```
Python 2.7.11 |Anaconda 2.4.1 (x86_64)| (default, Dec  6 2015,
18:57:58)
Type "copyright", "credits" or "license" for more information.

IPython 4.0.1 -- An enhanced Interactive Python.
?         -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help      -> Python's own help system.
object?   -> Details about 'object', use 'object??' for extra
details.
%guiref   -> A brief reference about the graphical user
interface.

In [1]: import pandas as pd
   ...: import numpy as np
   ...: import matplotlib.pylab as plt
   ...: #get_ipython().magic(u'matplotlib inline')
   ...: import collections
   ...: import sklearn.metrics as metrics
   ...: import sklearn.ensemble as ensemble
   ...: import sklearn.linear_model as linear_model
   ...: import sklearn.cross_validation as cross_validation
   ...: import sklearn.grid_search as grid_search
   ...: import sklearn.pipeline as pipeline
   ...: import scipy.stats as stats
   ...: import os
   ...:
   ...:
os.chdir("/Users/taneja/OneDrive/from_copy/machine_learning/wmt")
   ...:
#os.chdir("C:/Users/praveen.taneja/Copy/machine_learning/wmt")
   ...:

In [2]: train = pd.read_table('./data/train.csv', delimiter =
',')
   ...: test = pd.read_table('./data/test.csv', delimiter = ',')
   ...: sample_submission =
pd.read_table('./data/sample_submission.csv',
   ...:                                   delimiter = ',')
   ...:
   ...: print train.head(2)
   ...: print train.tail(2)
   ...: print ' '
   ...: print ' '
   ...: print test.head(2)
   ...: print test.tail(2)
```

```
   ...: print ' '
   ...: print ' '
   ...: print sample_submission.head(2)
   ...: print sample_submission.tail(2)
   ...:
   ...:
   TripType  VisitNumber Weekday          Upc  ScanCount  \
0       999            5  Friday  68113152929         -1
1        30            7  Friday  60538815980          1

  DepartmentDescription  FinelineNumber
0    FINANCIAL SERVICES            1000
1                 SHOES            8931
        TripType  VisitNumber Weekday         Upc  ScanCount  \
647052         8       191347  Sunday  4190007664          1
647053         8       191347  Sunday  3800059655          1

       DepartmentDescription  FinelineNumber
647052                 DAIRY            1512
647053     GROCERY DRY GOODS            3600


   VisitNumber Weekday          Upc  ScanCount
DepartmentDescription  \
0            1  Friday  72503389714          1
SHOES
1            1  Friday   1707710732          1
DAIRY

   FinelineNumber
0            3002
1            1526
        VisitNumber Weekday          Upc  ScanCount
DepartmentDescription  \
653644       191348  Sunday  80469193740          1
SWIMWEAR/OUTERWEAR
653645       191348  Sunday   7871535983          1
MENS WEAR

        FinelineNumber
653644             114
653645            4923


   VisitNumber  TripType_3  TripType_4  TripType_5  TripType_6
TripType_7  \
```

```
0            1           0           0           0           0
0
1            2           0           0           0           0
0

   TripType_8  TripType_9  TripType_12  TripType_14      ...
\
0           0           0            0            0      ...
1           0           0            0            0      ...

   TripType_36  TripType_37  TripType_38  TripType_39
TripType_40  \
0            0            0            0            0
0
1            0            0            0            0
0

   TripType_41  TripType_42  TripType_43  TripType_44
TripType_999
0            0            0            0            0
0
1            0            0            0            0
0

[2 rows x 39 columns]
       VisitNumber  TripType_3  TripType_4  TripType_5
TripType_6  \
95672       191341           0           0           0
0
95673       191348           0           0           0
0

       TripType_7  TripType_8  TripType_9  TripType_12
TripType_14  \
95672           0           0           0            0
0
95673           0           0           0            0
0

           ...       TripType_36  TripType_37  TripType_38
TripType_39  \
95672      ...                 0            0            0
0
95673      ...                 0            0            0
0
```

```
       TripType_40  TripType_41  TripType_42  TripType_43
TripType_44  \
95672            0            0            0            0
0
95673            0            0            0            0
0

       TripType_999
95672             0
95673             0

[2 rows x 39 columns]

In [3]: train_n_rows = train.shape[0]
   ...: # combine because many operations can be done together on
both datasets
   ...: unique_days = train['Weekday'].unique() # for later use
   ...: train_test = pd.concat([train, test], axis = 0)
   ...:

In [4]: print 'missing values in different cols'
   ...: print ' '
   ...: print train_test.isnull().sum()
   ...: print 'train_test.shape =', train_test.shape
   ...: print '% missing values in Upc, FinelineNumber =',
(8115.0/1300700.0)*100
   ...: print '% missing values in DepartmentDescription =',
(2689.0/1300700.0)*100
   ...:
   ...:
missing values in different cols

DepartmentDescription      2689
FinelineNumber             8115
ScanCount                     0
TripType                 653646
Upc                        8115
VisitNumber                   0
Weekday                       0
dtype: int64
train_test.shape = (1300700, 7)
% missing values in Upc, FinelineNumber = 0.623894825863
% missing values in DepartmentDescription = 0.206734835089

In [5]: '''
   ...: We can't drop rows with missing values even though %
```

```
missing is quite small,
   ...: as we are required to predict outcome for all visits in
test data.
   ...: We could do one of the following.
   ...:
   ...: 1. Set them to a distinct new category or value. Eg.
DepartmentDescription =
   ...: No_DepartmentDescription.
   ...: UPC = No_UPC; FinelineNumber = No_FinelineNumber. Makes
least assumptions.
   ...: 2. Set them equal to most probable value for that
category.
   ...: 3. More complicated imputations.
   ...:
   ...: For now lets go with option 1 as there are not so many
missing values.
   ...: '''
   ...: train_test['Upc'] =
train_test['Upc'].fillna('Missing_Upc')
   ...: train_test['FinelineNumber'] =
(train_test['FinelineNumber'].
   ...:
fillna('Missing_FinelineNumber'))
   ...: train_test['DepartmentDescription'] =
(train_test['DepartmentDescription'].
   ...:
fillna('Missing_DepartmentDescription'))
   ...:
   ...: test['Upc'] = test['Upc'].fillna('Missing_Upc')
   ...: test['FinelineNumber'] = (test['FinelineNumber'].
   ...:
fillna('Missing_FinelineNumber'))
   ...: test['DepartmentDescription'] =
(test['DepartmentDescription'].
   ...:
fillna('Missing_DepartmentDescription'))
   ...:
   ...:

In [6]: print len(train_test['TripType'].unique())
   ...: print len(train_test['VisitNumber'].unique())
   ...: print len(train_test['Weekday'].unique())
   ...: print len(train_test['Upc'].unique())
   ...: print len(train_test['ScanCount'].unique())
   ...: print len(train_test['DepartmentDescription'].unique())
   ...: print len(train_test['FinelineNumber'].unique())
```

```
   ...:
   ...: #There are total of 1300700 rows in train_test, one for
each item.
   ...: #But only 191348 unique visit numbers. This is because
many different rows have
   ...: #same visit number. Later we will convert data to wide
format with one row per
   ...: #visit because we need to predict trip type for each
visit
   ...:
39
191348
7
124694
54
69
5354

In [7]: def to_wide(data, groupby_key):
   ...:     ''' The data has many rows that have same visit
number and trip type.
   ...:     These correspond to different items purchased. Column
names are as follows.
   ...:
   ...:     TripType, VisitNumber, Weekday, Upc, ScanCount,
DepartmentDescription,
   ...:     FinelineNumber
   ...:
   ...:     We want each visit to be represented in a single row.
TripType,
   ...:     VisitNumber, Weekday are same for all items in a
single visit. For Upc
   ...:     which is item code we can have columns for all unique
UPCs and each row
   ...:     entry can be sum of ScanCount for that UPC.
Similarly, we
   ...:     can have different columns for different
FinelineNumber and for each row
   ...:     the entry can sum of FinelineNumber for that UPC.
   ...:
   ...:     Implementation - Convert to pandas data frame. Group
by visit number. We
   ...:     can make the dataframe in the begining and fill it
row by row.
   ...:     '''
   ...:
```

