less then 1 minutes 75.%
    B. smallest dispatch time category: between 1 and 5 minutes 25%

1. DISPAtCH TIME DELTAS CATEGORY BREAKDOWN:
    A. less than 1 minutes:
        a. highest % HOMICIDE 75%
        b. lowest % AUTO THEFT 22.1%

B. between 1 minute and 5 minutes:
   a. highest % HOMICIDE 25%
   b. lowest % LARCENY 0.762%
C. between 5 minutes and 10 minutes:
   a. highest % ROBBERY 11.8%
   b. lowest % AUTO THEFT 0.763
D. between 10 minutes and 30 minutes:
   a. highest % ROBBERY 17%
   b. lowest % AUTO THEFT 2.54%
E. between 30 minutes and 60 minutes:
   a. highest % ROBBERY 4.61%
   b. lowest % BURGLARY 1.1%
F. above 60 minutes:
   a. highest % AUTO THEFT 71%
   b. lowest % ASSAULT & BATTERY 13.3%

Map visualization

Having a map representation of the crime distribution can make the data more compelling to a casual reader.
With this in mind i used folium to plot crime on Montgomery district.
Columns used:

1. `Latitude`
2. `Longitude`

folium plotting function

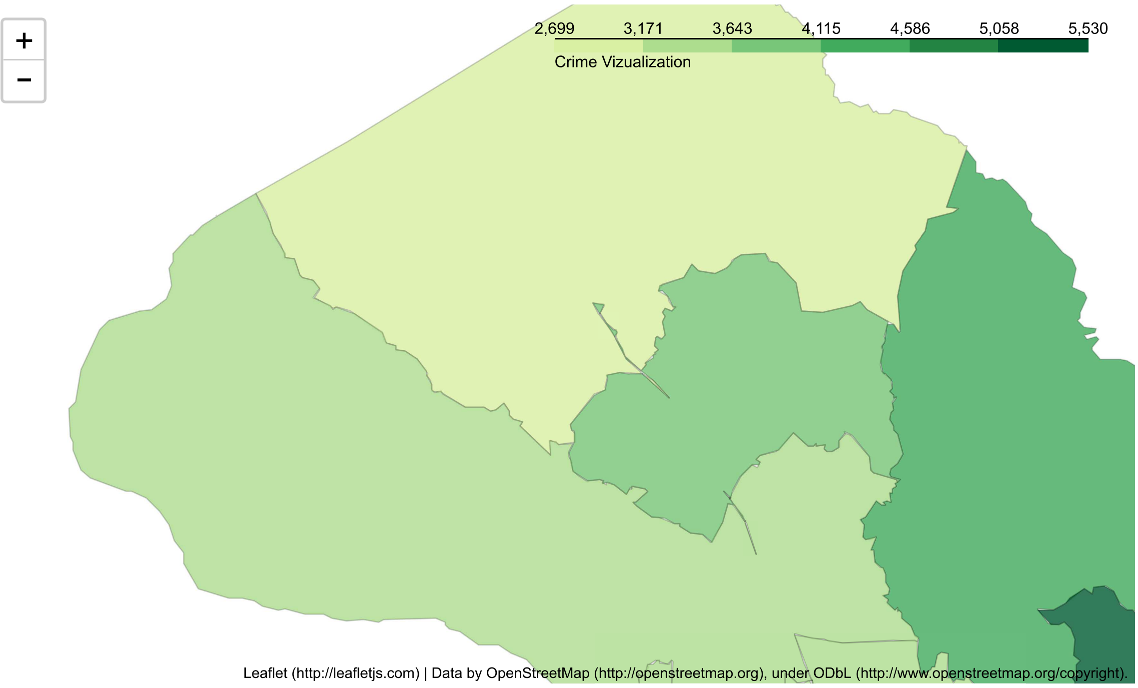
In [72]:

```
def folium_plot(data_out,data):

    # setup county boundries
    map = folium.Map(location=[39.154743, -77.240515], zoom_start=10.5)

    map.choropleth(geo_path = district_geo,
                  data_out = data_out,        # json file with PD boundries
                  data=data,                  # df used for plotting
                  columns = ['DIST', 'Number'],
                  key_on = 'feature.properties.DIST',  # bind the json and data file on district
                  fill_color='YlGn', fill_opacity=0.8, line_opacity=0.2,
                  legend_name='Crime Vizualization')
    return map
```

In [77]:

```
import folium

# Assigning the geojson file to a variable
district_geo = r'montgomery_county_pd.geojson'

# Preparing the data for plotting; this also means reordering the index to match the geojson file
crimedata2 = pd.DataFrame(crimes['Police District Number'].value_counts().astype(float))[:6]
crimedata2.index = [3,4,6,1,2,5]

# Cleaned data to json
crimedata2.to_json('crimeagg_new.json')

# Reset index; rename columns
crimedata2 = crimedata2.reset_index()

crimedata2.columns = ['DIST', 'Number']
# Initiate folium map and then plot it

map=folium_plot(data_out='crimeagg_new.json',data=crimedata2)
map
```

