In [ ]:

```
'''
This file is for Exploratory Data Analysis
@ Author: Shuyi Wang
@ Date: 2017/3/18
'''
import pandas as pd
import warnings # current version of seaborn generates a bunch of warnings that we'll i
gnore
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
sns.set(style="white", color_codes=True)

%matplotlib inline
```

For this competition, you are tasked with categorizing shopping trip types based on the items that customers purchased. To give a few hypothetical examples of trip types: a customer may make a small daily dinner trip, a weekly large grocery trip, a trip to buy gifts for an upcoming holiday, or a seasonal trip to buy clothes. \newline Each visit may only have one TripType.

Data fields

TripType - a categorical id representing the type of shopping trip the customer made. This is the ground truth that you are predicting. TripType_999 is an "other" category. VisitNumber - an id corresponding to a single trip by a single customer Weekday - the weekday of the trip Upc - the UPC number of the product purchased ScanCount - the number of the given item that was purchased. A negative value indicates a product return. DepartmentDescription - a high-level description of the item's department FinelineNumber - a more refined category for each of the products, created by Walmart

In [2]:

```
path = 'C:\Users\shuyi\Documents\StudyResource\Kaggle\\'
train_file = "train.csv"
```

In [3]:

```
# Read the training data
train_data = pd.read_csv(path + train_file)

print("number of rows:", train_data.shape[0])
print("number of columns:", train_data.shape[1])
train_data[1:10]
```

```
('number of rows:', 647054)
('number of columns:', 7)
```

Out[3]:

	TripType	VisitNumber	Weekday	Upc	ScanCount	DepartmentDescripti
1	30	7	Friday	6.053882e+10	1	SHOES
2	30	7	Friday	7.410811e+09	1	PERSONAL CARE
3	26	8	Friday	2.238404e+09	2	PAINT AND ACCESSORIES
4	26	8	Friday	2.006614e+09	2	PAINT AND ACCESSORIES
5	26	8	Friday	2.006619e+09	2	PAINT AND ACCESSORIES
6	26	8	Friday	2.006614e+09	1	PAINT AND ACCESSORIES
7	26	8	Friday	7.004803e+09	1	PAINT AND ACCESSORIES
8	26	8	Friday	2.238495e+09	1	PAINT AND ACCESSORIES
9	26	8	Friday	2.238400e+09	-1	PAINT AND ACCESSORIES

In [4]:

```
# Inspect missing values in the dataset
if train_data.shape[0] - train_data.dropna().shape[0] == train_data.shape[0]:
    print("There is no missing value in the data set.")
else:
    print("Find the missing value and do data cleaning.")
```

```
Find the missing value and do data cleaning.
```

In [5]:

```
# Delete the rows with missing value
train_data = train_data.dropna(axis = 0)
print("After deleting the rows with missing value, the shape after filtering is:", trai
n_data.shape)
```

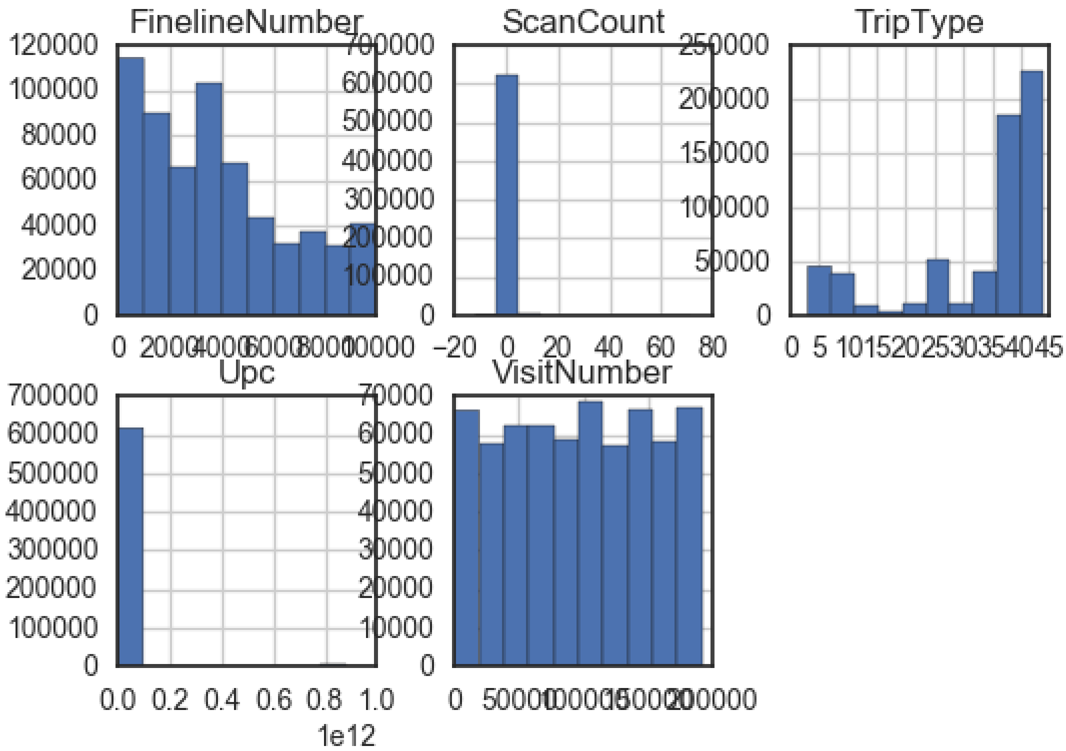
```
('After deleting the rows with missing value, the shape after filtering i
s:', (642925, 7))
```

In [45]:

```
train_data1.hist(layout=(2,3))
```

Out[45]:

```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000000003C78E58
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000000003C9EADD
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000042BBC6D
8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000000042CAF32
0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000042DEDC8
8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000000042E44B7
0>]], dtype=object)
```

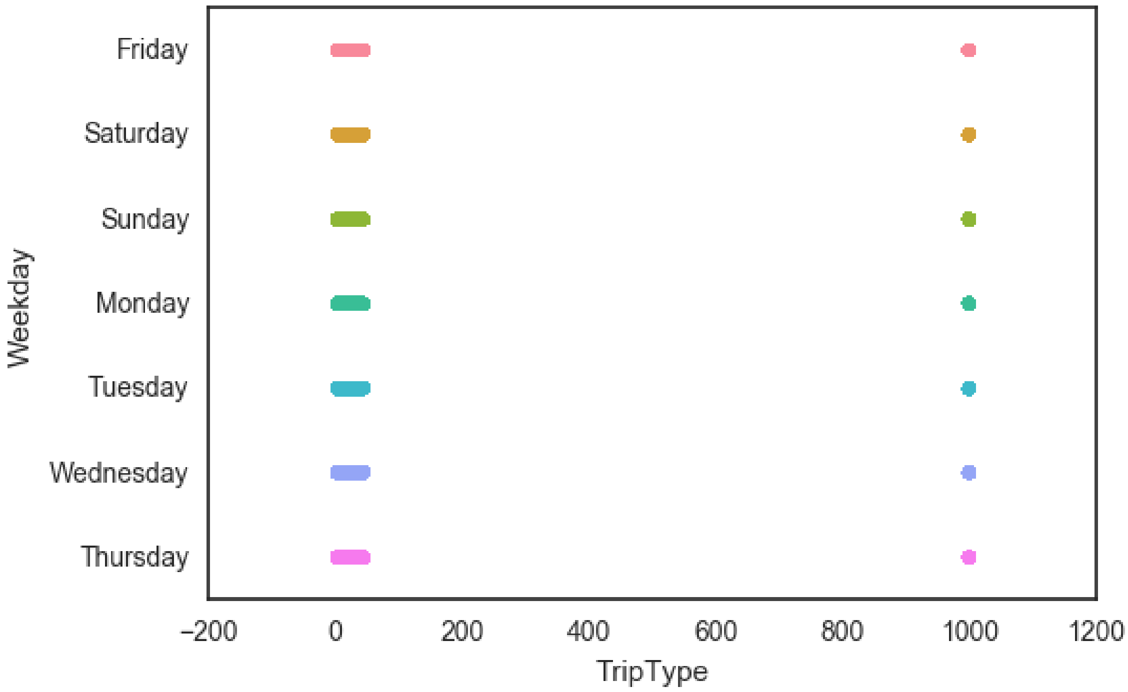


In [27]:

```
# Visualization of the Data
# visualizrion example reference: https://www.kaggle.com/benhamner/d/uciml/iris/python-data-visualizations
sns.stripplot(x="TripType", y="Weekday", data = train_data)
```