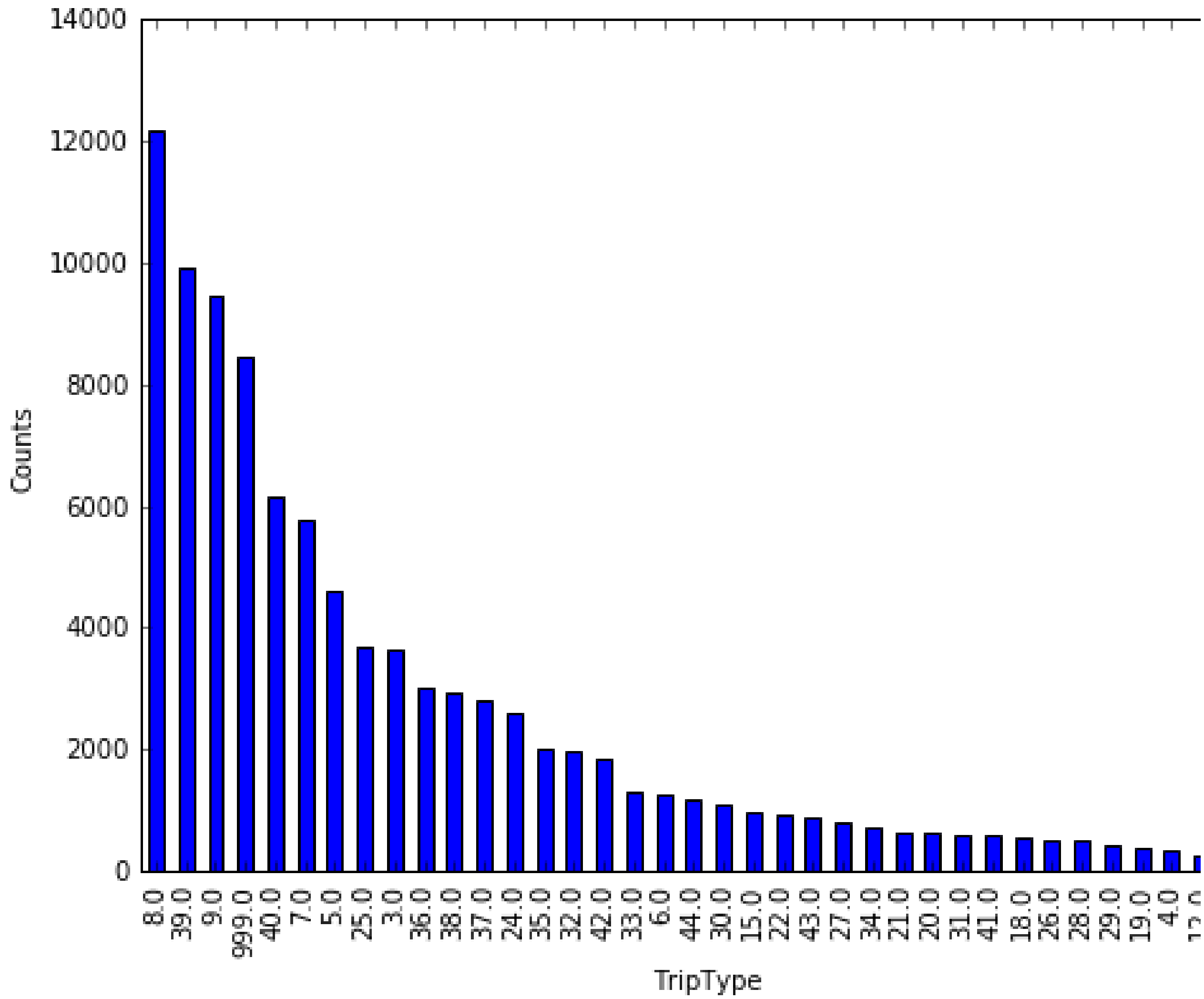
```
   ...:     unique_visits = data['VisitNumber'].unique()
   ...:     unique_days = data['Weekday'].unique()
   ...:     #unique_upcs = data['Upc'].unique()
   ...:     unique_dept_desc =
data['DepartmentDescription'].unique()
   ...:     #unique_fine_line_num =
data['FinelineNumber'].unique()
   ...:
   ...:     # additional columns
   ...:     additional_columns = ['unique_purchases',
'unique_returns',
   ...:                           'total_purchases']
   ...:
   ...:     num_unique_visits = len(unique_visits)
   ...:     print 'num_unique_visits =', num_unique_visits
   ...:
   ...:     cols = ['TripType', 'VisitNumber']
   ...:     for day in unique_days:
   ...:         cols.append(day)
   ...:     for dept_desc in unique_dept_desc:
   ...:         cols.append(str(dept_desc))
   ...:     #for upc in unique_upcs:
   ...:     #    cols.append(upc)
   ...:     #for fine_line_num in unique_fine_line_num:
   ...:     #    cols.append(str(fine_line_num))
   ...:     for additional_column in additional_columns:
   ...:         cols.append(str(additional_column))
   ...:
   ...:
   ...:     # initialize
   ...:     d = collections.OrderedDict()
   ...:     for col in cols:
   ...:         d[col] = [0]*num_unique_visits
   ...:
   ...:     grouped = data.groupby(groupby_key)
   ...:     #print 'groupby_key', groupby_key
   ...:
   ...:     i = 0
   ...:     for name, group in grouped:
   ...:
   ...:         d['TripType'][i] =  group['TripType'].iloc[0]
   ...:         d['VisitNumber'][i] =
group['VisitNumber'].iloc[0]
   ...:
   ...:         day = group['Weekday'].iloc[0]
   ...:         d[day][i] =  1
```

```
   ...:
   ...:         depts = group['DepartmentDescription'].unique()
   ...:         for dept in depts:
   ...:
   ...:             d[str(dept)][i] =
(group['DepartmentDescription']
   ...:
[group['DepartmentDescription'] == dept].
   ...:                                  count())
   ...:
   ...:         '''
   ...:
   ...:         fine_line_nums = group['FinelineNumber'].unique()
   ...:         for fine_line_num in fine_line_nums:
   ...:             d[str(fine_line_num)][i] =
   ...:             group['ScanCount'][group['FinelineNumber'] ==
fine_line_num].count()
   ...:
   ...:         '''
   ...:         upcs = group['Upc'][group['ScanCount'] >=
0].unique()
   ...:         d['unique_purchases'][i] = len(upcs)
   ...:
   ...:         upcs_returned = group['Upc'][group['ScanCount'] <
0].unique()
   ...:         d['unique_returns'][i] = len(upcs_returned)
   ...:
   ...:         upcs = group['ScanCount'][group['ScanCount'] >=
0]
   ...:         d['total_purchases'][i] = upcs.sum()
   ...:
   ...:
   ...:         i = i + 1
   ...:
   ...:     return pd.DataFrame(d)
   ...:
   ...:
   ...:
   ...:


In [8]: train_test_wide =  to_wide(train_test, 'VisitNumber')
   ...: # save to disk so we don't have to create it again the
next time
   ...: train_test_wide.to_csv('./data/train_test_wide.csv',
index = False)
   ...:
```

```
num_unique_visits = 191348

In [9]: train_test_wide =
pd.read_table('./data/train_test_wide.csv', delimiter = ',')
   ...:

In [10]: train =
train_test_wide[train_test_wide['TripType'].notnull()]
    ...: test =
train_test_wide[train_test_wide['TripType'].isnull()]
    ...: print sorted(train_test_wide['TripType'].unique())
    ...:
    ...:
[nan, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 12.0, 14.0, 15.0, 18.0,
19.0, 20.0, 21.0, 22.0, 23.0, 24.0, 25.0, 26.0, 27.0, 28.0, 29.0,
30.0, 31.0, 32.0, 33.0, 34.0, 35.0, 36.0, 37.0, 38.0, 39.0, 40.0,
41.0, 42.0, 43.0, 44.0, 999.0]

In [11]: fig = plt.figure(figsize = (8,6))
    ...: fig =
train_test_wide['TripType'].value_counts().plot(kind = 'bar')
    ...: plt.xlabel('TripType')
    ...: plt.ylabel('Counts')
    ...: print train_test_wide['TripType'].value_counts()
    ...: '''
    ...: TripType8 is most common, TripType14 is least (has only
4 visits!)
    ...: '''
    ...:
8      12161
39      9896
9       9464
999     8444
40      6130
7       5752
5       4593
25      3698
3       3643
36      3005
38      2912
37      2788
24      2609
35      2030
32      1984
42      1858
33      1315
```

