In [47]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from subprocess import check_output
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
df = pd.read_csv(r"C:\Users\Brent\Documents\GitHub\DataScience\Challenges\Personal
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 16598 entries, 0 to 16597
Data columns (total 11 columns):
Rank            16598 non-null int64
Name            16598 non-null object
Platform        16598 non-null object
Year            16327 non-null float64
Genre           16598 non-null object
Publisher       16540 non-null object
NA_Sales        16598 non-null float64
EU_Sales        16598 non-null float64
JP_Sales        16598 non-null float64
Other_Sales     16598 non-null float64
Global_Sales    16598 non-null float64
dtypes: float64(6), int64(1), object(4)
memory usage: 1.4+ MB
```

In [2]:

```
df.head(10)
```

Out[2]:

	Rank	Name	Platform	Year	Genre	Publisher	NA_Sales	EU_Sales	JP_Sales	C
0	1	Wii Sports	Wii	2006.0	Sports	Nintendo	41.49	29.02	3.77	
1	2	Super Mario Bros.	NES	1985.0	Platform	Nintendo	29.08	3.58	6.81	
2	3	Mario Kart Wii	Wii	2008.0	Racing	Nintendo	15.85	12.88	3.79	
3	4	Wii Sports Resort	Wii	2009.0	Sports	Nintendo	15.75	11.01	3.28	
4	5	Pokemon Red/Pokemon Blue	GB	1996.0	Role-Playing	Nintendo	11.27	8.89	10.22	
5	6	Tetris	GB	1989.0	Puzzle	Nintendo	23.20	2.26	4.22	
6	7	New Super Mario Bros.	DS	2006.0	Platform	Nintendo	11.38	9.23	6.50	
7	8	Wii Play	Wii	2006.0	Misc	Nintendo	14.03	9.20	2.93	
8	9	New Super Mario Bros. Wii	Wii	2009.0	Platform	Nintendo	14.59	7.06	4.70	
9	10	Duck Hunt	NES	1984.0	Shooter	Nintendo	26.93	0.63	0.28	

In [3]:

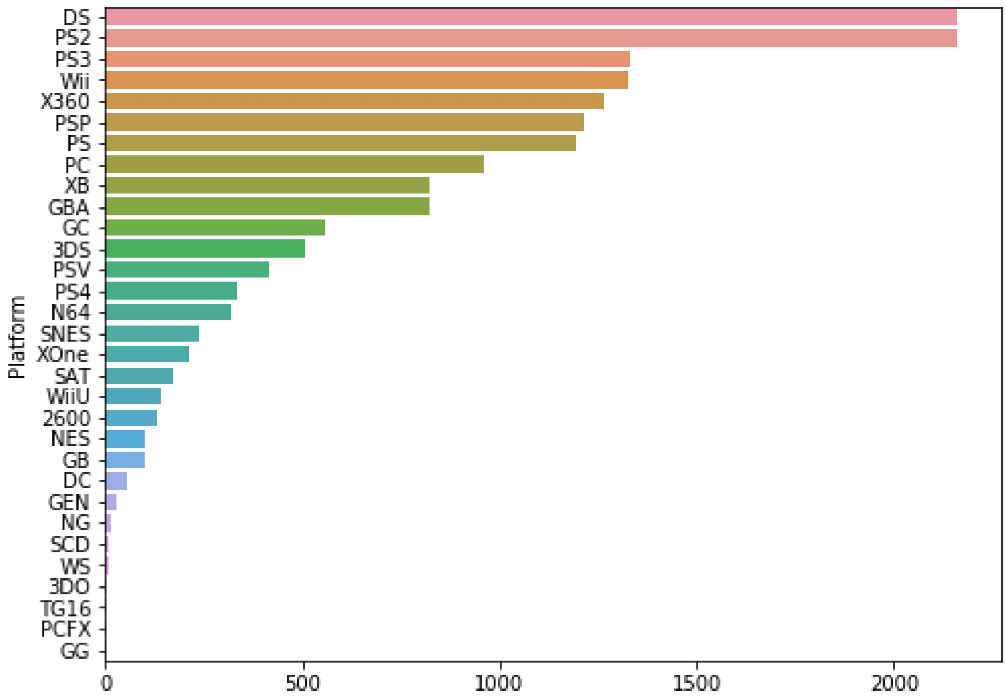
```
df.isna().sum()
```

Out[3]:

```
Rank              0
Name              0
Platform          0
Year            271
Genre             0
Publisher        58
NA_Sales          0
EU_Sales          0
JP_Sales          0
Other_Sales       0
Global_Sales      0
dtype: int64
```

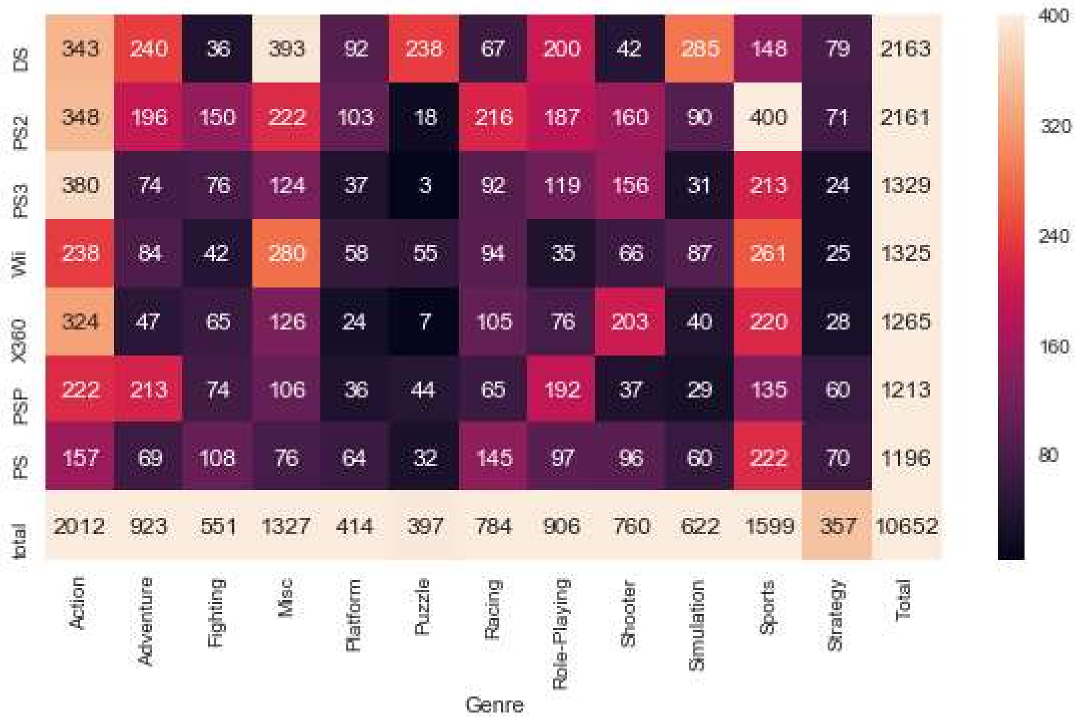
In [4]:

```
## I want to know how many games each platform has. i found a function called cross
## me with visualizing this.
platGenre = pd.crosstab(df.Platform,df.Genre)
platGenreTotal = platGenre.sum(axis=1).sort_values(ascending = False)
plt.figure(figsize=(8,6))
sns.barplot(y = platGenreTotal.index, x = platGenreTotal.values, orient='h')
plt.ylabel = "Platform"
plt.xlabel = "The amount of games"
plt.show()
```



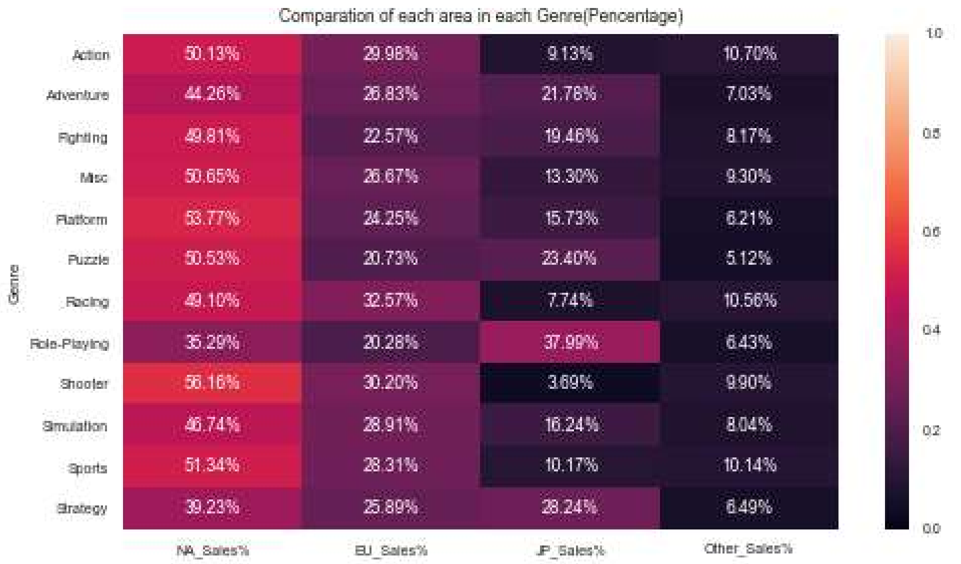
In [5]:

```
## Now i want to focus on the platforms that have 1000 or more games.
platGenre['Total'] = platGenre.sum(axis=1)
popPlatform = platGenre[platGenre['Total']>1000].sort_values(by='Total', ascending
neededdata = popPlatform.loc[:,:'Strategy']
maxi = neededdata.values.max()
mini = neededdata.values.min()
popPlatformfinal = popPlatform.append(pd.DataFrame(popPlatform.sum(), columns=['tot
sns.set(font_scale=1.0)
plt.figure(figsize=(10,5))
sns.heatmap(popPlatformfinal, vmin = mini, vmax = maxi, annot=True, fmt="d")
plt.xticks(rotation = 90)
plt.show()
```



In [6]:

```
GenreGroup = df.groupby(['Genre']).sum().loc[:, 'NA_Sales':'Global_Sales']
GenreGroup['NA_Sales%'] = GenreGroup['NA_Sales']/GenreGroup['Global_Sales']
GenreGroup['EU_Sales%'] = GenreGroup['EU_Sales']/GenreGroup['Global_Sales']
GenreGroup['JP_Sales%'] = GenreGroup['JP_Sales']/GenreGroup['Global_Sales']
GenreGroup['Other_Sales%'] = GenreGroup['Other_Sales']/GenreGroup['Global_Sales']
plt.figure(figsize=(8, 10))
sns.set(font_scale=0.7)
plt.subplot(211)
sns.heatmap(GenreGroup.loc[:, 'NA_Sales':'Other_Sales'], annot=True, fmt = '.1f')
plt.title("Comparation of each area in each Genre")
plt.subplot(212)
sns.heatmap(GenreGroup.loc[:,'NA_Sales%':'Other_Sales%'], vmax =1, vmin=0, annot=Tr
plt.title("Comparation of each area in each Genre(Pencentage)")
plt.show()
```



In [7]:

```
li = ['NA_Sales','EU_Sales','JP_Sales','Other_Sales','Global_Sales']

for i in li:
    print(i,end=" -->> ")
    print(df[i].mean(),end=" million")
    print("\n")
```