Out[27]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x26774898>
```



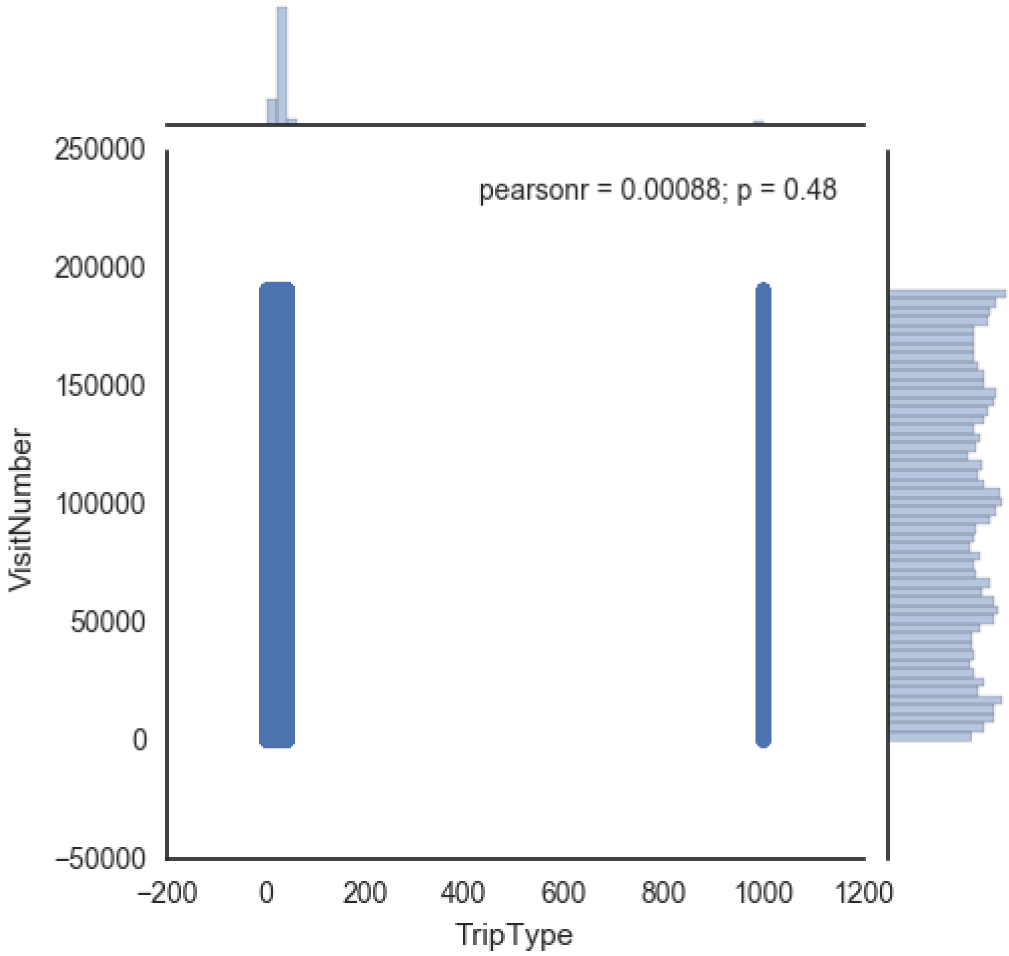
The Trip Type(TripType_999 is an "other" category) distributes in a similar way in different weekdays.

In [30]:

```
sns.jointplot(x="TripType", y="VisitNumber", data=train_data, size=5)
```

Out[30]:

<seaborn.axisgrid.JointGrid at 0x91306d8>



Most of the trip types concentrated at the regular range, a small portion of trip types = 999 exist. And the distribution of visitnumber to type 999 is similar.

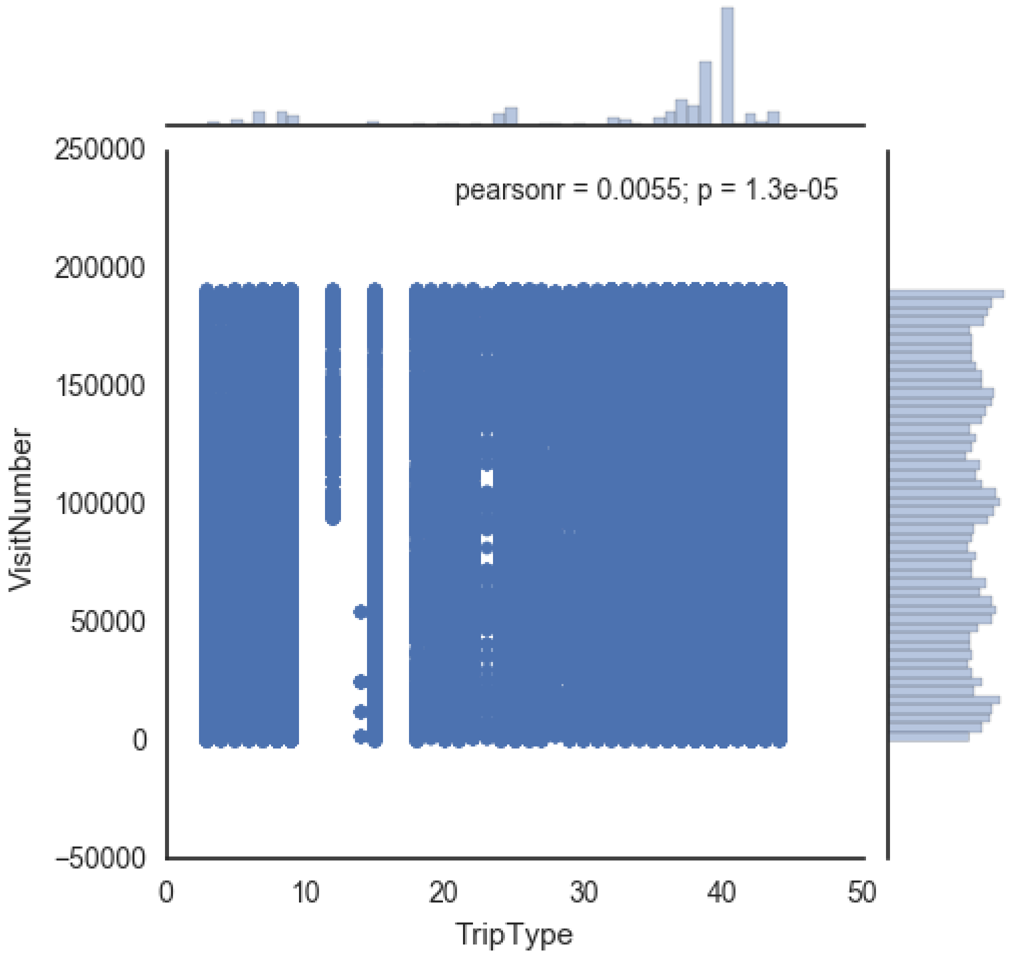
Let put the type 999 aside and investigate the distribution of other types first.

In [34]:

```
# filter the type999 in the dataset
train_data1 = train_data[train_data['TripType'] != 999]
sns.jointplot(x="TripType", y="VisitNumber", data=train_data1, size=5)
```

Out[34]:

```
<seaborn.axisgrid.JointGrid at 0x2abe0d68>
```

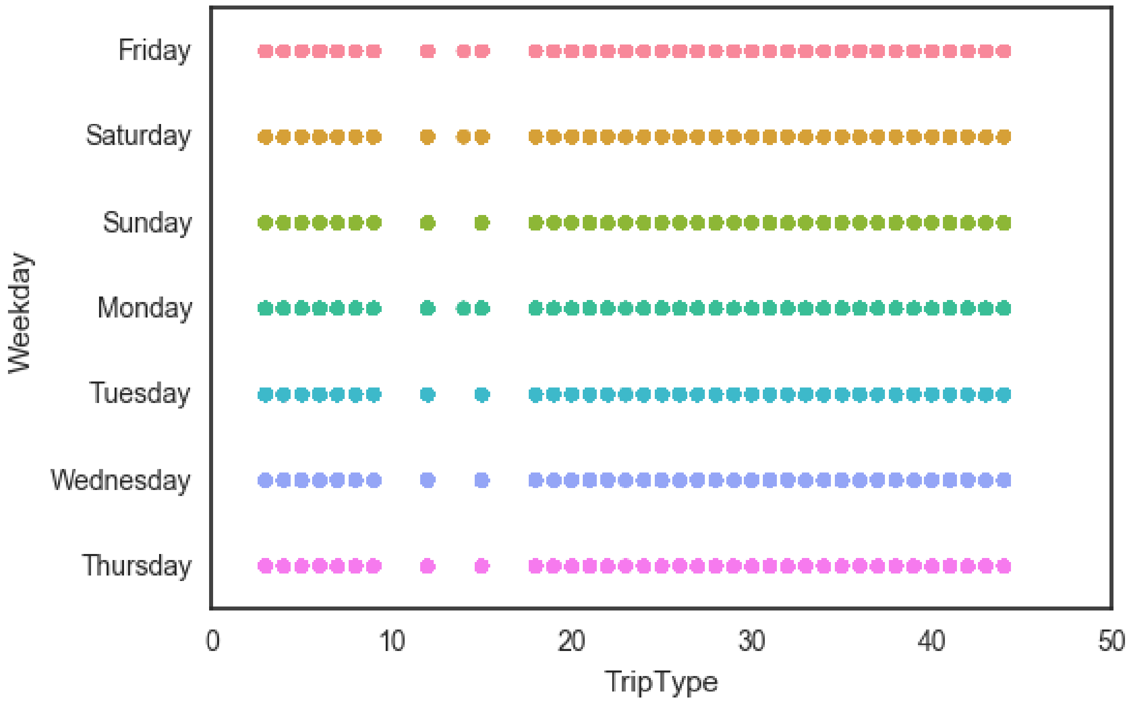


In [35]:

```
sns.stripplot(x="TripType", y="Weekday", data = train_data1)
```

Out[35]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2c8384e0>
```

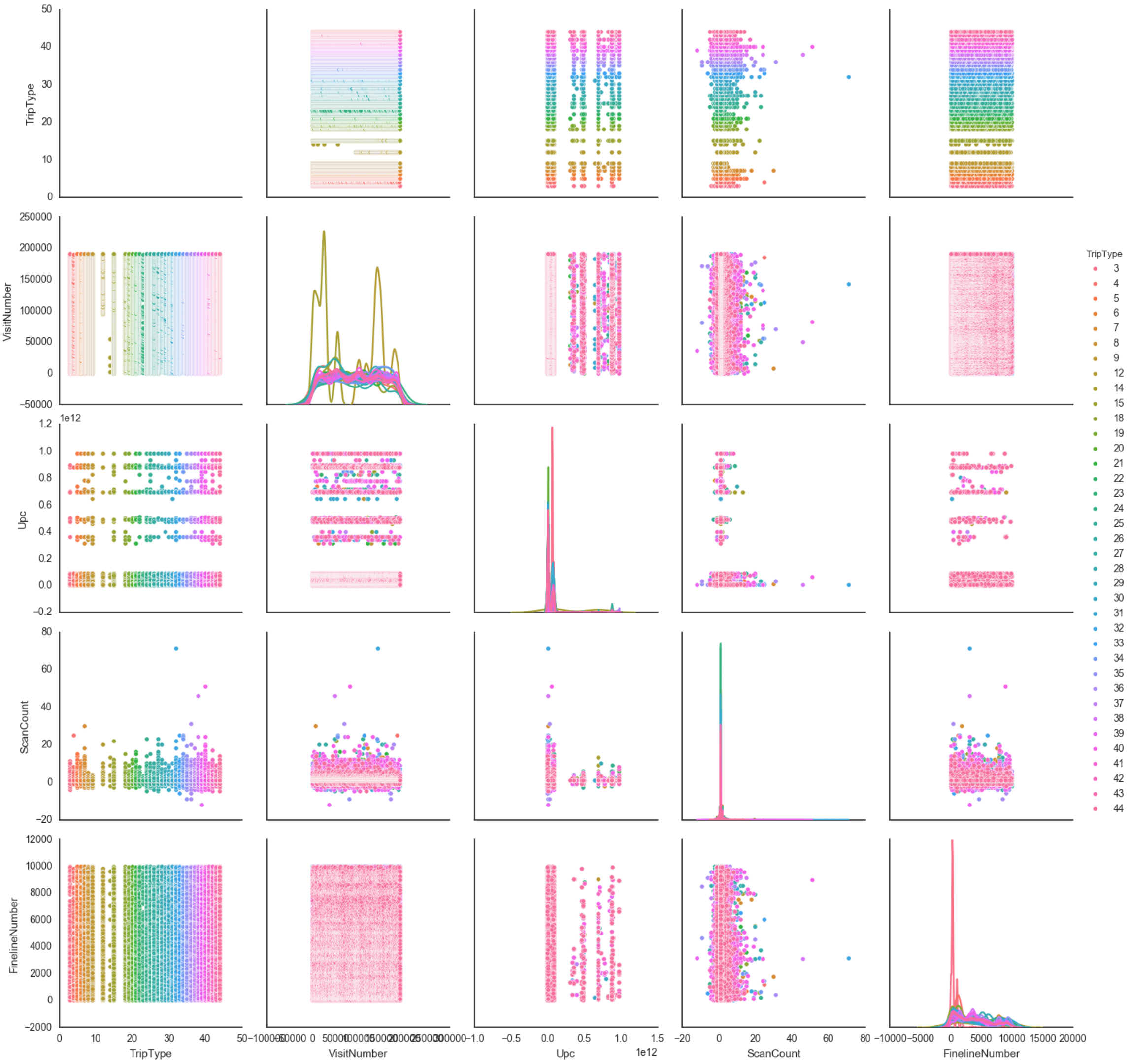


In [46]:

```
sns.pairplot(train_data1, hue="TripType", size=3, diag_kind="kde")
```

Out[46]:

<seaborn.axisgrid.PairGrid at 0x3c560080>



Stastistical Testing

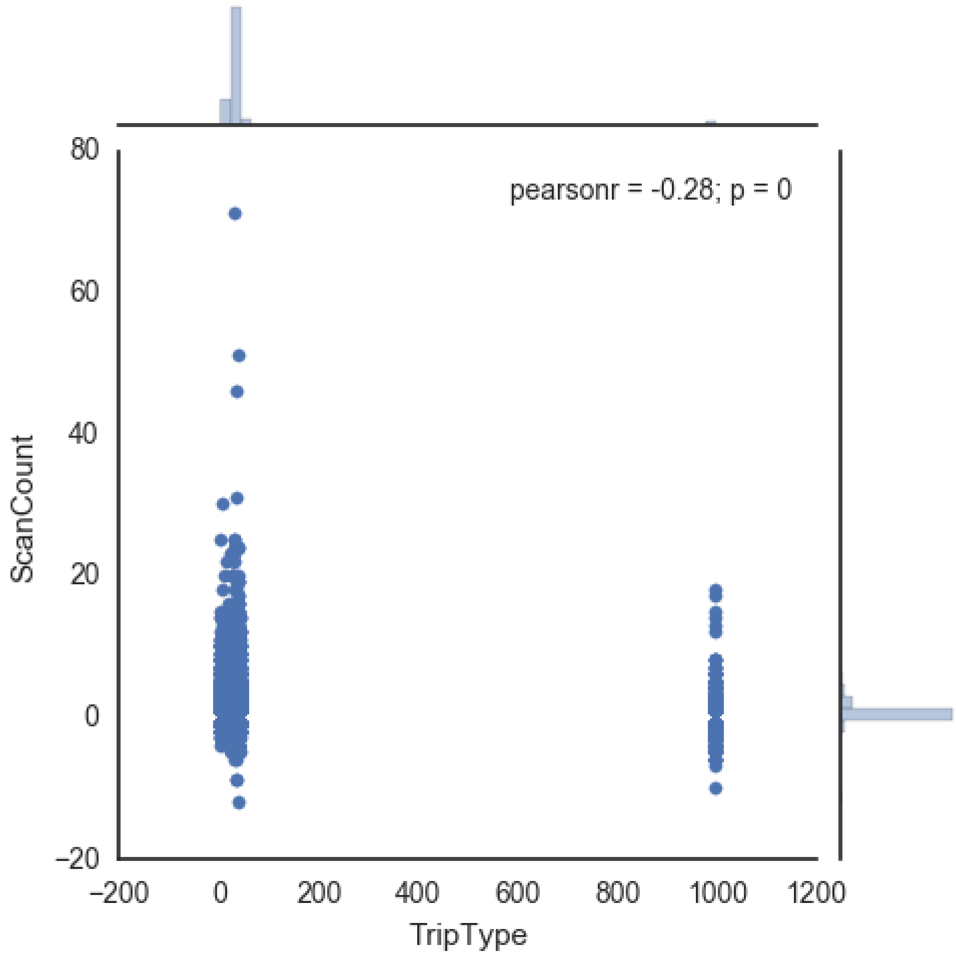
1. Outliers: we can see from the plots above that there are some outliers exist in the san count variable, therefore, we did investigation into it to detect them.