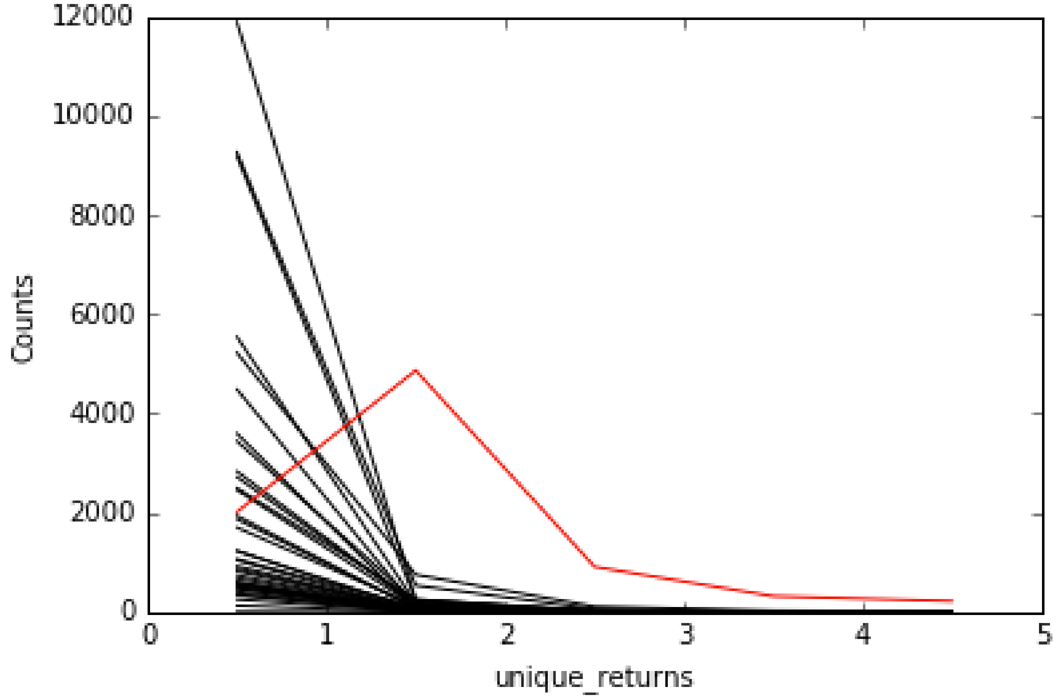
```
6       1277
44      1187
30      1081
15       978
22       928
43       872
27       785
34       719
21       641
20       637
31       594
41       583
18       549
26       503
28       492
29       433
19       375
4        346
12       269
23       139
14         4
Name: TripType, dtype: int64
```

```
Out[11]: '\nTripType8 is most common, TripType14 is least (has
only 4 visits!)\n'
```




```
In [12]: grouped = train.groupby(['TripType'])
    ...:
    ...: for name, group in grouped:
    ...:     #plt.figure()
    ...:     color = 'black'
    ...:     if name == 999:
    ...:         color = 'red'
    ...:     vals, binEdges=np.histogram(group['unique_returns'],
bins=5, range = [0, 5])
    ...:     bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    ...:     plt.plot(bincenters,vals,'-', color = color)
    ...:     plt.xlim([0, 5])
    ...:     #group['unique_returns'].hist(range = [0, 5], normed
= 'True', label = name)#value_counts().plot
    ...:     #plt.legend()
    ...:
```

```
   ...: plt.xlabel('unique_returns')
   ...: plt.ylabel('Counts')
   ...:
   ...: '''
   ...: Compared to other TripTypes, 999 has most unique returns
(ScanCount < 0) .
   ...: Although in many cases TripTypes = 999, even when
ScanCount >0 for all purchases)
   ...: Eg. 999   207 Friday  76163520390 1   PRODUCE
   ...:     999   207 Friday  2242229000  2   DSD GROCERY
   ...:
   ...:     999   295 Friday  1019906311  1   OFFICE SUPPLIES
   ...:
   ...:     999   351 Friday  68113178251 1   FINANCIAL SERVICES
   ...:     999   351 Friday  68113178252 1   FINANCIAL SERVICES
   ...:     999   357 Friday  60538807733 1   FINANCIAL SERVICES
   ...:     999   357 Friday  68113107941 1   FINANCIAL SERVICES
   ...: '''
   ...:
Out[12]: '\nCompared to other TripTypes, 999 has most unique
returns (ScanCount < 0) .\nAlthough in many cases TripTypes =
999, even when ScanCount >0 for all purchases)\nEg.
999\t207\tFriday\t76163520390\t1\tPRODUCE\n
999\t207\tFriday\t2242229000\t2\tDSD GROCERY\n    \n
999\t295\tFriday\t1019906311\t1\tOFFICE SUPPLIES\n    \n
999\t351\tFriday\t68113178251\t1\tFINANCIAL SERVICES\n
999\t351\tFriday\t68113178252\t1\tFINANCIAL SERVICES\n
999\t357\tFriday\t60538807733\t1\tFINANCIAL SERVICES\n
999\t357\tFriday\t68113107941\t1\tFINANCIAL SERVICES\n'
```

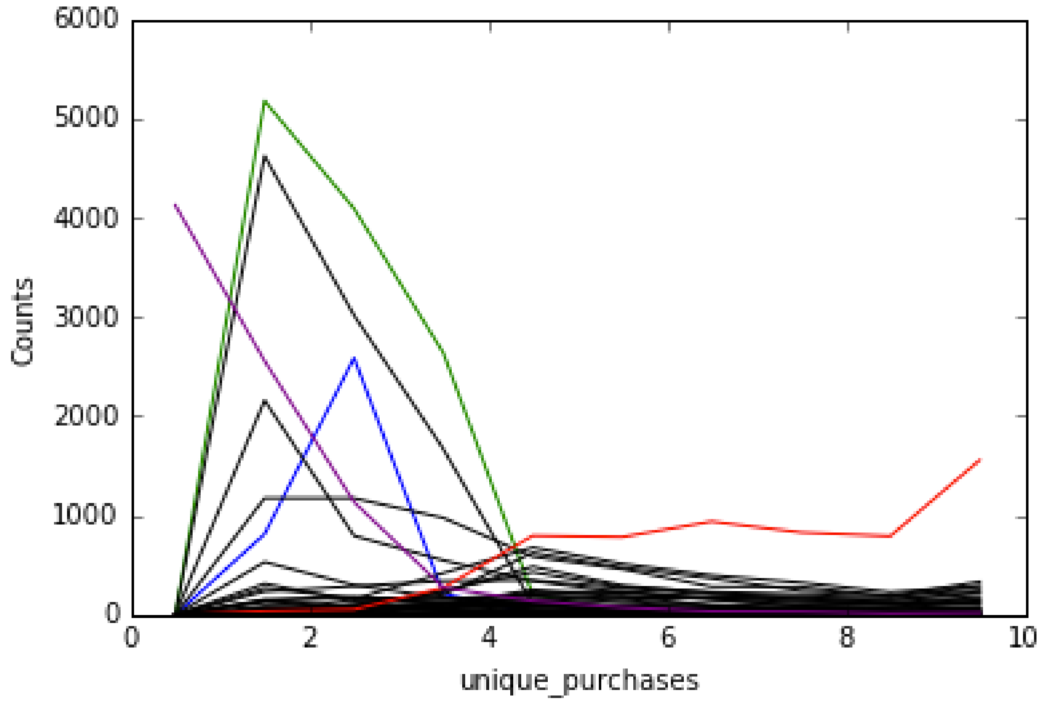



```
In [13]: grouped = train.groupby(['TripType'])
    ...: color = 'black'
    ...: for name, group in grouped:
    ...:     color = 'black'
    ...:     if name == 39: # 2nd most popular trip type
    ...:         color = 'red'
    ...:     if name == 3:
    ...:         color = 'blue'
    ...:     if name == 8: # most popular trip type
    ...:         color = 'green'
    ...:     if name == 999: # TripTypes with most returns
    ...:         color = 'purple'
    ...:     vals,
binEdges=np.histogram(group['unique_purchases'], bins=10, range =
[0, 10])
    ...:     bincenters = 0.5*(binEdges[1:]+binEdges[:-1])
    ...:     print 'TripType', name
    ...:     print vals
    ...:     plt.plot(bincenters,vals,'-', color = color)
    ...:     plt.xlim([0, 10])
    ...:     #group['unique_returns'].hist(range = [0, 5], normed
= 'True', label = name)#value_counts().plot
    ...:     #plt.legend()
    ...:
    ...: plt.xlabel('unique_purchases')
    ...: plt.ylabel('Counts')
```