Out[77]:

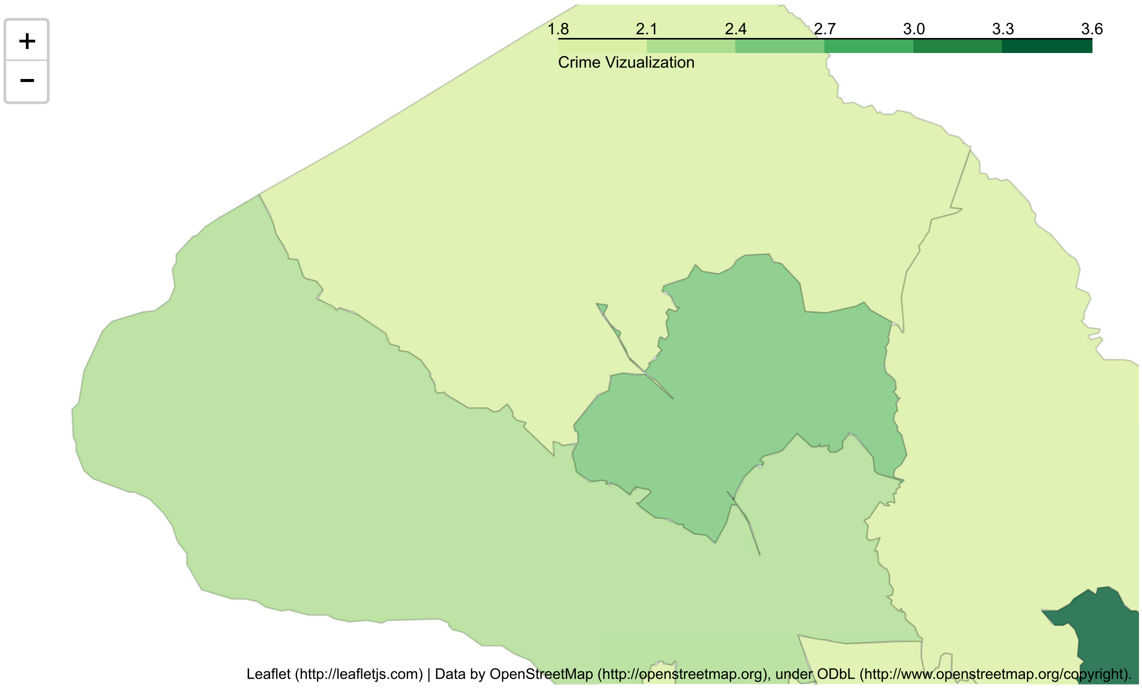


1. This map representation is a little bit misleading in the sense that it does not take into account the population percentage.
2. With this in mind let's do the same thing but try to get the percentage by/100K

In [74]:

```
import folium
district_geo = r'montgomery_county_pd.geojson'

crimedata=crimes['Police District Number'].value_counts()
crimedata=crimedata.iloc[:6]

crimedata=crimedata.to_frame().reset_index()
crimedata.columns=['Police District Number','Number']
crimedata.index=[3,4,6,1,2,5]

#get the crime percentage per 100K
census.index=[1,2,3,4,5,6]
crimedata['Number']=crimedata['Number']*100/census['Population'].astype(float)

json_conv=pd.DataFrame({'Police District Number':crimedata['Number']},index=[3,4,6,1,2])
json_conv.to_json('crimeaggcc.json')


crimedata=crimedata.loc[:,['Police District Number','Number']]
crimedata.columns=['DIST','Number']
crimedata=crimedata.reset_index(drop=True)
crimedata['DIST']=crimedata['DIST'].str.strip('D').astype(float)

map=folium_plot(data_out='crimeaggcc.json',data=crimedata)
map
```

Out[74]:



Conclusions

Analysis conducted on this dataset has 3 directions: time analysis, location analysis and crime classification analysis

Time analysis:

1. Week:
   A. most of the crimes are committed Tuesday and least crimes are Sunday
   B. there is a general decreasing trends towards the end of the weekly

2. Hour:
   A. most of the crimes are committed between 7am -11pm and least crimes between 5am-7am

3. Month:
   A. highest crimes rates are in October
   B. lowest crimes rate in July
   C. amendment dataset is not complete as it contains timep stamps from July onward

4. Dispatch Time conclusion:
   A. highest dispatch time interval is less then 1 min 48.4% followed by a dispatch time above 60 min 41%
   B. smallest dispatch category is between 1 min and 5 min 1.72%

5. Dispatch time deltas based on violent crime subcategories:
   A. less than 1 minutes:
      - highest % HOMICIDE 75%