In [48]:

```
sns.jointplot(x="TripType", y="ScanCount", data=train_data, size=5)
```

Out[48]:

<seaborn.axisgrid.JointGrid at 0x491bb128>

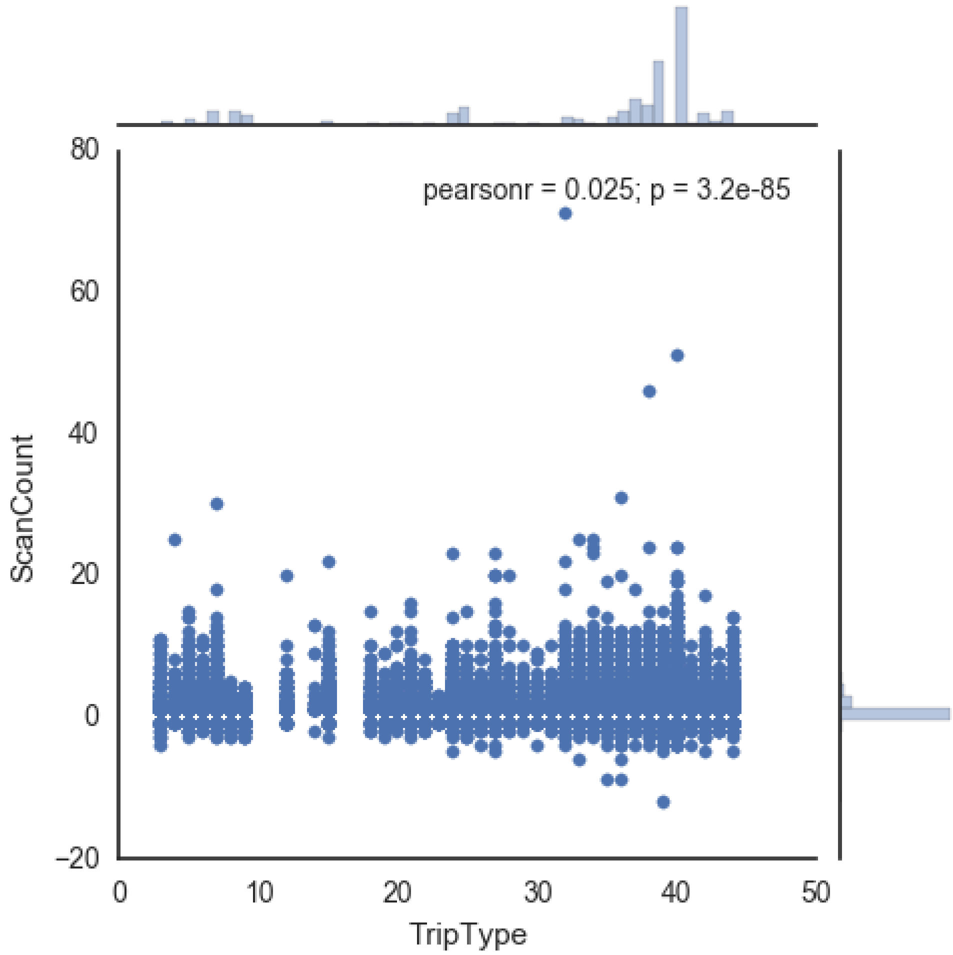


In [49]:

```
sns.jointplot(x="TripType", y="ScanCount", data=train_data1, size=5)
```

Out[49]:

<seaborn.axisgrid.JointGrid at 0x44e34c50>



A point that falls outside the data set's inner fences is classified as a minor outlier, while one that falls outside the outer fences is classified as a major outlier. To find the inner fences for your data set, first, multiply the interquartile range by 1.5. Then, add the result to Q3 and subtract it from Q1.

In [98]:

```
# group rows by triptype
grouped_data = train_data.sort('TripType')
```

In [116]:

```
trip_type = train_data.TripType.unique()
filtered = pd.DataFrame(columns = train_data.columns)
for t in trip_type:
    temp = grouped_data.loc[grouped_data['TripType'] == t]
    p1 = temp['ScanCount'].quantile(0.25)
    p3 = temp['ScanCount'].quantile(0.75)
    minimum = p1 - 1.5*(p3 - p1)
    maximum = p3 + 1.5*(p3 - p1)

    filtered_temp = temp.loc[(temp['ScanCount'] >= minimum) & (temp['ScanCount'] <= max
imum)]
    filtered = pd.concat([filtered, filtered_temp], axis = 0)

filtered = filtered.sort('TripType')
filtered
```

Out[116]:

...	TripType	VisitNumber	Weekday	Upc	ScanCount	DepartmentDes
386289	3.0	113742.0	Tuesday	6.811318e+10	1.0	FINANCIAL SEF
599912	3.0	179058.0	Saturday	6.053881e+10	1.0	FINANCIAL SEF
589812	3.0	176229.0	Friday	6.811318e+10	1.0	FINANCIAL SEF
127133	3.0	37950.0	Thursday	6.053880e+10	1.0	IMPULSE MER(
423739	3.0	125999.0	Thursday	6.811311e+10	1.0	FINANCIAL SEF
423738	3.0	125999.0	Thursday	6.053889e+10	1.0	FINANCIAL SEF
75147	3.0	21400.0	Monday	6.811316e+10	1.0	FINANCIAL SEF
75148	3.0	21400.0	Monday	6.811316e+10	1.0	FINANCIAL SEF
511299	3.0	151436.0	Monday	6.811316e+10	1.0	FINANCIAL SEF
99020	3.0	28701.0	Tuesday	6.811319e+10	1.0	FINANCIAL SEF
961	3.0	412.0	Friday	6.811311e+10	1.0	FINANCIAL SEF
364627	3.0	107505.0	Monday	6.053890e+10	1.0	FINANCIAL SEF
3862	3.0	1536.0	Friday	6.811316e+10	1.0	FINANCIAL SEF
3863	3.0	1536.0	Friday	6.053890e+10	1.0	FINANCIAL SEF
3864	3.0	1537.0	Friday	6.811316e+10	1.0	FINANCIAL SEF
137921	3.0	41578.0	Thursday	6.811316e+10	1.0	FINANCIAL SEF
3865	3.0	1537.0	Friday	6.811316e+10	1.0	FINANCIAL SEF
364626	3.0	107505.0	Monday	6.811316e+10	1.0	FINANCIAL SEF
960	3.0	412.0	Friday	6.053889e+10	1.0	FINANCIAL SEF
423455	3.0	125879.0	Thursday	8.303240e+10	1.0	IMPULSE MER(
603543	3.0	180078.0	Saturday	6.053881e+10	1.0	FINANCIAL SEF
603544	3.0	180078.0	Saturday	6.053881e+10	1.0	FINANCIAL SEF
98932	3.0	28665.0	Tuesday	6.053886e+10	1.0	FINANCIAL SEF
137920	3.0	41578.0	Thursday	6.053890e+10	1.0	FINANCIAL SEF
423454	3.0	125879.0	Thursday	6.053882e+10	1.0	FINANCIAL SEF
203895	3.0	60414.0	Sunday	6.053881e+10	1.0	FINANCIAL SEF
423688	3.0	125975.0	Thursday	6.811319e+10	1.0	FINANCIAL SEF
25320	3.0	7914.0	Saturday	8.303240e+10	1.0	IMPULSE MER(
423689	3.0	125975.0	Thursday	6.811316e+10	1.0	FINANCIAL SEF
70989	3.0	19875.0	Monday	6.053889e+10	1.0	FINANCIAL SEF
...	...	...	...	...	...	...
381502	999.0	112238.0	Monday	2.840044e+09	1.0	IMPULSE MER(
388055	999.0	114382.0	Tuesday	6.460077e+10	-1.0	BRAS & SHAPE'
380392	999.0	111972.0	Monday	8.191910e+10	1.0	BEAUTY