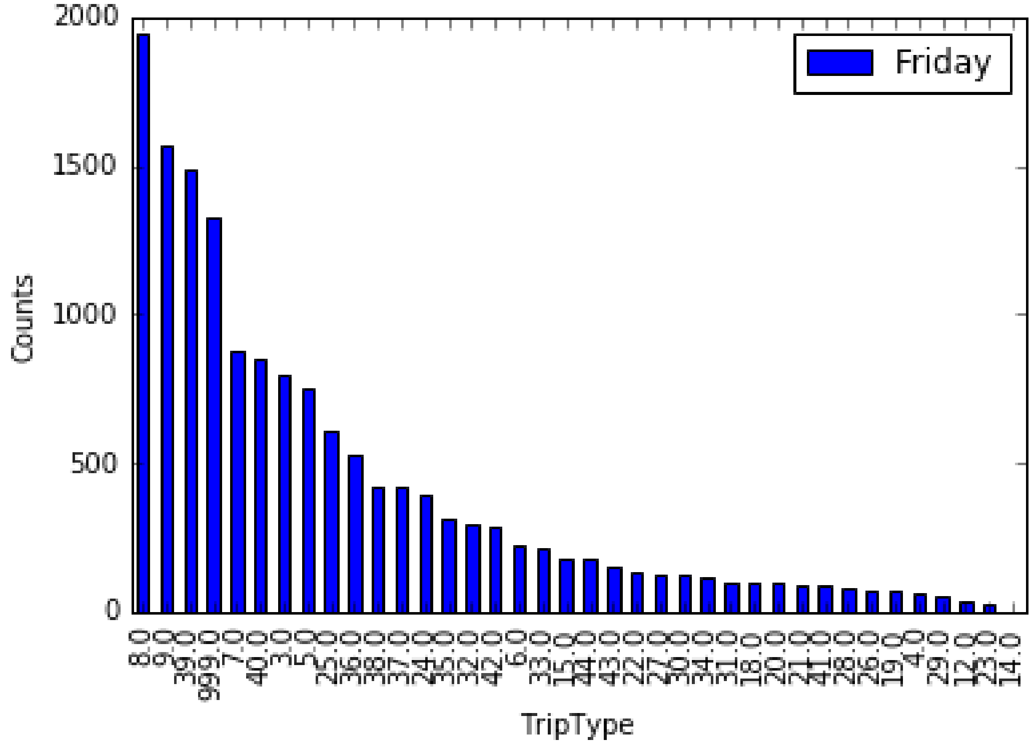
```
   ...:
   ...: '''
   ...: While most trips have 1 unique purchase, TripType3 has 2
unique most often.
   ...: TripType39 has biggest number of unique purchases.
TripType39 is also the
   ...: 2nd most popular TripType. Likely TripType39 is weekly
shopping.
   ...: TripType999 (unlike other trips) has 0 as the most
unique purchase.
   ...:
   ...: '''
   ...:
TripType 3.0
[   0  814 2590  214   18    3    2    1    0    0]
TripType 4.0
[  0 148  78  40  26  20  14   6   6   6]
TripType 5.0
[   0 2161  797  550  342  226  157   87   77   79]
TripType 6.0
[  0 533 300 172 103  53  36  25  19  18]
TripType 7.0
[   0 1176 1176  978  587  462  285  282  192  334]
TripType 8.0
[   0 5178 4092 2639  203   35    8    2    0    2]
TripType 9.0
[   0 4622 3018 1671  109   32    5    3    2    1]
TripType 12.0
[ 0  5 12 30 46 37 26 15 14 25]
TripType 14.0
[0 0 0 0 0 2 1 0 0 1]
TripType 15.0
[  0  46  57 106 166 154 103  70  47  65]
TripType 18.0
[ 0 43 74 67 99 73 63 37 19 27]
TripType 19.0
[  0 129  82  45  38  26  14  11  10  11]
TripType 20.0
[  0  37 102 111 123  82  56  40  24  30]
TripType 21.0
[  0  27  52  74 149  80  70  39  34  41]
TripType 22.0
[  0 268 170 120 135  78  40  19  23  25]
TripType 23.0
[ 0 63 40 11  8  8  5  2  0  1]
TripType 24.0
```

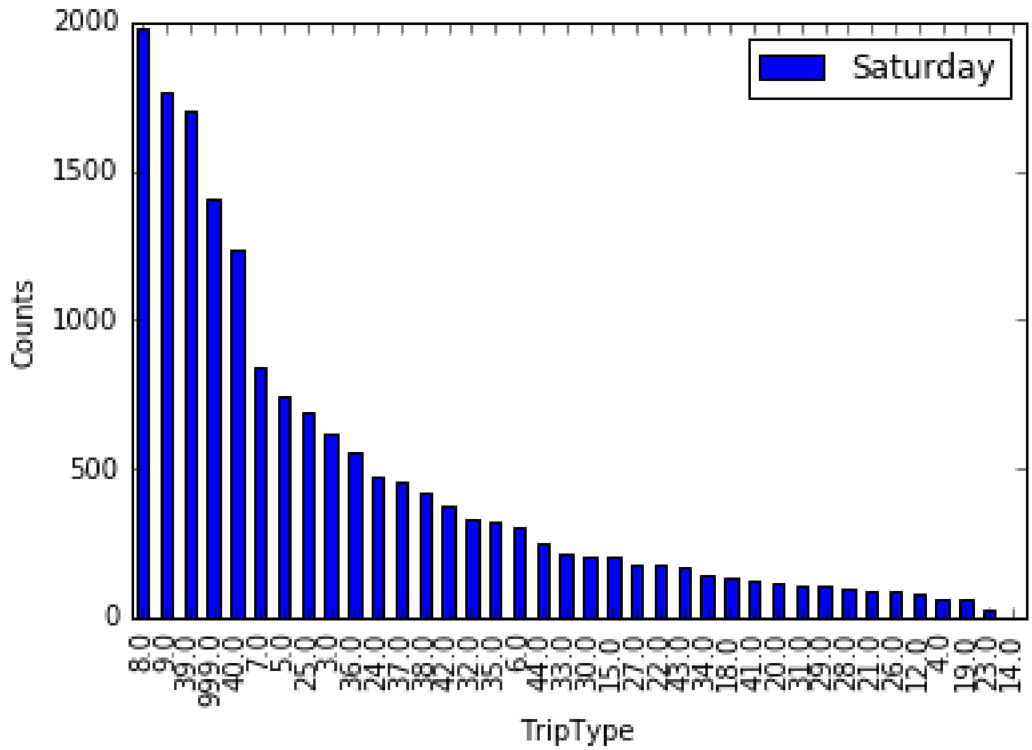
```
[  0 237 289 342 349 271 205 184 128 181]
TripType 25.0
[  0  19 160 422 680 516 399 326 231 291]
TripType 26.0
[  0  48  62  66 106  80  42  28  21  25]
TripType 27.0
[  0  88  72 123 111  88  57  48  45  63]
TripType 28.0
[ 0 58 46 80 97 56 36 30 25 21]
TripType 29.0
[ 0 46 71 73 83 51 29 22  9 17]
TripType 30.0
[  0 139 174 164 199 129  89  48  41  49]
TripType 31.0
[  0 155 185 123  52  26  10  12   5  13]
TripType 32.0
[  0 314 146 158 211 227 170 133 112 163]
TripType 33.0
[  0  21  26 101 255 180 162  90  81 135]
TripType 34.0
[  0  26  64  80 119 102  68  67  43  49]
TripType 35.0
[  0  67 107 233 487 297 223 156 107 147]
TripType 36.0
[  0  35  98 203 626 479 367 244 196 249]
TripType 37.0
[  0  17  48 126 228 240 164 161 151 216]
TripType 38.0
[  0  18  54 250 430 282 232 208 177 291]
TripType 39.0
[   0   39   55  274  798  785  941  831  794 1561]
TripType 40.0
[ 0  0  3  1  8  4 12 14 15 47]
TripType 41.0
[ 0  1  2 29 68 78 67 47 49 79]
TripType 42.0
[  0  13  11  71 199 176 175 161 156 232]
TripType 43.0
[  0   0   3   3 120 195  92 112 122 116]
TripType 44.0
[  0   0   1   0  10  24  21  32  70 124]
TripType 999.0
[4127 2559 1137  276  141   76   37   33   15   22]
Out[13]: '\nWhile most trips have 1 unique purchase, TripType3
has 2 unique most often.\nTripType39 has biggest number of unique
purchases. TripType39 is also the\n2nd most popular TripType.
```

