In [1]:
```
import pandas as pd
import pyodbc
import numpy as np
from datetime import datetime, timedelta
from sklearn.metrics import mean_squared_error
from math import sqrt
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn import tree as dt
from sklearn import linear_model
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import RandomForestClassifier
%matplotlib inline
from flask import Flask, jsonify,request
```

In [2]:
```
%config IPCompleter.greedy=True
```

In [3]:
```
app = Flask(__name__)

@app.route('/')
def index():
    return "Hello, World!"
```

In [4]:
```
@app.route('/api/PredictShelfLife', methods=['POST'])
def PredShelfLife():

    ClassName= request.json.get("ClassName","")
    CategoryName= request.json.get("CategoryName","")
    GroupName=request.json.get("GroupName","")
    ShopCode=request.json.get("ShopCode","")
    Season=request.json.get("Season","")
    SalePrice=request.json.get("SalePrice","")
    ShelfLife=itemShelfLifePrediction(ClassName, CategoryName, GroupName, ShopCod


    return jsonify({'ShelfLife': ShelfLife}), 201
```

In [5]:
```
@app.route('/api/PredictPrice', methods=['POST'])
def PredPrice():

    ClassName= request.json.get("ClassName","")
    CategoryName= request.json.get("CategoryName","")
    SubClassName= request.json.get("SubClassName","")
    ColorName=request.json.get("ColorName","")
    GroupName=request.json.get("GroupName","")
    TypeName=request.json.get("TypeName","")
    Season=request.json.get("Season","")

    Price=itemPricePrediction(ClassName, CategoryName, SubClassName,ColorName, Gr


    return jsonify({'Price': Price}), 201
```

In [6]:
```
# query = "select items.ItemCode, DesignCode,category.CategoryName,class.ClassNam
query = "select * from shopWiseShelfLife"
queryDFS = "SELECT distinct [ItemCode],[VoucherDate] FROM [AliFashion].[dbo].[Rec
```

In [7]:
```
sql_conn = pyodbc.connect('DRIVER={SQL Server};SERVER=DESKTOP-2BUG72N;DATABASE=Al
df = pd.read_sql(query, sql_conn)
dfs = pd.read_sql(queryDFS, sql_conn)
```

In [8]:
```
pf = dfs
pf['month'] = pd.DatetimeIndex(pf['VoucherDate']).month
pf["Season"] = np.where(pf["month"]>3, '1','0')
pf["Season2"] = np.where(pf["month"]<10, '1','0')
pf["SaleSeason"] = np.where(pf["Season"] == pf["Season2"], 'Summer','Winter')

pf = pf.drop("month",axis=1)
pf = pf.drop("Season",axis=1)
pf = pf.drop("Season2",axis=1)

pf['freq'] = pf.groupby('ItemCode')['ItemCode'].transform('count')
```

In [9]:
```
newDF = pd.DataFrame(columns=["ItemCode", "Season"])

for item in pf.ItemCode.unique():
    indexes = pf.ItemCode == item
    temp = pf[indexes]
    flags = temp.SaleSeason.str.contains('Winter', case=True, regex=True).value_c
    if flags.size < 2:
        newDF = newDF.append({"ItemCode":item, "Season":flags.keys()[0]},ignore_i
    else:
        if flags[0] > flags[1]:
            newDF = newDF.append({"ItemCode":item, "Season":flags.keys()[0]},igno
        else:
            newDF = newDF.append({"ItemCode":item, "Season":flags.keys()[1]},igno

newDF.Season = newDF.Season.astype(str)
newDF.ItemCode = newDF.ItemCode.astype('int64')

newDF.Season[newDF.Season == 'True'] = "Winter"
newDF.Season[newDF.Season == 'False'] = "Summer"
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-pac
kages\ipykernel_launcher.py:18: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\site-pac
kages\ipykernel_launcher.py:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stab
le/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pandas-doc
s/stable/indexing.html#indexing-view-versus-copy)
```

In [10]:
```
df = df.dropna()
train = df
trainData = df


train["Season"] = pd.Series()
for item, season in zip(newDF.ItemCode, newDF.Season):
        mask = train.ItemCode == item
        column_name = 'Season'
        train.loc[mask, column_name] = season
```