	TripType	VisitNumber	Weekday	Upc	ScanCount	DepartmentDes
380394	999.0	111972.0	Monday	7.780254e+09	1.0	BEAUTY
388277	999.0	114475.0	Tuesday	6.676461e+10	1.0	SLEEPWEAR/F(
388276	999.0	114475.0	Tuesday	6.676461e+10	1.0	SLEEPWEAR/F(
388275	999.0	114475.0	Tuesday	7.102972e+10	1.0	SLEEPWEAR/F(
388177	999.0	114449.0	Tuesday	3.600043e+09	-1.0	INFANT CONSU HARDLINES
388176	999.0	114449.0	Tuesday	3.600043e+09	1.0	INFANT CONSU HARDLINES
388168	999.0	114439.0	Tuesday	7.891581e+10	-1.0	SWIMWEAR/OU
388061	999.0	114394.0	Tuesday	6.811311e+10	-1.0	PERSONAL CAI
388060	999.0	114394.0	Tuesday	2.245700e+10	1.0	SERVICE DELI
618925	999.0	184308.0	Saturday	7.429942e+09	-1.0	IMPULSE MER(
384412	999.0	113074.0	Tuesday	7.355870e+09	-1.0	BEDDING
378467	999.0	111472.0	Monday	8.479120e+10	-1.0	WIRELESS
380393	999.0	111972.0	Monday	7.215146e+09	1.0	BEAUTY
380097	999.0	111903.0	Monday	8.329920e+10	1.0	PHARMACY OT
380693	999.0	112075.0	Monday	4.900001e+09	1.0	DSD GROCERY
380692	999.0	112075.0	Monday	7.874203e+09	-1.0	BAKERY
380691	999.0	112075.0	Monday	7.874203e+09	1.0	BAKERY
380690	999.0	112075.0	Monday	4.900001e+09	-1.0	DSD GROCERY
380611	999.0	112056.0	Monday	1.326150e+09	1.0	BOYS WEAR
380610	999.0	112056.0	Monday	3.700087e+09	1.0	HOUSEHOLD CHEMICALS/SU
380609	999.0	112056.0	Monday	3.700087e+09	-1.0	HOUSEHOLD CHEMICALS/SU
380608	999.0	112056.0	Monday	1.326150e+09	2.0	BOYS WEAR
380531	999.0	112026.0	Monday	3.187803e+09	-1.0	INFANT CONSU HARDLINES
380487	999.0	112005.0	Monday	4.741738e+09	1.0	ACCESSORIES
380449	999.0	111983.0	Monday	1.650056e+09	1.0	PHARMACY OT
380694	999.0	112075.0	Monday	8.066095e+09	-2.0	LIQUOR,WINE,I
167142	999.0	50671.0	Saturday	7.746300e+10	-1.0	OFFICE SUPPL

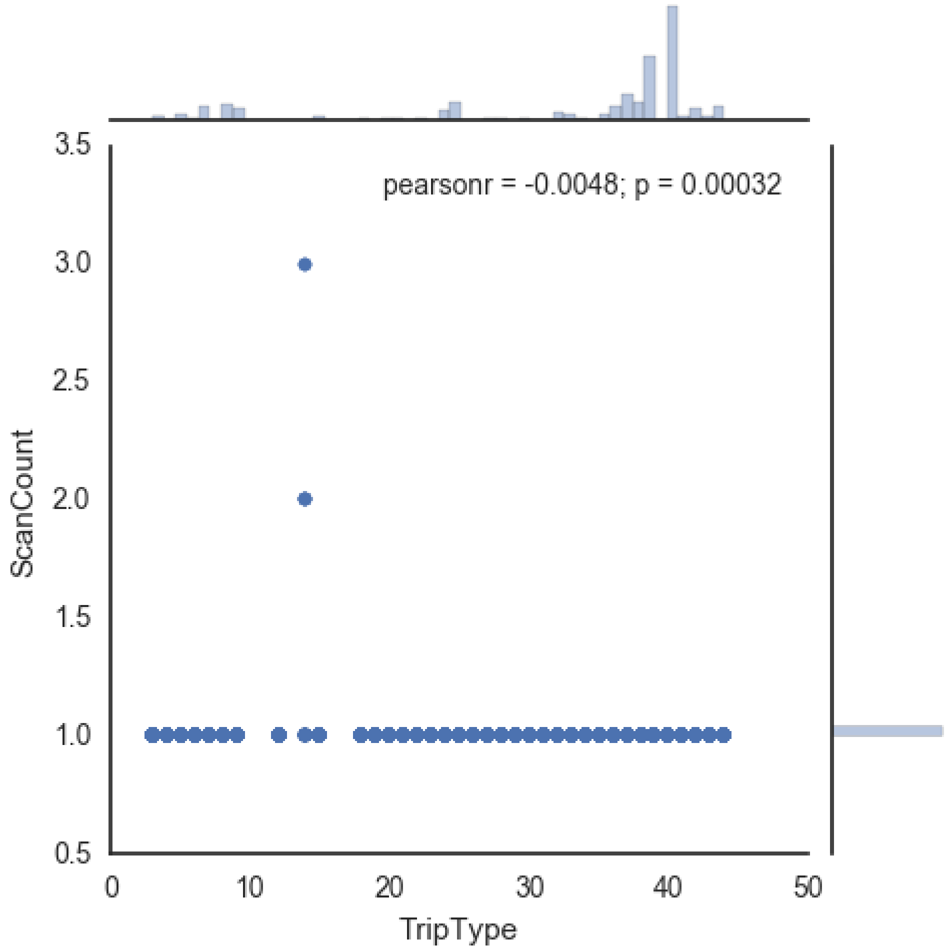
567984 rows × 7 columns

In [110]:

```
sns.jointplot(x="TripType", y="ScanCount", data = filtered[filtered['TripType'] < 999],
 size=5)
```

Out[110]:

<seaborn.axisgrid.JointGrid at 0x5742f9b0>



1. Cateogrical Variable: One-hot encoding

对于 Categorical Variable，常用的做法就是 One-hot encoding。即对这一变量创建一组新的伪变量，对应其所有可能的取值。这些变量中只有这条数据对应的取值为 1，其他都为 0。

In [6]:

```
# we still use train_data here( reserve filtering for later)
trip_type = train_data.TripType.unique()
print len(trip_type)
visit_num = train_data.VisitNumber.unique()
print(len(visit_num))
weekday = train_data.Weekday.unique()
print len(weekday)
upc = train_data.Upc.unique()
print len(upc)
department = train_data.DepartmentDescription.unique()
print len(department)
```

```
38
94247
7
97714
68
```

对 weekday 和 department 进行 encoding

In [12]:

```
weekday = pd.get_dummies(train_data['Weekday'])
department = pd.get_dummies(train_data['DepartmentDescription'])
```

In [ ]:

```
train_data.drop(['Weekday'], axis = 1, inplace = "True")
train_data.drop(['DepartmentDescription'], axis = 1, inplace = "True")
train_data = train_data.join(weekday)
train_data = train_data.join(department)
```

In [17]:

```
# save the data for the next step
train_data.to_csv(path + "step1.csv")
```

Feature Engineering first: generating more features based on all the features we have now