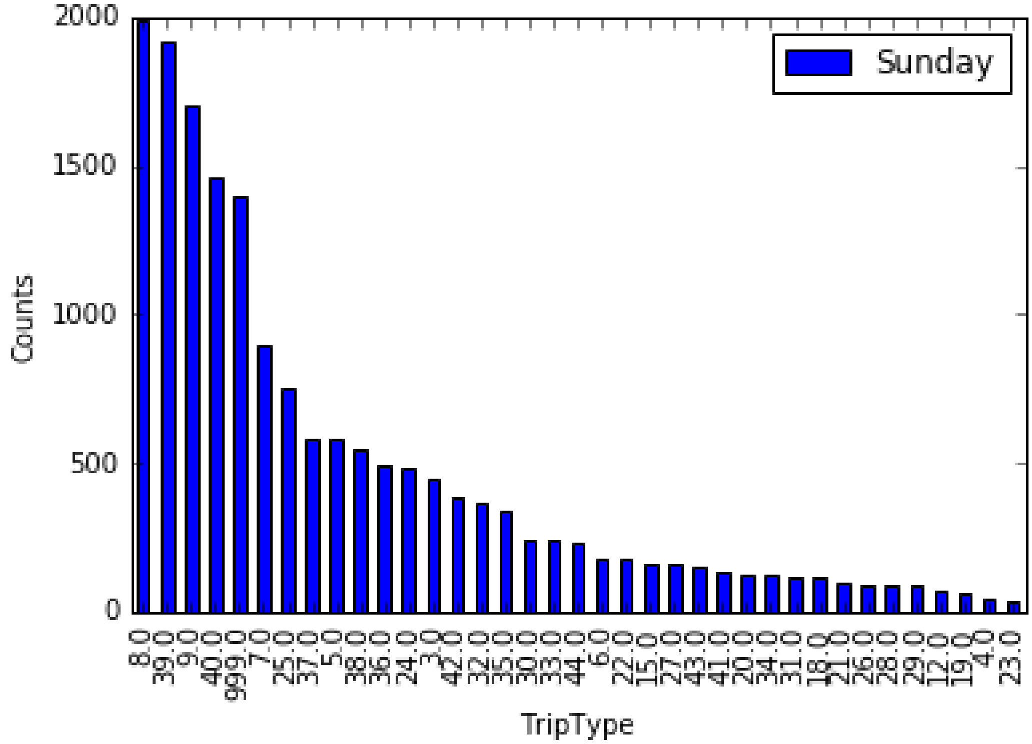
Likely TripType39 is weekly shopping.\nTripType999 (unlike other trips) has 0 as the most unique purchase.\n\n'

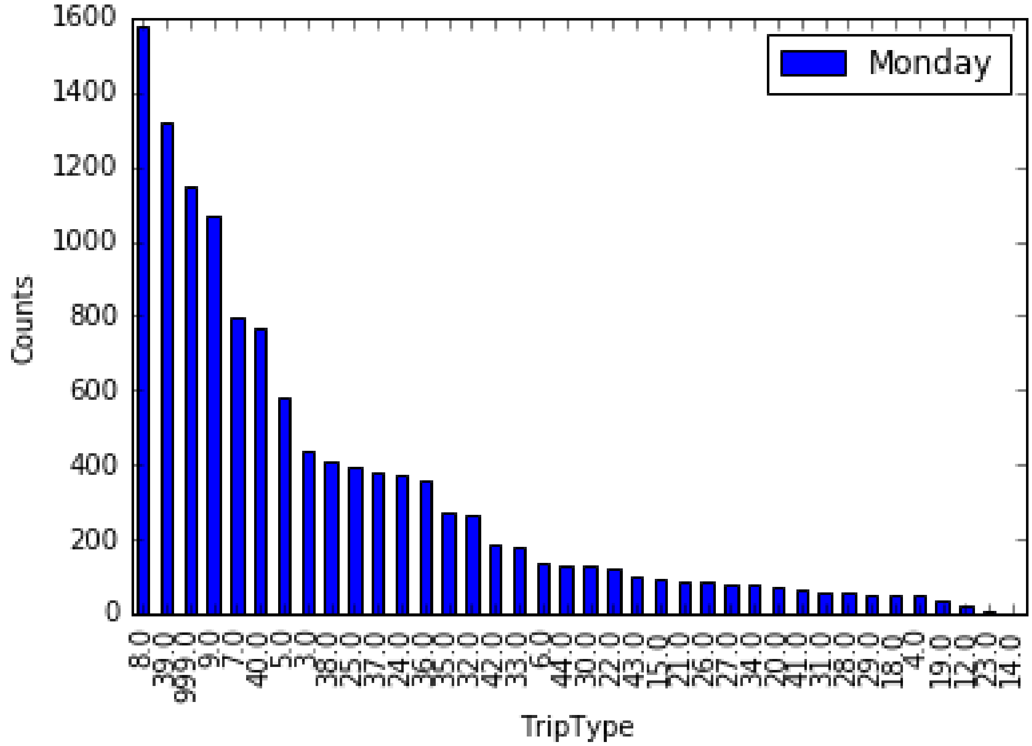


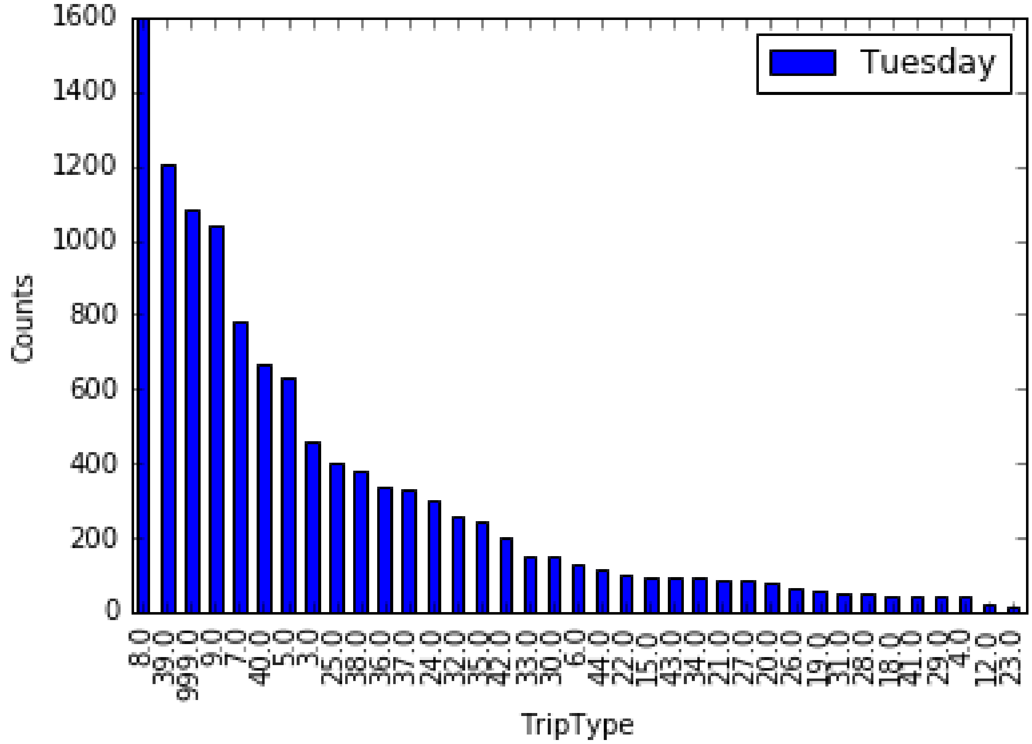

```
In [14]: for day in unique_days:
    ...:     fig = plt.figure()
    ...:     day_data = train[train[day] == 1]
    ...:     day_data['TripType'].value_counts().plot(kind =
'bar', label = day)
    ...:     plt.xlabel('TripType')
    ...:     plt.ylabel('Counts')
    ...:     plt.legend()
    ...:
    ...:
    ...: '''
    ...: Trip types don't seem to depend that much on days of the
week. For example,
    ...: Trip types 8, 39, 9, 999 are in top 5 every day. Though
the overall number of
    ...: visits are higher on Friday, Saturday, Sunday.
    ...: '''
    ...:
Out[14]: "\nTrip types don't seem to depend that much on days of
the week. For example, \nTrip types 8, 39, 9, 999 are in top 5
every day. Though the overall number of\nvisits are higher on
Friday, Saturday, Sunday.\n"
```