In [11]:
```
# train = train.dropna()
# trainData = trainData.dropna()

train['ClassNameEncoded'] = train['ClassName']
train['SubClassNameEncoded'] = train['SubClassName']
train['ItemNameEncoded'] = train['ItemName']
train['TypeNameEncoded'] = train['TypeName']
train['CategoryNameEncoded'] = train['CategoryName']
train['CategoryNameEncoded'] = train['CategoryName']
train['ColorNameEncoded'] = train['ColorName']
train['GroupNameEncoded'] = train['GroupName']
train['SeasonEncoded'] = train['Season']

counter = 0
for className in train.ClassName.unique():
    train['ClassNameEncoded'] = train.apply(lambda x: counter if x['ClassNameEnco
    counter = counter + 1

counter = 0
for className in train.SubClassName.unique():
    train['SubClassNameEncoded'] = train.apply(lambda x: counter if x['SubClassNa
    counter = counter + 1

counter = 0
for item in train.ItemName.unique():
    train['ItemNameEncoded'] = train.apply(lambda x: counter if x['ItemNameEncode
    counter = counter + 1

counter = 0
for typeN in train.TypeName.unique():
    train['TypeNameEncoded'] = train.apply(lambda x: counter if x['TypeNameEncode
    counter = counter + 1

counter = 0
for category in train.CategoryName.unique():
    train['CategoryNameEncoded'] = train.apply(lambda x: counter if x['CategoryNa
    counter = counter + 1

counter = 0
for color in train.ColorName.unique():
    train['ColorNameEncoded'] = train.apply(lambda x: counter if x['ColorNameEnco
    counter = counter + 1

counter = 0
for group in train.GroupName.unique():
    train['GroupNameEncoded'] = train.apply(lambda x: counter if x['GroupNameEnco
    counter = counter + 1

counter = 0
for group in train.Season.unique():
    train['SeasonEncoded'] = train.apply(lambda x: counter if x['SeasonEncoded'] 
    counter = counter + 1
```

In [12]:
```
train
```

Out[12]:

	ShopCode	shlefLife	ItemCode	DesignCode	CategoryName	ClassName	SubClassName	I
0	3.0	0.087324	1	X3296	Suit 03 Piece	Bombo Slub	Bombo Slub Shirt + Bombo Dupatta	
1	3.0	0.200000	2	X2595	Suit 03 Piece	Swiss Lawn	Swiss Lawn Shirt + Chiffon Dupatta	
3	3.0	0.055556	4	X2574	Suit 03 Piece	Swiss Lawn	Swiss Lawn Shirt + Chiffon Dupatta	
							Swiss Lawn	

In [13]:
```
def correlation( attribute_1, attribute_2 ):
    """
    Implement the function correlation that takes as input 2 numeric variables an
    Use Pearson Correlation - The one which you studied in class
    Do not use built in libraries
    """
    meanA = np.mean(attribute_1)
    meanB = np.mean(attribute_2)

    stdA = np.std(attribute_1)
    stdB = np.std(attribute_2)

    const = len(train) * meanA * meanB
    sum = 0
    for a,b in zip(attribute_1, attribute_2):
        sum = sum + ((a*b) - const)

    divisor = ((len(train) -1)* stdA * stdB)
    corr = 0
    if divisor != 0:
        corr = sum / divisor

    return corr
```