In [1]:

```
from sklearn import preprocessing
import pandas as pd
```

In [55]:

```
'''
data: the raw input data we have
'''
def Generate_Features(data):

    # encoding the department description label by sklearn
    le = preprocessing.LabelEncoder()
    # encode and transform the department description label
    data['DepartmentDescription'] =
le.fit_transform(list(data['DepartmentDescription']))
    data['Weekday'] = preprocessing.LabelEncoder().fit_transform(list(data['Weekday']))
     # assign a new column with scancount as the base value
    data['Count'] = data['ScanCount']
    data['Count'][data['ScanCount']<0] = 0  # filter the negative values
    data['FinelineNumber'].fillna(value = 10000, inplace = True) # replace the na value
s with 10000
    data['Upc'].fillna(value = -9999, inplace = True)# replace the na values with -9999
    #==================== Missing Value Indicators ====================================
===#
    # null value exist in Department Description, encoded as 67
    data1 = data[data['DepartmentDescription'] == 67]
    # the number of non na observations of each visit number
    data1 = data[data['DepartmentDescription']==67]
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'Count_Null'}, inplace=True)
    data = data.merge(data1, how='left', on=['VisitNumber'], copy=True)
    data['Count_Null'].fillna(value=0, inplace=True)
    data['Count_Null'][data['Count_Null']>0] = 1 # 把count 换成1

    data1 = data[data['ScanCount']<0]
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'ScanCount_Neg'}, inplace=True)
    data = data.merge(data1, how='left', on=['VisitNumber'], copy=True)
    data['ScanCount_Neg'].fillna(value=0, inplace=True)
    data['ScanCount_Neg'][data['ScanCount_Neg']>0] = 1

    data1 = data[data['FinelineNumber']==10000]
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'FinelineNumber_Missing'}, inplace=True)
    data = data.merge(data1, how='left', on=['VisitNumber'], copy=True)
    data['FinelineNumber_Missing'].fillna(value=0, inplace=True)
    data['FinelineNumber_Missing'][data['FinelineNumber_Missing']>0] = 1

    data1 = data.groupby(['VisitNumber',  'FinelineNumber'], as_index=False)['Count'].c
ount()
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'N_Fineline'}, inplace=True)
    data = data.merge(data1, how='left', on=['VisitNumber'], copy=True)
```

```
    data1 = data.groupby(['VisitNumber',  'Upc'], as_index=False)['Count'].count()
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'N_Upc'}, inplace=True)
    data = data.merge( data1, how='left', on=['VisitNumber'], copy=True)

    data1 = data.groupby(['VisitNumber', 'DepartmentDescription'], as_index=False)['Cou
nt'].count()
    data1 = data1.groupby(['VisitNumber'], as_index=False)['Count'].count()
    data1.rename(columns={'Count': 'N_Dep'}, inplace=True)
    data = data.merge( data1, how='left', on=['VisitNumber'], copy=True)

    # group data for new features:
    # 1. visit number and departmant description
    # the scan counts for each visitnumber and department combination
    temp1 = data.groupby(['VisitNumber', 'DepartmentDescription'], as_index=False)['Sca
nCount'].sum()
    temp11 = temp1.groupby(['VisitNumber'], as_index=False)['ScanCount'].min()
    temp12 = temp1.groupby(['VisitNumber'], as_index=False)['ScanCount'].max()
    temp13 = temp1.groupby(['VisitNumber'], as_index=False)['ScanCount'].mean()
    temp11.rename(columns={'ScanCount': 'Min_Count'}, inplace=True)
    temp12.rename(columns={'ScanCount': 'Max_Count'}, inplace=True)
    temp13.rename(columns={'ScanCount': 'Mean_Count'}, inplace=True)
    # left join to the dataset
    data = data.merge(temp11, how='left', on=['VisitNumber'], copy=True)
    data = data.merge(temp12, how='left', on=['VisitNumber'], copy=True)
    data = data.merge(temp13, how='left', on=['VisitNumber'], copy=True)

    # 2. UPC: A UPC should have 12 digits. The first 6 digits are company code. The nex
t four are item code.
    # add check sum to the end of every upc and missing zeros at the begining of the up
c
    # convert Upc to string first
    data['Upc'] = data['Upc']*10
    data['Upc'] = data.Upc.apply(string_convert)
    data['Upc_full'] = data.Upc.apply(upc_fullfill)
    data['company'] =  data.Upc_full.apply(company_extractor)
    return data

def string_convert(x):
    return ('%.2f' % (x,)).rstrip('0').rstrip('.')

def upc_checksum_calculator(x):
    try:
        odd = map(int, ','.join(x[-1::-2]).split(','))
        even = map(int, ','.join(x[-2::-2]).split(','))
        sum_odd = sum(odd) * 3
        total = sum_odd + sum(even)
        rest = total % 10
        if rest == 0:
            return rest
        return 10 - rest
    except:
        return -9999 # return na for upc which can not be decoded

def upc_fullfill(x):
    try:
        if len(x) < 12:
            missing_zeros = 11 - len(x)
            zeros = ['0'] * missing_zeros
            full_upc = zeros + ','.join(x).split(',') +
[str(upc_checksum_calculator(x))]
```

```
            full_upc = ''.join(full_upc)
            return full_upc
        else:
            return x
    except:
        return -9999

def company_extractor(x):
    try:
        p = x[:6]
        if p == '000000':
            return x[-5]
        return p
    except:
        return -9999
```

In [56]:

```
path = 'C:\Users\shuyi\Documents\StudyResource\Kaggle\\'
train_file = "train.csv"
train_data = pd.read_csv(path + train_file)
```

In [ ]:

```
train_data_step2 = Generate_Features(train_data)
```

In [58]:

```
train_data_step2[10:20]
```

Out[58]:

	TripType	VisitNumber	Weekday	Upc	ScanCount	DepartmentDescrip
10	26	8	0	52000102390	1	17
11	26	8	0	886793005010	2	49
12	26	8	0	220060000000	1	41
13	26	8	0	22367604520	1	49
14	26	8	0	886793005010	-1	49
15	26	8	0	22384002000	2	49
16	26	8	0	30192942030	1	49
17	26	8	0	724504088400	1	49
18	26	8	0	255415000000	2	16
19	26	8	0	23100107760	1	51

In [59]:

```
# save the data
train_data_step2.to_csv(path + "step2.csv")
```

In [1]:

```
'''
This file is for Exploratory Data Analysis
@ Author: Shuyi Wang
@ Date: 2017/3/18
'''
import pandas as pd
import warnings # current version of seaborn generates a bunch of warnings that we'll i
gnore
warnings.filterwarnings("ignore")
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestClassifier
import numpy as np
sns.set(style="white", color_codes=True)

%matplotlib inline
```

In [2]:

```
path = 'C:\Users\shuyi\Documents\StudyResource\Kaggle\\'
train_file = "step2.csv"
# Read the training data
train_data = pd.read_csv(path + train_file)

print("number of rows:", train_data.shape[0])
print("number of columns:", train_data.shape[1])
train_data[1:10]
```

```
('number of rows:', 647054)
('number of columns:', 20)
```

Out[2]:

	Unnamed: 0	TripType	VisitNumber	Weekday	Upc	ScanCount	Departm
1	1	30	7	0	605388159800	1	62
2	2	30	7	0	74108110990	1	50
3	3	26	8	0	22384035100	2	49
4	4	26	8	0	20066137440	2	49
5	5	26	8	0	20066187830	2	49
6	6	26	8	0	20066137430	1	49
7	7	26	8	0	70048027370	1	49
8	8	26	8	0	22384953180	1	49
9	9	26	8	0	22384002000	-1	49

1.Feature Selection: Feature Selection 最实用的方法也就是看 Random Forest 训练完以后得到的 Feature Importance 了。

In [3]:

```
# initial random forest tree for variable selection
train_data.drop('Unnamed: 0', 1, inplace = True)
clf = RandomForestClassifier(n_jobs=2)
features = [ f for f in train_data.columns if f != 'TripType' ]
y = train_data['TripType']
# y, _ = pd.factorize(train_data['TripType'])
clf.fit(train_data[features], y)
```