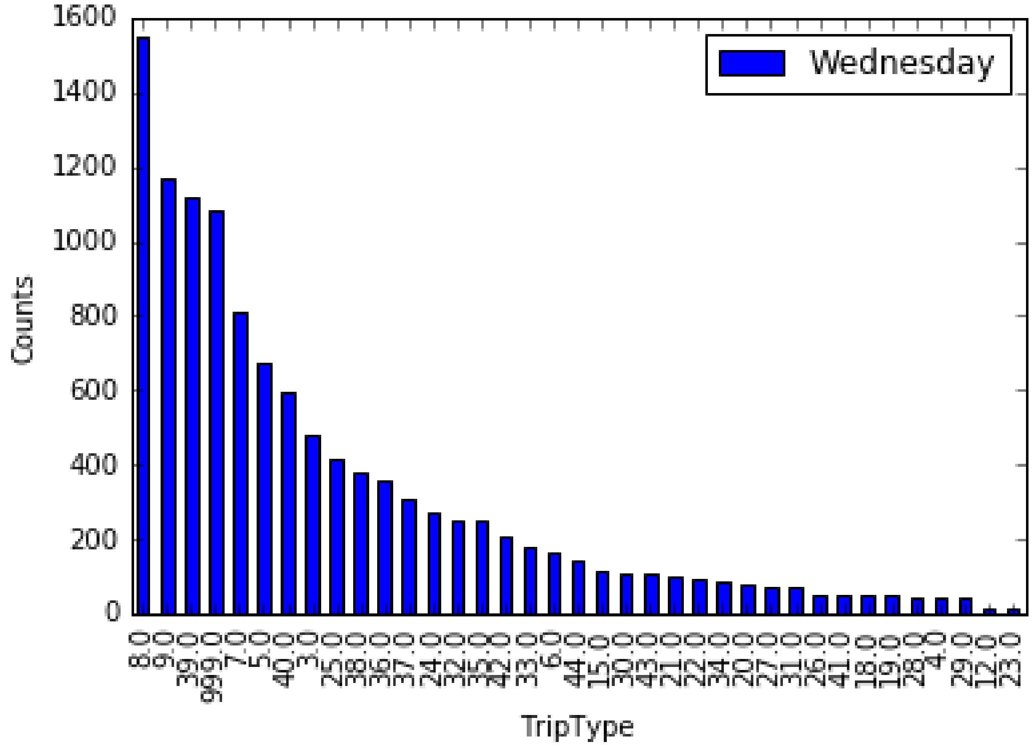

Friday
Counts
2000
1500
1000
500
0
TripType

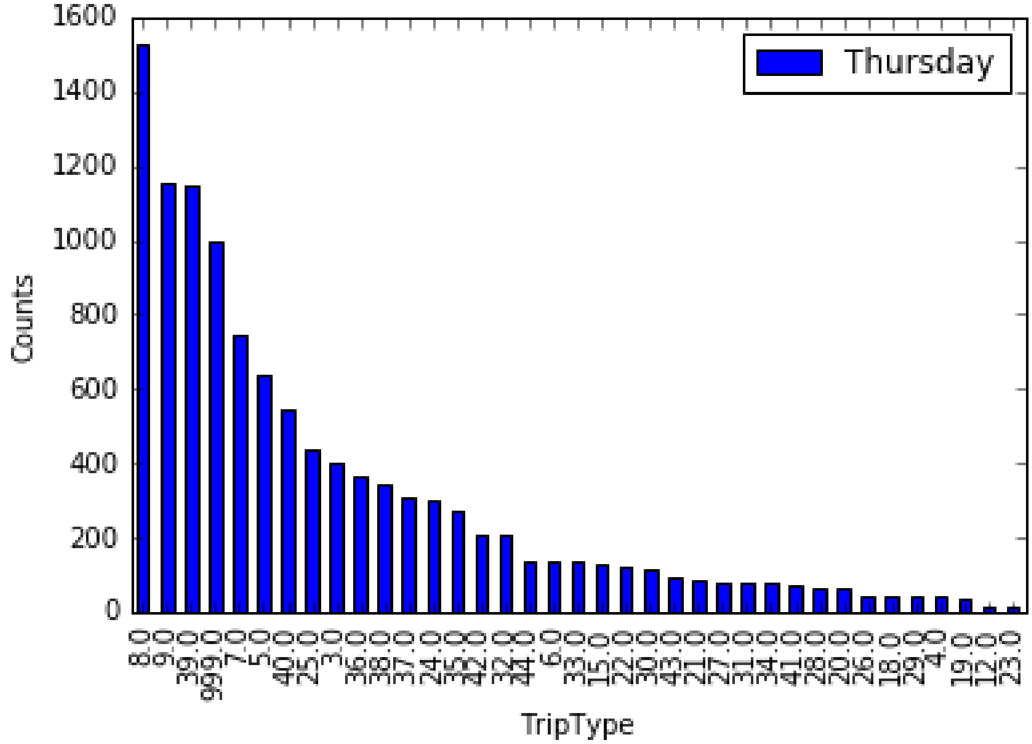

Saturday
Counts
2000
1500
1000
500
0
TripType


Sunday
Counts
TripType
0
500
1000
1500
2000

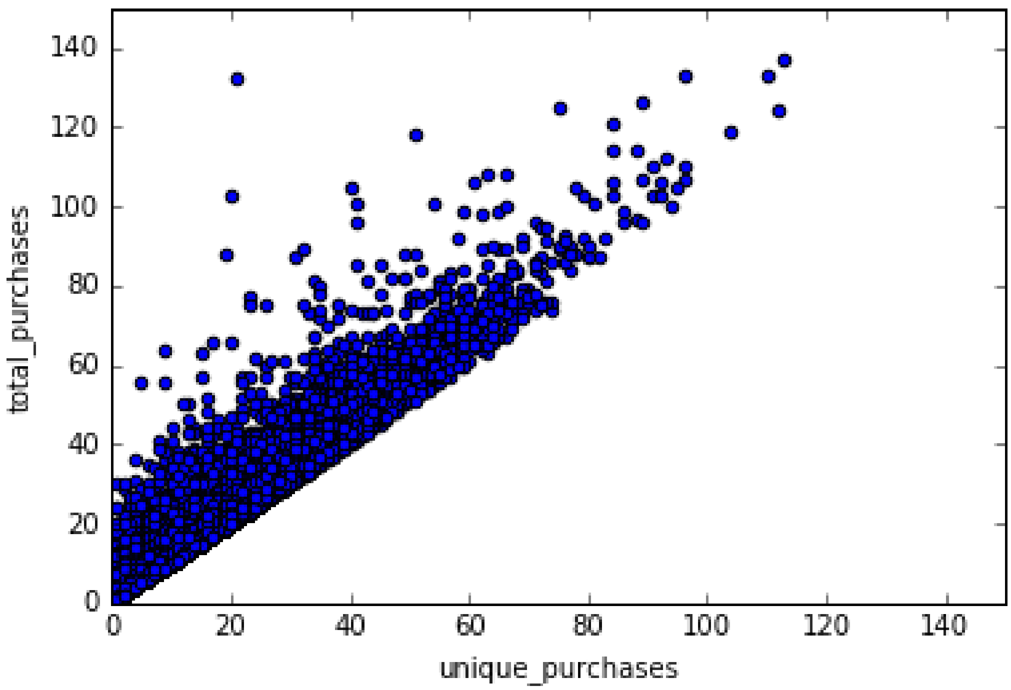

Monday
Counts
TripType
0
200
400
600
800
1000
1200
1400
1600


Tuesday
Counts
1600
1400
1200
1000
800
600
400
200
0
8.0 39.0 999.0 9.0 7.0 40.0 5.0 3.0 25.0 38.0 36.0 37.0 24.0 32.0 35.0 42.0 33.0 30.0 6.0 44.0 22.0 15.0 43.0 34.0 21.0 27.0 20.0 26.0 19.0 31.0 28.0 18.0 41.0 29.0 4.0 12.0 23.0
TripType