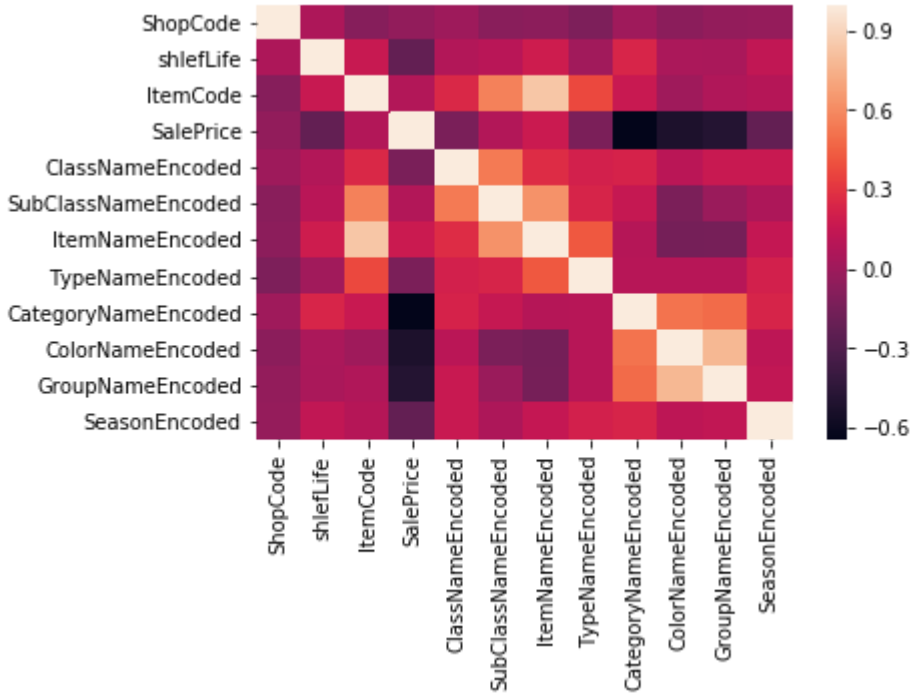
In [14]:
```
train.describe()
```

Out[14]:

	ShopCode	shlefLife	ItemCode	SalePrice	ClassNameEncoded	SubClassNameEn
count	2315.000000	2315.000000	2315.000000	2315.000000	2315.000000	2315.0
mean	2.498920	0.247573	1224.772354	1855.528294	15.278186	51.7
std	0.513747	0.690210	691.663976	1166.015140	12.904557	37.8
min	2.000000	0.000000	1.000000	16.000000	0.000000	0.0
25%	2.000000	0.029412	611.500000	1040.000000	6.000000	20.0
50%	2.000000	0.085106	1228.000000	1800.000000	10.000000	52.0
75%	3.000000	0.213039	1860.500000	2632.000000	21.000000	65.0
max	4.000000	11.900000	2397.000000	10080.000000	56.000000	160.0

In [15]:
```
corr = train.corr()
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

Out[15]: <matplotlib.axes._subplots.AxesSubplot at 0x189e12f2f28>



In [16]:
```
colNames = train[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClassNameEncoded
corrDict = {}
for col in colNames.columns:
    corrDict[col] = (correlation(train[col], train.SalePrice))

corrDict = (sorted(corrDict.items(), key=lambda kv: kv[1], reverse= 1))
corrDict
```

Out[16]:
```
[('GroupNameEncoded', -1330.9367893227916),
 ('ColorNameEncoded', -1381.1674599909884),
 ('CategoryNameEncoded', -2924.7028943715914),
 ('SeasonEncoded', -3862.7560802561534),
 ('ClassNameEncoded', -4363.592502557872),
 ('SubClassNameEncoded', -5029.53573931745),
 ('TypeNameEncoded', -13433.118063980326),
 ('ShopCode', -17926.94611840598)]
```

In [17]:
```
trainData = train.sample(frac=0.8)
y_trainSales = trainData['SalePrice']
x_trainSales = trainData[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClassNam
                    'ColorNameEncoded', 'GroupNameEncoded', 'TypeNameEncoded','S

testData = train.sample(frac=0.2)
y_testSales = testData['SalePrice']
x_testSales = testData[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClassNameE
                   'ColorNameEncoded', 'GroupNameEncoded', 'TypeNameEncoded', 'Se

decClf = RandomForestClassifier(n_estimators=100, max_depth=20,
                                random_state=0)
decClf.fit(x_trainSales,y_trainSales)

# print("On random training data,  score is:",decClf.score(x_train,y_train))
### Testing Accuracy = ???
print("Random Forest Classifier score:",decClf.score(x_testSales,y_testSales))
### The score varies due the training set being random
```

```
Random Forest Classifier score: 0.8682505399568035
```

In [18]:
```
reg = LinearRegression().fit(x_trainSales, y_trainSales)
y_predictedSales = reg.predict(x_testSales)
rms = sqrt(mean_squared_error(y_testSales, y_predictedSales))

print("RMSE For Linear Regression:" ,rms)
```

```
RMSE For Linear Regression: 825.0990819389741
```

In [19]:
```
# regr_1 = linear_model.Ridge(alpha=.5)
regr_1 = RandomForestRegressor(max_depth=25, random_state=0,
                               n_estimators=100)

regr_2Sales = DecisionTreeRegressor(max_depth=20)
regr_1.fit(x_trainSales, y_trainSales)
regr_2Sales.fit(x_trainSales, y_trainSales)

RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=0, verbose=0, warm_start=False)

y_1 = regr_1.predict(x_testSales)
y_2Sales = regr_2Sales.predict(x_testSales)

rms1 = sqrt(mean_squared_error(y_testSales, y_1))
rms2 = sqrt(mean_squared_error(y_testSales, y_2Sales))
print("RMSE For Random Forest Regression:" ,rms1)
print("RMSE For Decision Tree Regression:" ,rms2)
```

```
RMSE For Random Forest Regression: 206.1752700905291
RMSE For Decision Tree Regression: 197.48663210391675
```

In [20]:
```
def trainAndMakePredictionReady_ItemPrice():
    trainData = train.sample(frac=0.8)
    y_trainSales = trainData['SalePrice']
    x_trainSales = trainData[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClas
                          'ColorNameEncoded', 'GroupNameEncoded', 'TypeNameEncoded

    testData = train.sample(frac=0.2)
    y_testSales = testData['SalePrice']
    x_testSales = testData[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClassN
                        'ColorNameEncoded', 'GroupNameEncoded', 'TypeNameEncoded',

    regr_2Sales = DecisionTreeRegressor(max_depth=20)
    regr_2Sales.fit(x_trainSales, y_trainSales)

    RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
               max_features='auto', max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=1, min_samples_split=2,
               min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
               oob_score=False, random_state=0, verbose=0, warm_start=False)
    return regr_2Sales
```

In [21]:
```
def itemPricePrediction(ClassName, CategoryName, SubClassName,ColorName, GroupNam
    print("--------------------------------")
    print(Season)
    ClassNameEncoded = train.loc[train['ClassName'] == ClassName].ClassNameEncode
    CategoryNameEncoded = train.loc[train['CategoryName'] == CategoryName].Catego
    SubClassNameEncoded = train.loc[train['SubClassName'] == SubClassName].SubCla
    ColorNameEncoded = train.loc[train['ColorName'] == ColorName].ColorNameEncode
    GroupNameEncoded = train.loc[train['GroupName'] == GroupName].GroupNameEncode
    TypeNameEncoded = train.loc[train['TypeName'] == TypeName].TypeNameEncoded.un
    SeasonEncoded = train.loc[train['Season'] == Season].SeasonEncoded.unique()[0

    Sales = trainAndMakePredictionReady_ItemPrice().predict([[ClassNameEncoded, C
                    ColorNameEncoded, GroupNameEncoded, TypeNameEncoded,SeasonEn
    return Sales[0]
itemPricePrediction("Swiss Lawn",'Suit 03 Piece', 'Bombo Slub Shirt + Bombo Dupat
```

```
--------------------------------
Winter
```

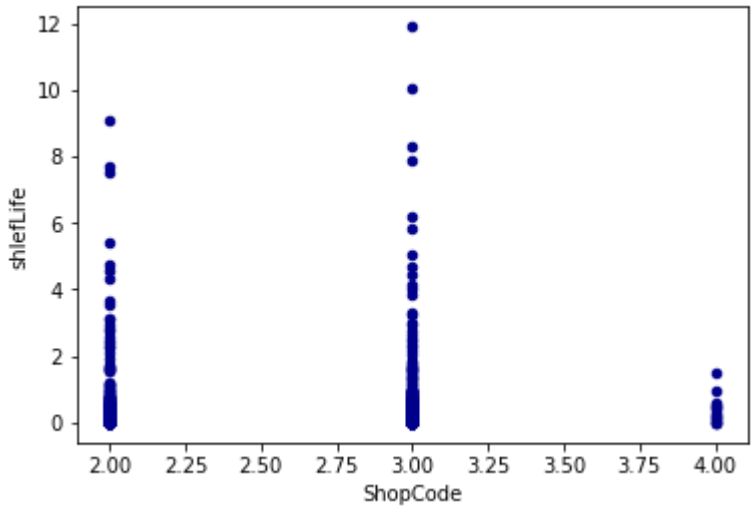
Out[21]:
```
2632.0
```

In [22]:
```
colNames = train[['ClassNameEncoded', 'CategoryNameEncoded', 'SubClassNameEncoded
                  'GroupNameEncoded', 'TypeNameEncoded','ShopCode', 'SeasonEncode
corrDict = {}
for col in colNames.columns:
    corrDict[col] = (correlation(train[col], train.shlefLife))