Out[3]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_split=1e-07, min_samples_leaf=1,
            min_samples_split=2, min_weight_fraction_leaf=0.0,
            n_estimators=10, n_jobs=2, oob_score=False, random_state=None,
            verbose=0, warm_start=False)
```

In [4]:

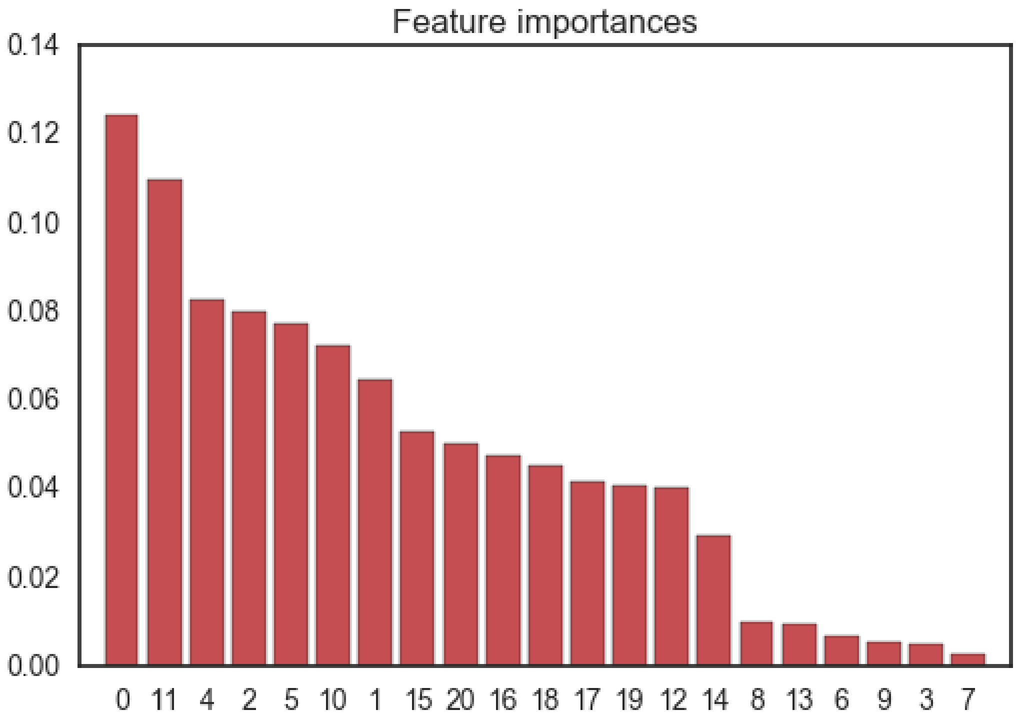
```
# display the importance of each feature
importances = clf.feature_importances_
std = np.std([clf.feature_importances_ for tree in clf.estimators_],
             axis=0)
indices = np.argsort(importances)[::-1]

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

for f in range(train_data.shape[1] - 1): # exclusing the y column
    print("%d. feature %d (%f)" % (f + 1, indices[f], importances[indices[f]]))

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

```
Feature ranking:
1. feature 0 (0.124356)
2. feature 11 (0.109684)
3. feature 4 (0.082663)
4. feature 2 (0.079965)
5. feature 5 (0.077523)
6. feature 10 (0.072318)
7. feature 1 (0.064707)
8. feature 15 (0.053032)
9. feature 20 (0.050440)
10. feature 16 (0.047699)
11. feature 18 (0.045063)
12. feature 17 (0.041520)
13. feature 19 (0.040616)
14. feature 12 (0.040527)
15. feature 14 (0.029509)
16. feature 8 (0.010230)
17. feature 13 (0.009679)
18. feature 6 (0.007035)
19. feature 9 (0.005565)
20. feature 3 (0.005050)
21. feature 7 (0.002820)
```



From the feature importance ranking we can see that the following features contribute about 95% importance: Feature ranking:

1. feature 0 (0.124356)
2. feature 11 (0.109684)
3. feature 4 (0.082663)
4. feature 2 (0.079965)
5. feature 5 (0.077523)
6. feature 10 (0.072318)
7. feature 1 (0.064707)
8. feature 15 (0.053032)
9. feature 20 (0.050440)
10. feature 16 (0.047699)
11. feature 18 (0.045063)
12. feature 17 (0.041520)
13. feature 19 (0.040616)
14. feature 12 (0.040527)
15. feature 14 (0.029509)
16. feature 8 (0.010230)
17. feature 13 (0.009679)
18. feature 6 (0.007035)
19. feature 9 (0.005565)
20. feature 3 (0.005050)
21. feature 7 (0.002820)

In [5]:

```
selected_features = []
for i in [0, 11, 4, 2, 5, 10, 1, 15, 20, 16, 18, 17, 19, 12, 14, 8, 13, 6, 9, 3, 7]:
    print "Feature index: ", i, "Feature Name: ", features[i]
    selected_features.append(features[i])
# select the important features and reconstruct the dataframe

data_selected = train_data[['TripType'] + selected_features]
data_selected[1:10]
```

```
Feature index:  0 Feature Name:  VisitNumber
Feature index:  11 Feature Name:  N_Upc
Feature index:  4 Feature Name:  DepartmentDescription
Feature index:  2 Feature Name:  Upc
Feature index:  5 Feature Name:  FinelineNumber
Feature index:  10 Feature Name:  N_Fineline
Feature index:  1 Feature Name:  Weekday
Feature index:  15 Feature Name:  Mean_Count
Feature index:  20 Feature Name:  max_to_mean
Feature index:  16 Feature Name:  Range
Feature index:  18 Feature Name:  Ratio_U_D
Feature index:  17 Feature Name:  Ratio_F_D
Feature index:  19 Feature Name:  mean_to_min
Feature index:  12 Feature Name:  N_Dep
Feature index:  14 Feature Name:  Max_Count
Feature index:  8 Feature Name:  ScanCount_Neg
Feature index:  13 Feature Name:  Min_Count
Feature index:  6 Feature Name:  Count
Feature index:  9 Feature Name:  FinelineNumber_Missing
Feature index:  3 Feature Name:  ScanCount
Feature index:  7 Feature Name:  Count_Null
```

Out[5]:

	TripType	VisitNumber	N_Upc	DepartmentDescription	Upc	FinelineNu
1	30	7	2	62	6.053882e+10	8931.0
2	30	7	2	50	7.410811e+09	4504.0
3	26	8	21	49	2.238404e+09	3565.0
4	26	8	21	49	2.006614e+09	1017.0
5	26	8	21	49	2.006619e+09	1017.0
6	26	8	21	49	2.006614e+09	1017.0
7	26	8	21	49	7.004803e+09	2802.0
8	26	8	21	49	2.238495e+09	4501.0
9	26	8	21	49	2.238400e+09	3565.0

9 rows × 22 columns

1. Model Selection

In [5]:

```
# xgboost
import xgboost as xgb
import numpy as np
#from sklearn.model_selection import KFold
import sklearn
```

In [ ]:

```
train = data_selected.ix[:, data_selected.columns != 'TripType']
target = data_selected['TripType']

# label need to be 0 to num_class -1, so relabel all the target values to 1...class - 1
target_reindexed = np.arange(0,len(set(target)))
target_indexmap = {}
trip = list(set(target))
for i in target_reindexed:
    target_indexmap[trip[i]] = i
# reindex the target to the new index
target_new = []
for row in target:
    target_new.append(target_indexmap[row])
target_new = pd.DataFrame(target_new)
```

In [8]:

```
# 调参
# setup parameters for xgboost
param = {}
# use softmax multi-class classification
param['objective'] = 'multi:softmax'
# scale weight of positive examples
param['eta'] = 0.1
param['max_depth'] = 6
param['silent'] = 1
param['nthread'] = 4
param['num_class'] = len(set(target))
num_round = 5
# evallist  = [(dtest,'eval'), (dtrain,'train')]
```

In [ ]:

```
# cross validation, 5 folder to have the test data set label
kf = sklearn.model_selection.KFold(5)
accuracy = 0
n = 1
for train_fold, test_fold in kf.split(train):
    #print type(train_fold), numpy array
    X_train, X_test, y_train, y_test = train.loc[train_fold], train.loc[test_fold], tar
get_new.loc[train_fold], target_new.loc[test_fold]

    xgtrain = xgb.DMatrix(X_train.values, y_train.values)
    xgtest = xgb.DMatrix(X_test.values, y_test.values)
    watchlist = [ (xgtrain,'train'), (xgtest, 'test') ]
    bst = xgb.train( param, xgtrain, num_round, watchlist)
    # get prediction
    pred = bst.predict( xgtest );
    bst.save_model( 'xgboost' + str(n) + '.model')
    accuracy += sum( int(pred[i]) != y_test.loc[i].values[0] for i in
range(len(y_test))) / float(len(y_test))
    print ('predicting, classification error=%f' % (sum( int(pred[i]) !=
y_test.loc[i].values[0] for i in range(len(y_test))) / float(len(y_test)) ))
    n += 1
print ("The accuracy of prediction for xgboost is ", accuracy)
```