Wednesday
Counts
1600
1400
1200
1000
800
600
400
200
0
8.0 9.0 39.0 999.0 7.0 5.0 40.0 3.0 25.0 38.0 36.0 37.0 24.0 32.0 35.0 42.0 33.0 6.0 44.0 15.0 30.0 43.0 21.0 22.0 34.0 20.0 27.0 31.0 26.0 41.0 18.0 19.0 28.0 4.0 29.0 12.0 23.0
TripType




```
In [15]: train.plot('unique_purchases', 'total_purchases', kind =
'scatter')
    ...: plt.xlim((0, 150))
    ...: plt.ylim((0, 150))
    ...: y1 = train['unique_purchases'].values
    ...: y2 = train['total_purchases'].values
    ...: print 'Correlation between unique and total purchases
=', stats.pearsonr(y1, y2)
    ...:
    ...: # Dropping total_purchases as it's highly correlated to
unique_purchases
    ...: train = train.drop('total_purchases', axis = 1)
    ...: test = test.drop('total_purchases', axis = 1)
    ...:
    ...:
Correlation between unique and total purchases =
(0.97298096998505879, 0.0)
```



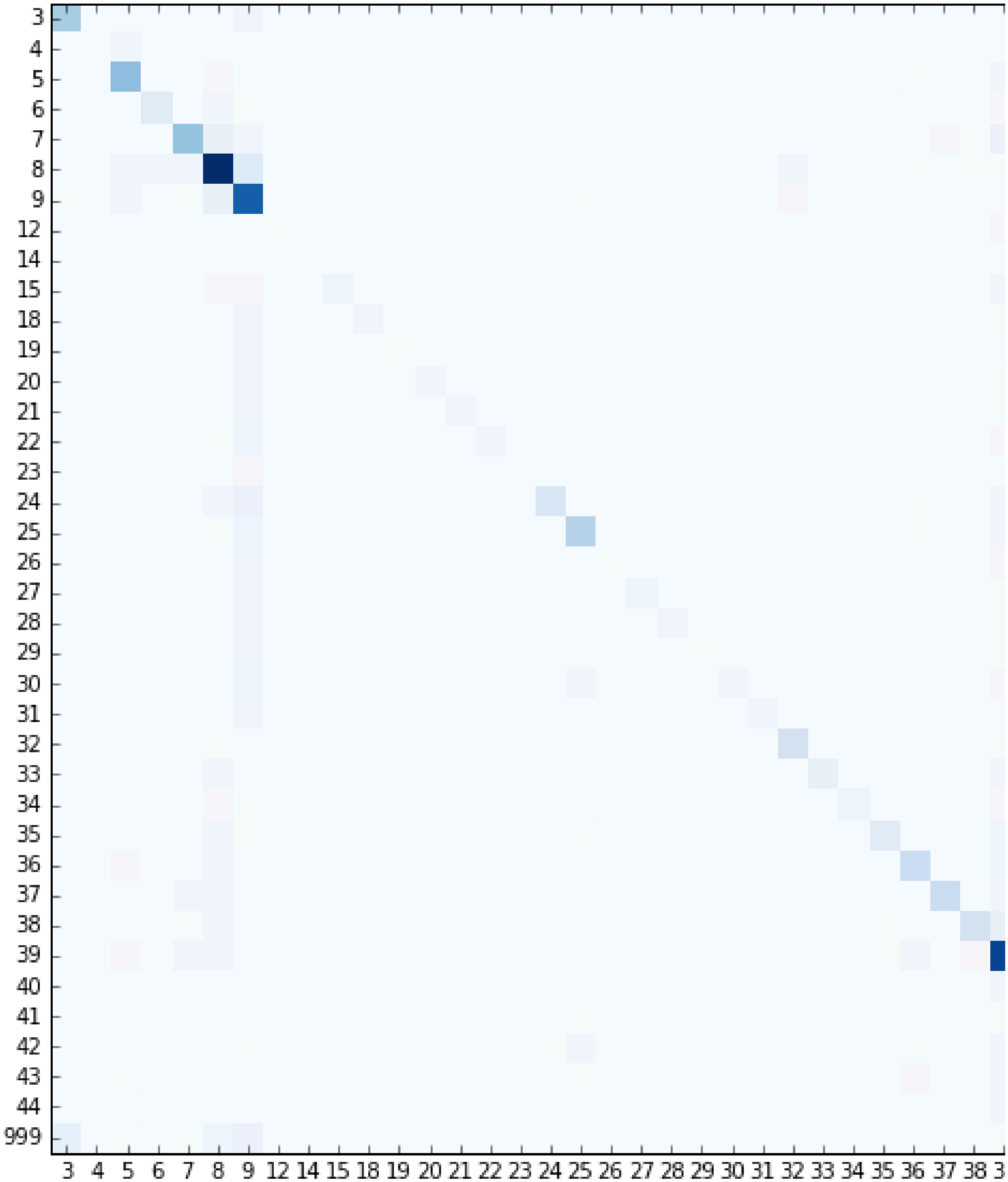

```
In [16]: y_train = train['TripType'].values
    ...: X_train = train.drop(['TripType', 'VisitNumber'], axis =
1).values
    ...: X_test = test.drop(['TripType', 'VisitNumber'], axis =
1).values
    ...:
    ...: clf = ensemble.RandomForestClassifier(n_estimators=50)
    ...: '''
    ...: clf.fit(X_train, y_train)
    ...: important_features = clf.feature_importances_
    ...:
    ...: std = np.std([tree.feature_importances_ for tree in
clf.estimators_], axis = 0)
    ...:
    ...: indices = np.argsort(important_features)[::-1] # sort in
descending order
    ...: col_names = (train.drop(['TripType', 'VisitNumber'],
axis = 1).
    ...:                            columns.values)
    ...:
    ...: for col in range(X_train.shape[1]):
    ...:     print (col, indices[col], col_names[indices[col]],
    ...:     important_features[indices[col]])
    ...:
    ...: plt.bar(range(X_train.shape[1]),
important_features[indices],
```

```
    ...:         yerr= std[indices], color = 'red')
    ...:
    ...: top_features = col_names[indices][0:40]
    ...: top_features_w_TripType = ['TripType'] +
list(top_features)
    ...: '''
    ...: #######train = train[top_features_w_TripType]
    ...: unique_visits = test['VisitNumber'] # to be used later
in output file
    ...: #######test = test[top_features]
    ...:
    ...: ########y_train = train['TripType'].values
    ...: # top features doesn't have VisitNumber in it, so no
need to drop
    ...: ########X_train = train.drop(['TripType'], axis =
1).values
    ...: ########X_test = test.values
    ...:

In [17]: scores = cross_validation.cross_val_score(clf, X_train,
y_train, cv = 5)
    ...: print 'cross_val score', scores, scores.mean()
    ...:
cross_val score [ 0.6421223   0.6449018   0.63097664  0.64950071
0.64314484] 0.642129258752
/Users/taneja/anaconda/lib/python2.7/site-
packages/sklearn/cross_validation.py:516: Warning: The least
populated class in y has only 4 members, which is too few. The
minimum number of labels for any class cannot be less than
n_folds=5.
  % (min_labels, self.n_folds)), Warning)

In [17]:

In [18]: clf.fit(X_train, y_train)
    ...: predicted_train_classes = clf.predict(X_train)
    ...: print 'training error', metrics.accuracy_score(y_train,
predicted_train_classes)
    ...:
training error 0.912578129899

In [19]: pars = {'max_depth': [5, 10, 15, 20]}
    ...: clf = grid_search.GridSearchCV(clf, param_grid = pars,
verbose = 5) #RandomizedSearchCV(clf, pars)
    ...: clf.fit(X_train, y_train)
    ...:
```

```
Fitting 3 folds for each of 4 candidates, totalling 12 fits
[CV] max_depth=5
.....................................................
[CV] ............................ max_depth=5, score=0.429624 -
2.9s
[CV] max_depth=5
.....................................................
[CV] ............................ max_depth=5, score=0.462498 -
2.9s
[CV] max_depth=5
.....................................................
[CV] ............................ max_depth=5, score=0.424910 -
2.9s
[CV] max_depth=10
....................................................
[CV] ........................... max_depth=10, score=0.560410 -
4.2s
[CV] max_depth=10
....................................................
[CV] ........................... max_depth=10, score=0.551580 -
4.2s
[CV] max_depth=10
....................................................
[CV] ........................... max_depth=10, score=0.557396 -
4.2s
[CV] max_depth=15
....................................................
[CV] ........................... max_depth=15, score=0.595794 -
5.4s
[CV] max_depth=15
....................................................
[CV] ........................... max_depth=15, score=0.591214 -
5.4s
[CV] max_depth=15
....................................................
[CV] ........................... max_depth=15, score=0.600094 -
5.4s
[CV] max_depth=20
....................................................
[CV] ........................... max_depth=20, score=0.619770 -
6.5s
[CV] max_depth=20
....................................................
[CV] ........................... max_depth=20, score=0.614010 -
6.5s
[CV] max_depth=20
```