corrDict = (sorted(corrDict.items(), key=lambda kv: kv[1], reverse= 1))
corrDict
```

Out[22]:
```
[('GroupNameEncoded', -299.84364035701475),
 ('ColorNameEncoded', -311.1461985910116),
 ('CategoryNameEncoded', -658.8493836993512),
 ('SeasonEncoded', -870.490215778091),
 ('ClassNameEncoded', -983.4553318093098),
 ('SubClassNameEncoded', -1133.586285826131),
 ('SalePrice', -1322.2006289043802),
 ('TypeNameEncoded', -3027.81949868617),
 ('ShopCode', -4040.713429031384)]
```

In [23]:
```
ax1 = df.plot.scatter(x='ShopCode',
                      y='shlefLife',
                      c='DarkBlue')

ax1.plot()
```

Out[23]: []



In [24]:
```
trainData = train.sample(frac=0.80)
y_trainShelfLife = trainData['shlefLife']
x_trainShelfLife = trainData[['ClassNameEncoded', 'CategoryNameEncoded',  'GroupN
                     'ShopCode','SeasonEncoded', 'SalePrice']]

testData = train.sample(frac=0.20)
y_testShelfLife = testData['shlefLife']
x_testShelfLife = testData[['ClassNameEncoded', 'CategoryNameEncoded',  'GroupNam
                   'ShopCode','SeasonEncoded', 'SalePrice']]
```

In [25]:
```
reg = LinearRegression().fit(x_trainShelfLife, y_trainShelfLife)
y_predictedShelfLife = reg.predict(x_testShelfLife)
rms = sqrt(mean_squared_error(y_testShelfLife, y_predictedShelfLife))

print("RMSE For Linear Regression:" ,rms)
```

```
RMSE For Linear Regression: 0.4792012829965125
```

In [26]:
```
# regr_1 = linear_model.Ridge(alpha=.5)
regr_1 = RandomForestRegressor(max_depth=25, random_state=0,
                               n_estimators=100)

regr_2ShelfLife = DecisionTreeRegressor(max_depth=20)
regr_1.fit(x_trainShelfLife, y_trainShelfLife)
regr_2ShelfLife.fit(x_trainShelfLife, y_trainShelfLife)

RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
           max_features='auto', max_leaf_nodes=None,
           min_impurity_decrease=0.0, min_impurity_split=None,
           min_samples_leaf=1, min_samples_split=2,
           min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
           oob_score=False, random_state=0, verbose=0, warm_start=False)
# print(regr_1.feature_importances_)
# print(regr_2.feature_importances_)

y_1 = regr_1.predict(x_testShelfLife)
y_2ShelfLife = regr_2ShelfLife.predict(x_testShelfLife)

rms1 = sqrt(mean_squared_error(y_testShelfLife, y_1))
rms2 = sqrt(mean_squared_error(y_testShelfLife, y_2ShelfLife))
print("RMSE For Random Forest Regression:" ,rms1)
print("RMSE For Decision Tree Regression:" ,rms2)
```

```
RMSE For Random Forest Regression: 0.2848486628218718
RMSE For Decision Tree Regression: 0.2683883096787264
```

In [27]:
```
def trainAndMakePredictionReady_ShelfLife():
    trainData = train.sample(frac=0.80)
    y_trainShelfLife = trainData['shlefLife']
    x_trainShelfLife = trainData[['ClassNameEncoded', 'CategoryNameEncoded',  'Gr
                          'ShopCode','SeasonEncoded', 'SalePrice']]

    testData = train.sample(frac=0.20)
    y_testShelfLife = testData['shlefLife']
    x_testShelfLife = testData[['ClassNameEncoded', 'CategoryNameEncoded',  'Grou
                        'ShopCode','SeasonEncoded', 'SalePrice']]
    regr_2ShelfLife = DecisionTreeRegressor(max_depth=20)
    regr_2ShelfLife.fit(x_trainShelfLife, y_trainShelfLife)

    RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=2,
               max_features='auto', max_leaf_nodes=None,
               min_impurity_decrease=0.0, min_impurity_split=None,
               min_samples_leaf=1, min_samples_split=2,
               min_weight_fraction_leaf=0.0, n_estimators=100, n_jobs=None,
               oob_score=False, random_state=0, verbose=0, warm_start=False)

    return regr_2ShelfLife
```

In [28]:
```
def itemShelfLifePrediction(ClassName, CategoryName, GroupName, ShopCode, Season,
    ClassNameEncoded = train.loc[train['ClassName'] == ClassName].ClassNameEncode
    CategoryNameEncoded = train.loc[train['CategoryName'] == CategoryName].Catego
#     SubClassNameEncoded = train.loc[train['SubClassName'] == SubClassName].SubC
#     ColorNameEncoded = train.loc[train['ColorName'] == ColorName].ColorNameEnco
    GroupNameEncoded = train.loc[train['GroupName'] == GroupName].GroupNameEncode
#     TypeNameEncoded = train.loc[train['TypeName'] == TypeName].TypeNameEncoded.
    SeasonEncoded = train.loc[train['Season'] == Season].SeasonEncoded.unique()[0

    ShelfLife = trainAndMakePredictionReady_ShelfLife().predict([[ClassNameEncode
                            GroupNameEncoded, ShopCode ,SeasonEncoded, SalePrice]]
    return ShelfLife[0]
itemShelfLifePrediction("Swiss Lawn",'Suit 03 Piece', 'Ladies Fabric', 3, 'Winter
```

Out[28]:
```
0.027056636191749323
```

In [29]:
```
query = "SELECT top 1000 tblTrans_ReceiveAtShopDetail.ShopCode,tblTrans_ReceiveAt
```

In [30]:
```
df1 = pd.read_sql(query, sql_conn)

df1.head(100)
```

Out[30]:

	ShopCode	StoreCode	VoucherNoRef	VoucherLineNoRef	VoucherNo	VoucherLineNo	ItemCode
0	2	1	17	0001	1	0001	274
1	2	1	17	0002	1	0002	275
2	2	1	17	0003	1	0003	276
3	2	1	17	0004	1	0004	277
4	2	1	17	0005	1	0005	278
5	2	1	17	0006	1	0006	279
6	2	1	17	0007	1	0007	280
7	2	1	17	0008	1	0008	281
8	2	1	17	0009	1	0009	282
9	2	1	18	0001	2	0001	205
10	2	1	18	0002	2	0002	207
11	2	1	18	0003	2	0003	207
12	2	1	18	0004	2	0004	208
13	2	1	18	0005	2	0005	209
14	2	1	18	0006	2	0006	210
15	2	1	18	0007	2	0007	211
16	2	1	18	0008	2	0008	212
17	2	1	18	0009	2	0009	285
18	2	1	19	0001	3	0001	298
19	2	1	19	0002	3	0002	286
20	2	1	19	0003	3	0003	287
21	2	1	19	0004	3	0004	288
22	2	1	19	0005	3	0005	289
23	2	1	19	0006	3	0006	293
24	2	1	19	0007	3	0007	290
25	2	1	19	0008	3	0008	291
26	2	1	19	0009	3	0009	292
27	2	1	19	0010	3	0010	294
28	2	1	19	0011	3	0011	295
29	2	1	19	0012	3	0012	296
...	...	...	...	...	...	...	..
70	2	1	21	0014	6	0014	458
71	2	1	21	0015	6	0015	459

	ShopCode	StoreCode	VoucherNoRef	VoucherLineNoRef	VoucherNo	VoucherLineNo	ItemCode
72	2	1	21	0016	6	0016	460
73	2	1	21	0017	6	0017	461
74	2	1	21	0018	6	0018	464
75	2	1	21	0019	6	0019	462
76	2	1	21	0020	6	0020	463
77	2	1	25	0001	7	0001	324
78	2	1	25	0002	7	0002	325
79	2	1	25	0003	7	0003	326
80	2	1	25	0004	7	0004	327
81	2	1	25	0005	7	0005	328
82	2	1	25	0006	7	0006	329
83	2	1	25	0007	7	0007	330
84	2	1	25	0008	7	0008	331
85	2	1	25	0009	7	0009	332
86	2	1	25	0010	7	0010	333
87	2	1	25	0011	7	0011	334
88	2	1	25	0012	7	0012	335
89	2	1	25	0013	7	0013	336
90	2	1	25	0014	7	0014	337
91	2	1	25	0015	7	0015	338
92	2	1	25	0016	7	0016	339
93	2	1	25	0017	7	0017	340
94	2	1	25	0018	7	0018	341
95	2	1	25	0019	7	0019	342
96	2	1	25	0020	7	0020	343
97	2	1	25	0021	7	0021	344
98	2	1	25	0022	7	0022	345
99	2	1	25	0023	7	0023	346

100 rows × 18 columns

In [31]:
```
query = "SELECT tblTrans_SaleInvoiceDetail.RefVoucherNo,[RefVoucherLineNo],[Store
```

In [32]:
```
df2 = pd.read_sql(query, sql_conn)