In [34]:

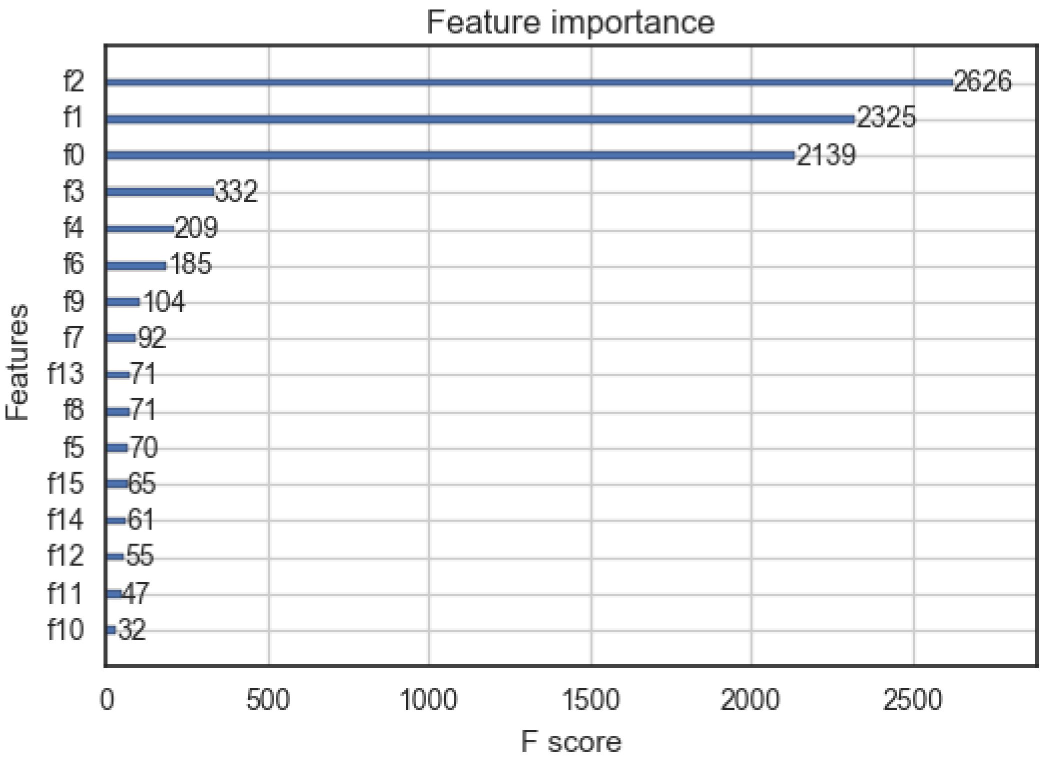
```
# Load the results from xgboost

bst = xgb.Booster({'nthread':4}) #init model
bst.load_model("xgboost1.model") # Load data

# feature importance from this model
xgb.plot_importance(bst)
```

Out[34]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x22a89080>
```



In [37]:

```
xgboost_features = [2,1,0,3,4,6,9,7,13,8,5,15,14,12,11,10]
xgboost_selectedFeatures = []
for i in xgboost_features:
    print "Feature name: ", selected_features[i]
    xgboost_selectedFeatures.append(selected_features[i])
```

```
 Feature name:  FinelineNumber
Feature name:  Upc
Feature name:  VisitNumber
Feature name:  ScanCount
Feature name:  GROCERY DRY GOODS
Feature name:  PRODUCE
Feature name:  Saturday
Feature name:  Sunday
Feature name:  Thursday
Feature name:  MENS WEAR
Feature name:  FINANCIAL SERVICES
Feature name:  Wednesday
Feature name:  Tuesday
Feature name:  Monday
Feature name:  Friday
Feature name:  INFANT CONSUMABLE HARDLINES
```

deep learning neural network model

In [ ]:

```
'''
import os
print(os.path.expanduser('~'))
To find the keras.jason file and change the backened option to theano
'''
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.utils import np_utils
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.pipeline import Pipeline
```

This is important to ensure that the results we achieve from this model can be achieved again precisely. It ensures that the stochastic process of training a neural network model can be reproduced.

In [8]:

```
# fix random seed for reproducibility
seed = 7
np.random.seed(seed)
```

number of layer equal to number of categorical output in y

In [9]:

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD
```

In [ ]:

```
model = Sequential()
# Dense(64) is a fully-connected layer with 64 hidden units.
# in the first layer, you must specify the expected input data shape:
# here, 20-dimensional vectors.
model.add(Dense(64, activation='relu', input_dim=20))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd,
              metrics=['accuracy'])

model.fit(x_train, y_train,
          epochs=20,
          batch_size=128)
score = model.evaluate(x_test, y_test, batch_size=128)
```

Logistic regression

In [15]:

```
import matplotlib.pyplot as plt
import sklearn
from sklearn.linear_model import LogisticRegression
```

Stack all the models we selected and build the pipline

In [2]:

```
import numpy as np
np.random.seed(1234) # set seed
import pandas as pd
from scipy import sparse
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier, NearestNeighbors
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, ExtraTreesClas
sifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
```

In [3]:

```
# 2rd layer: ensemble the XGboost and CNN, use data from stacking

bagging = True
bagging_size = 50 # number of bagging size, stablizing the predictions

n_folds = 5 # folds for cross validation

# load data, load log data from previous steps
path = 'C:\Users\shuyi\Documents\StudyResource\Kaggle\\'
train_file = "step2.csv"
# Read the training data
train_data = pd.read_csv(path + train_file)

print("number of rows:", train_data.shape[0])
print("number of columns:", train_data.shape[1])
train_data = train_data[train_data.Upc != -99990]
train_data[1:10]
```

```
('number of rows:', 647054)
('number of columns:', 20)
```

Out[3]:

	Unnamed: 0	TripType	VisitNumber	Weekday	Upc	ScanCount	Departm
1	1	30	7	0	605388159800	1	62
2	2	30	7	0	74108110990	1	50
3	3	26	8	0	22384035100	2	49
4	4	26	8	0	20066137440	2	49
5	5	26	8	0	20066187830	2	49
6	6	26	8	0	20066137430	1	49
7	7	26	8	0	70048027370	1	49
8	8	26	8	0	22384953180	1	49
9	9	26	8	0	22384002000	-1	49

In [ ]:

```
# stack algorithms for multiple models
'''
use n different classifiers to obtain out of fold prefictions for target data.
It uses the train data to get the predictions for test
Adds n features to both train and test data
both input data are in pandas dataframe format
'''
def StackModels(train, test, y, models, n_folds):
    num_class = np.unique(y).shape[0]
    # The folds are made by preserving the percentage of samples for each class.
    y_folds = list(StratifiedKFold(y, n_folds))

    train_sc = train
    test_sc = test

    # number of rows * number of classifiers
    blend_train = np.zeros((train.shape[0], num_class*len(models)))
    blend_test = np.zeros((test.shape[0], num_class*len(models)))
    for j, model in enumerate(models):
        print("Training the model [%s]" %(i))

        for i, (train_i, cv_i) in enumerate(y_folds):
            print("Now training the fold [%s]" %(j))

            #train on 2 folds, predict the 3rd fold
            x_train = train[train_i]# select this fold by index from cross validation
            y_train = y[train_i]
            x_cv = train[cv_i] #针对这个fold，所形成的余下data组合成的cross validation

            model.fit(x_train, y_train)
            prediction = model.predict_proba(x_cv)
            blend_train[cv_i, j*num_class:(j+1)*num_class] = prediction #the jth mode
l's prediction on each cross validation of each fold

        print("Stacking test data")
        model.fit(train, y)
        prediction = model.predict_prob(test)
        blend_test[:, j*num_class:(j+1)*num_class] # columns belong to different models

    return blend_train, blend_test
```