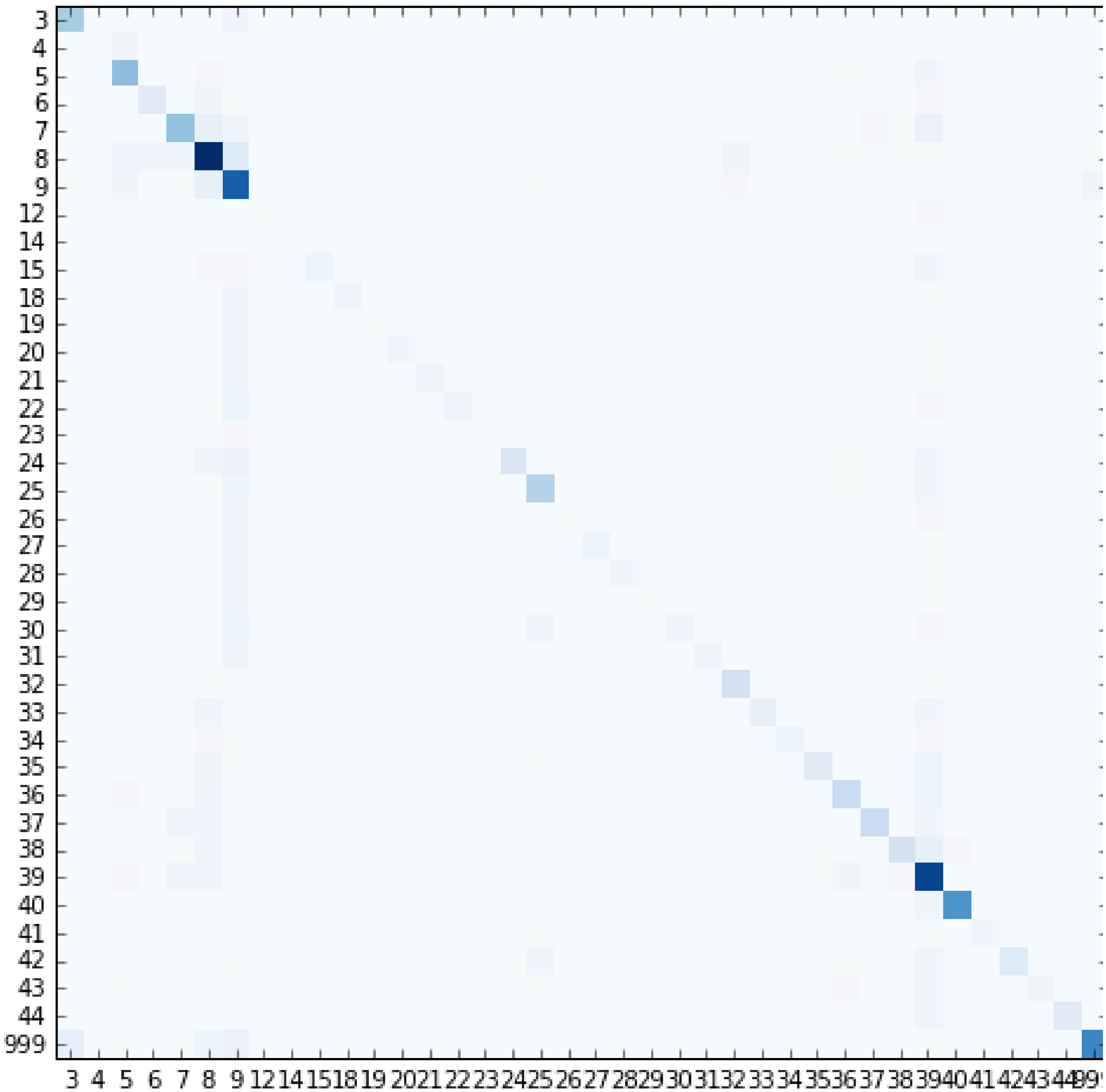
```
....................................................
[CV] ........................... max_depth=20, score=0.621553 -
6.6s
[Parallel(n_jobs=1)]: Done  12 out of  12 | elapsed:   57.4s
finished
Out[19]:
GridSearchCV(cv=None, error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True,
class_weight=None, criterion='gini',
            max_depth=None, max_features='auto',
max_leaf_nodes=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=50,
n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params={}, iid=True, n_jobs=1,
       param_grid={'max_depth': [5, 10, 15, 20]},
pre_dispatch='2*n_jobs',
       refit=True, scoring=None, verbose=5)

In [20]: predicted_train_classes = clf.predict(X_train)
    ...: print 'training error (after tuning)',
metrics.accuracy_score(y_train, predicted_train_classes)
    ...:
    ...:
    ...:
training error (after tuning) 0.744591006961

In [21]: predicted_proba = clf.predict_proba(X_test)
    ...: print predicted_proba.shape
    ...:
(95674, 38)

In [22]: unique_trip_types =  np.sort(train['TripType'].unique())
    ...: unique_trip_types =  ['TripType_'+ str(int(t)) for t in
unique_trip_types]
    ...: print unique_trip_types
    ...:
    ...: df_predicted = pd.DataFrame(predicted_proba, columns =
unique_trip_types)
    ...: df_unique_visits =
pd.DataFrame(unique_visits).reset_index(drop = True)
    ...: df_predicted = pd.concat([df_unique_visits,
df_predicted], axis = 1)
    ...: print df_predicted.shape
```

```
    ...: df_predicted.to_csv('./data/test_predicted.csv', index =
False)
    ...:
    ...:
['TripType_3', 'TripType_4', 'TripType_5', 'TripType_6',
'TripType_7', 'TripType_8', 'TripType_9', 'TripType_12',
'TripType_14', 'TripType_15', 'TripType_18', 'TripType_19',
'TripType_20', 'TripType_21', 'TripType_22', 'TripType_23',
'TripType_24', 'TripType_25', 'TripType_26', 'TripType_27',
'TripType_28', 'TripType_29', 'TripType_30', 'TripType_31',
'TripType_32', 'TripType_33', 'TripType_34', 'TripType_35',
'TripType_36', 'TripType_37', 'TripType_38', 'TripType_39',
'TripType_40', 'TripType_41', 'TripType_42', 'TripType_43',
'TripType_44', 'TripType_999']
(95674, 39)

In [23]: cm = metrics.confusion_matrix(y_train,
predicted_train_classes)
    ...: plt.figure(figsize = (12, 10))
    ...: plt.imshow(cm, interpolation='nearest',
cmap=plt.cm.Blues)
    ...: unique_trip_types =  np.sort(train['TripType'].unique())
    ...: unique_trip_types = unique_trip_types.astype(int)
    ...: tick_marks = np.arange(len(unique_trip_types))
    ...: plt.xticks(tick_marks, unique_trip_types, rotation=0)
    ...: plt.yticks(tick_marks, unique_trip_types)
    ...: plt.colorbar()
    ...:
Out[23]: <matplotlib.colorbar.Colorbar at 0x12ead4310>
```




```
In [24]: cm = metrics.confusion_matrix(y_train,
predicted_train_classes)
    ...: plt.figure(figsize = (10, 8))
    ...: plt.imshow(cm, interpolation='nearest',
```

```
cmap=plt.cm.Blues)
    ...: unique_trip_types =  np.sort(train['TripType'].unique())
    ...: unique_trip_types = unique_trip_types.astype(int)
    ...: tick_marks = np.arange(len(unique_trip_types))
    ...: plt.xticks(tick_marks, unique_trip_types, rotation=0)
    ...: plt.yticks(tick_marks, unique_trip_types)
    ...: plt.colorbar()
    ...:
Out[24]: <matplotlib.colorbar.Colorbar at 0x12ea42290>
```




```
In [25]:
```