df2.head(100)
```

Out[32]:

	RefVoucherNo	RefVoucherLineNo	StoreCode	VoucherNo	VoucherLineNo	ItemCode	DesignCo
0	0.0		1	1	0001	463	X36
1	0.0		1	2	0001	63	PLA
2	0.0		1	3	0001	469	X17
3	0.0		1	4	0001	63	PLA
4	0.0		1	5	0001	326	X34
5	0.0		1	6	0001	448	X36
6	0.0		1	6	0002	291	X18
7	0.0		1	7	0001	332	X34
8	0.0		1	8	0001	465	X17
9	0.0		1	9	0001	469	X17
10	0.0		1	10	0001	205	SPX385(
11	0.0		1	11	0001	458	X36
12	0.0		1	11	0002	293	X18
13	0.0		1	12	0001	445	X36
14	0.0		1	13	0001	295	X24
15	0.0		1	13	0002	296	X18
16	0.0		1	14	0001	465	X17
17	0.0		1	15	0001	258	X24
18	0.0		1	16	0001	467	X17
19	0.0		1	17	0001	393	PLA
20	0.0		1	18	0001	468	X17
21	0.0		1	18	0002	465	X17
22	0.0		1	19	0001	279	X29
23	0.0		1	20	0001	446	X36
24	0.0		1	20	0002	253	X27
25	0.0		1	21	0001	299	X16
26	0.0		1	22	0001	244	X24
27	0.0		1	23	0001	248	X28
28	0.0		1	23	0002	463	X36
29	0.0		1	23	0003	333	X34
...	...	...	...	...	...	...	
70	0.0		1	55	0001	454	X36
71	0.0		1	56	0001	466	X17

	RefVoucherNo	RefVoucherLineNo	StoreCode	VoucherNo	VoucherLineNo	ItemCode	DesignCo
72	0.0		1	57	0001	295	X24
73	0.0		1	58	0001	448	X36
74	0.0		1	59	0001	467	X17
75	0.0		1	60	0001	209	SPX385
76	0.0		1	61	0001	468	X17
77	0.0		1	61	0002	467	X17
78	0.0		1	61	0003	466	X17
79	0.0		1	62	0001	330	X34
80	0.0		1	63	0001	393	PLA
81	0.0		1	64	0001	63	PLA
82	0.0		1	65	0001	151	PLA
83	0.0		1	66	0001	345	X32
84	0.0		1	67	0001	456	X36
85	0.0		1	68	0001	345	X32
86	0.0		1	69	0001	467	X17
87	0.0		1	70	0001	294	X14
88	0.0		1	71	0001	254	X26
89	0.0		1	71	0002	469	X17
90	0.0		1	72	0001	63	PLA
91	0.0		1	73	0001	277	X31
92	0.0		1	74	0001	254	X26
93	0.0		1	75	0001	332	X34
94	0.0		1	76	0001	468	X17
95	0.0		1	77	0001	467	X17
96	0.0		1	77	0002	465	X17
97	0.0		1	77	0003	279	X29
98	0.0		1	77	0004	448	X36
99	0.0		1	78	0001	461	X36

100 rows × 27 columns

In [33]:
```
writer = pd.ExcelWriter('season.xlsx', engine='xlsxwriter')
newDF.to_excel(writer, sheet_name='Sheet1')
writer.save()
```

In [34]:
```
if __name__ == '__main__':
    app.run(debug=False)
```

```
 * Running on http://127.0.0.1:5000/ (http://127.0.0.1:5000/) (Press CTRL+C to
 quit)
127.0.0.1 - - [19/May/2019 18:00:06] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:17] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:19] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:20] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:21] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:22] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:23] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:23] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:24] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:25] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:26] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:27] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:28] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:00:36] "POST /api/PredictShelfLife HTTP/1.1" 201
 -
127.0.0.1 - - [19/May/2019 18:02:17] "POST /api/PredictPrice HTTP/1.1" 201 -

---------------------------------
Winter
```

In [ ]:
```

```