```
NA_Sales -->> 0.26466742981084057 million

EU_Sales -->> 0.1466520062658483 million

JP_Sales -->> 0.07778166044101108 million

Other_Sales -->> 0.048063019640913515 million

Global_Sales -->> 0.53744065550074 million
```

Type *Markdown* and LaTeX: α^2

Finding Correlations

First i want to group up the publishers and platforms that don't have a lot of games. In my opinion, this will make the visualizations cleaner and analyzing more easier.

In [8]:

```
for i in df['Publisher'].unique():
    if df['Publisher'][df['Publisher'] == i].count() < 50:
        df['Publisher'][df['Publisher'] == i] = 'Other'

for i in df['Platform'].unique():
    if df['Platform'][df['Platform'] == i].count() < 100:
        df['Platform'][df['Platform'] == i] = 'Other'
```

```
E:\Anaconda\lib\site-packages\ipykernel_launcher.py:3: SettingWithCopy
Warning:
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/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  This is separate from the ipykernel package so we can avoid doing im
ports until
E:\Anaconda\lib\site-packages\ipykernel_launcher.py:7: SettingWithCopy
Warning:
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/stable/indexing.html#indexing-view-versus-copy (http://pandas.pyd
ata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy)
  import sys
```

In [9]:

```
df['Publisher'].value_counts()
```

Out[9]:

```
Other                                     3342
Electronic Arts                           1351
Activision                                 975
Namco Bandai Games                         932
Ubisoft                                    921
Konami Digital Entertainment               832
THQ                                        715
Nintendo                                   703
Sony Computer Entertainment                683
Sega                                       639
Take-Two Interactive                       413
Capcom                                     381
Atari                                      363
Tecmo Koei                                 338
Square Enix                                233
Warner Bros. Interactive Entertainment     232
Disney Interactive Studios                 218
Unknown                                    203
Eidos Interactive                          198
Midway Games                               198
505 Games                                  192
Microsoft Game Studios                     189
Acclaim Entertainment                      184
D3Publisher                                184
Vivendi Games                              164
Codemasters                                152
Idea Factory                               129
Deep Silver                                122
Nippon Ichi Software                       105
Zoo Digital Publishing                     104
Majesco Entertainment                       92
LucasArts                                   90
Rising Star Games                           86
Hudson Soft                                 81
Banpresto                                   73
Bethesda Softworks                          71
Crave Entertainment                         71
Atlus                                       67
Infogrames                                  62
Virgin Interactive                          62
5pb                                         61
Ignition Entertainment                      61
Focus Home Interactive                      58
Marvelous Interactive                       56
SquareSoft                                  52
Empire Interactive                          52
Kadokawa Shoten                             50
Name: Publisher, dtype: int64
```

In [10]:

```
df['Platform'].value_counts()
```

Out[10]:

```
DS       2163
PS2      2161
PS3      1329
Wii      1325
X360     1265
PSP      1213
PS       1196
PC        960
XB        824
GBA       822
GC        556
3DS       509
PSV       413
PS4       336
N64       319
Other     306
SNES      239
XOne      213
SAT       173
WiiU      143
2600      133
Name: Platform, dtype: int64
```

Much cleaner lists in my opinion just by grouping up the publishers and platforms with not a lot of games.

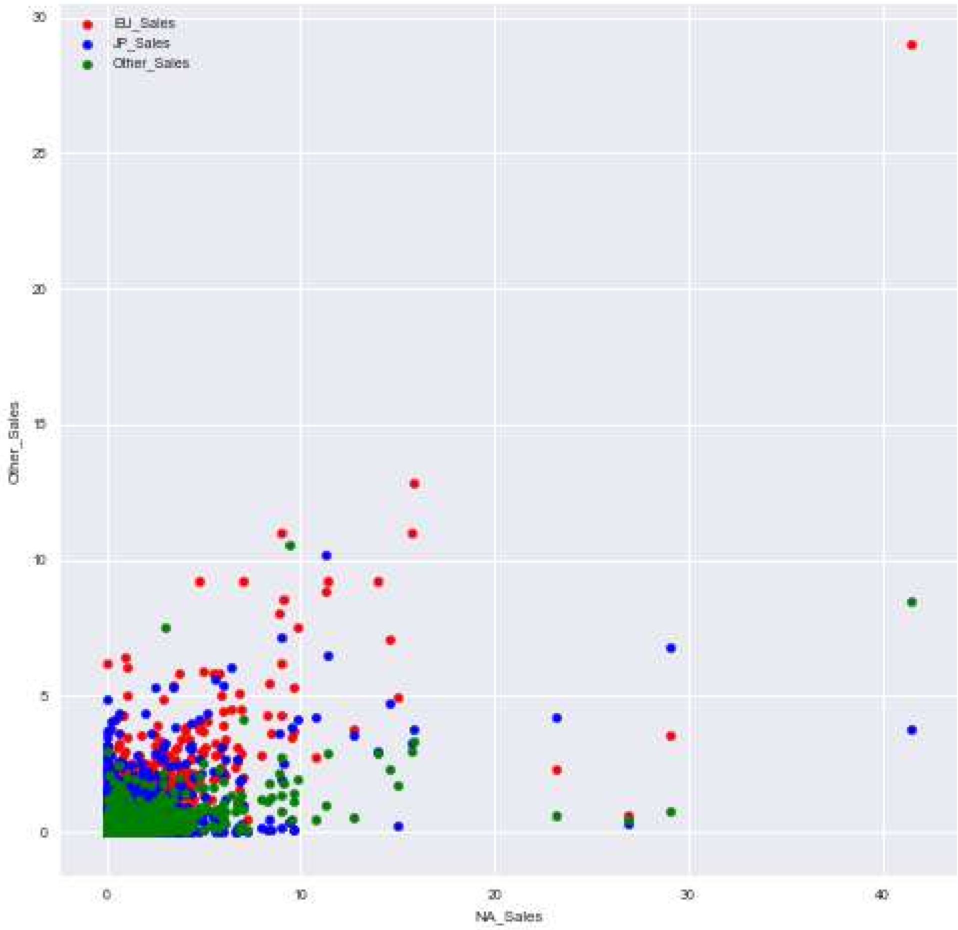
In [ ]:

```

```

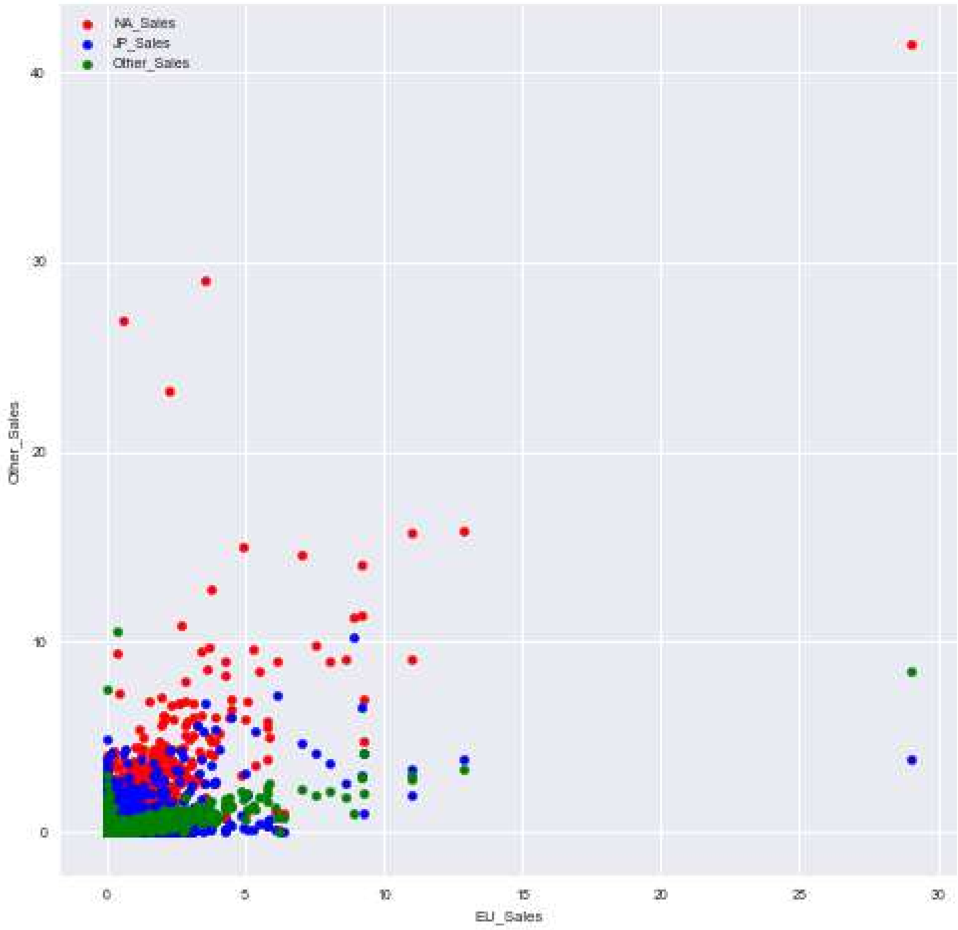
In [11]:

```
ax = df.plot(kind='scatter',x='NA_Sales',y='EU_Sales',color='red',label='EU_Sales',
df.plot(kind='scatter',x='NA_Sales',y='JP_Sales',ax=ax,color='blue',label='JP_Sales
df.plot(kind='scatter',x='NA_Sales',y='Other_Sales',ax=ax,color='green',label='Othe
plt.show()
```



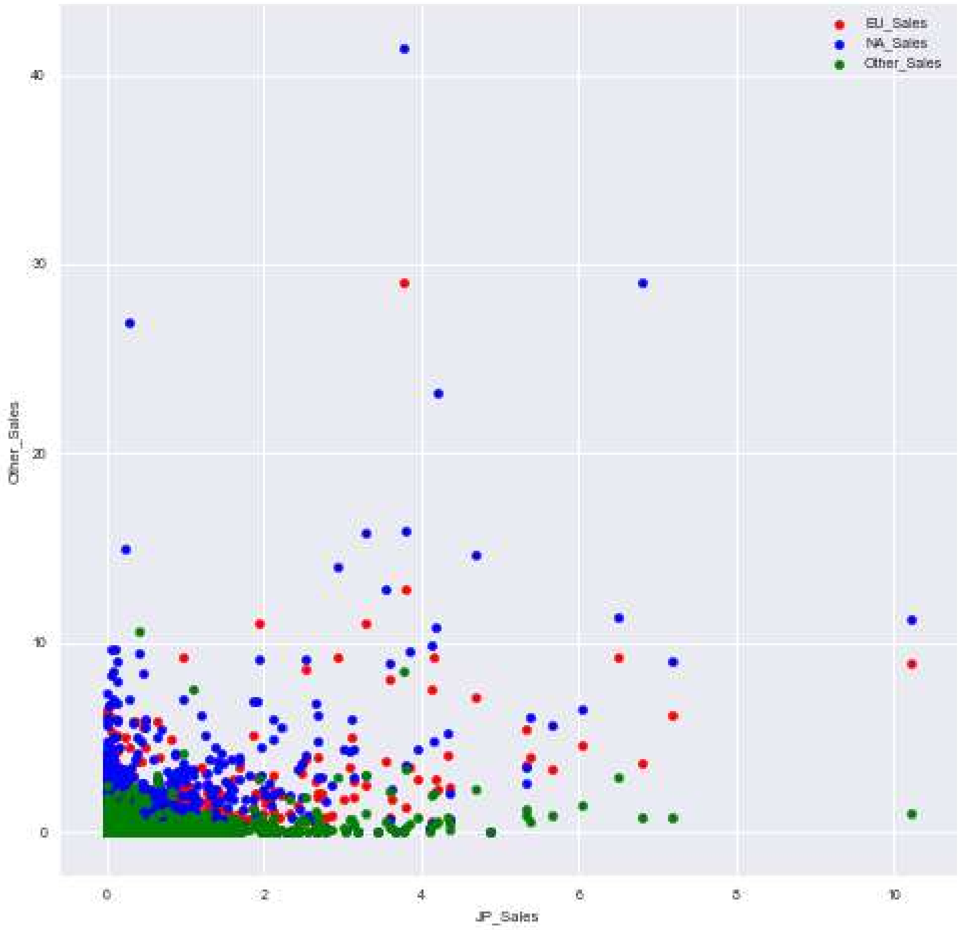
In [12]:

```
ax = df.plot(kind='scatter',x='EU_Sales',y='NA_Sales',color='red',label='NA_Sales',
df.plot(kind='scatter',x='EU_Sales',y='JP_Sales',ax=ax,color='blue',label='JP_Sales
df.plot(kind='scatter',x='EU_Sales',y='Other_Sales',ax=ax,color='green',label='Othe
plt.show()
```



In [13]:

```
ax = df.plot(kind='scatter',x='JP_Sales',y='EU_Sales',color='red',label='EU_Sales',
df.plot(kind='scatter',x='JP_Sales',y='NA_Sales',ax=ax,color='blue',label='NA_Sales
df.plot(kind='scatter',x='JP_Sales',y='Other_Sales',ax=ax,color='green',label='Othe
plt.show()
```



In [14]:

```
corr_1 = []
corr_2 = []
corr_res = []
sales_list = ['NA_Sales','EU_Sales','JP_Sales','Other_Sales','Global_Sales']
for i in sales_list:
    for j in sales_list:
        corr_1.append(i)
        corr_2.append(j)
        corr_res.append(df[i].corr(df[j]))
corr_data = pd.DataFrame(
    {'Corr_1': corr_1,
     'Corr_2': corr_2,
     'Correlation': corr_res
    })
corr_data
```

Out[14]:

	Corr_1	Corr_2	Correlation
0	NA_Sales	NA_Sales	1.000000
1	NA_Sales	EU_Sales	0.767727
2	NA_Sales	JP_Sales	0.449787
3	NA_Sales	Other_Sales	0.634737
4	NA_Sales	Global_Sales	0.941047
5	EU_Sales	NA_Sales	0.767727
6	EU_Sales	EU_Sales	1.000000
7	EU_Sales	JP_Sales	0.435584
8	EU_Sales	Other_Sales	0.726385
9	EU_Sales	Global_Sales	0.902836
10	JP_Sales	NA_Sales	0.449787
11	JP_Sales	EU_Sales	0.435584
12	JP_Sales	JP_Sales	1.000000
13	JP_Sales	Other_Sales	0.290186
14	JP_Sales	Global_Sales	0.611816
15	Other_Sales	NA_Sales	0.634737
16	Other_Sales	EU_Sales	0.726385
17	Other_Sales	JP_Sales	0.290186
18	Other_Sales	Other_Sales	1.000000
19	Other_Sales	Global_Sales	0.748331
20	Global_Sales	NA_Sales	0.941047
21	Global_Sales	EU_Sales	0.902836
22	Global_Sales	JP_Sales	0.611816
23	Global_Sales	Other_Sales	0.748331
24	Global_Sales	Global_Sales	1.000000

In [15]:

```
corr_data = corr_data.pivot(values='Correlation',index='Corr_1',columns='Corr_2')
corr_data
```

Out[15]:

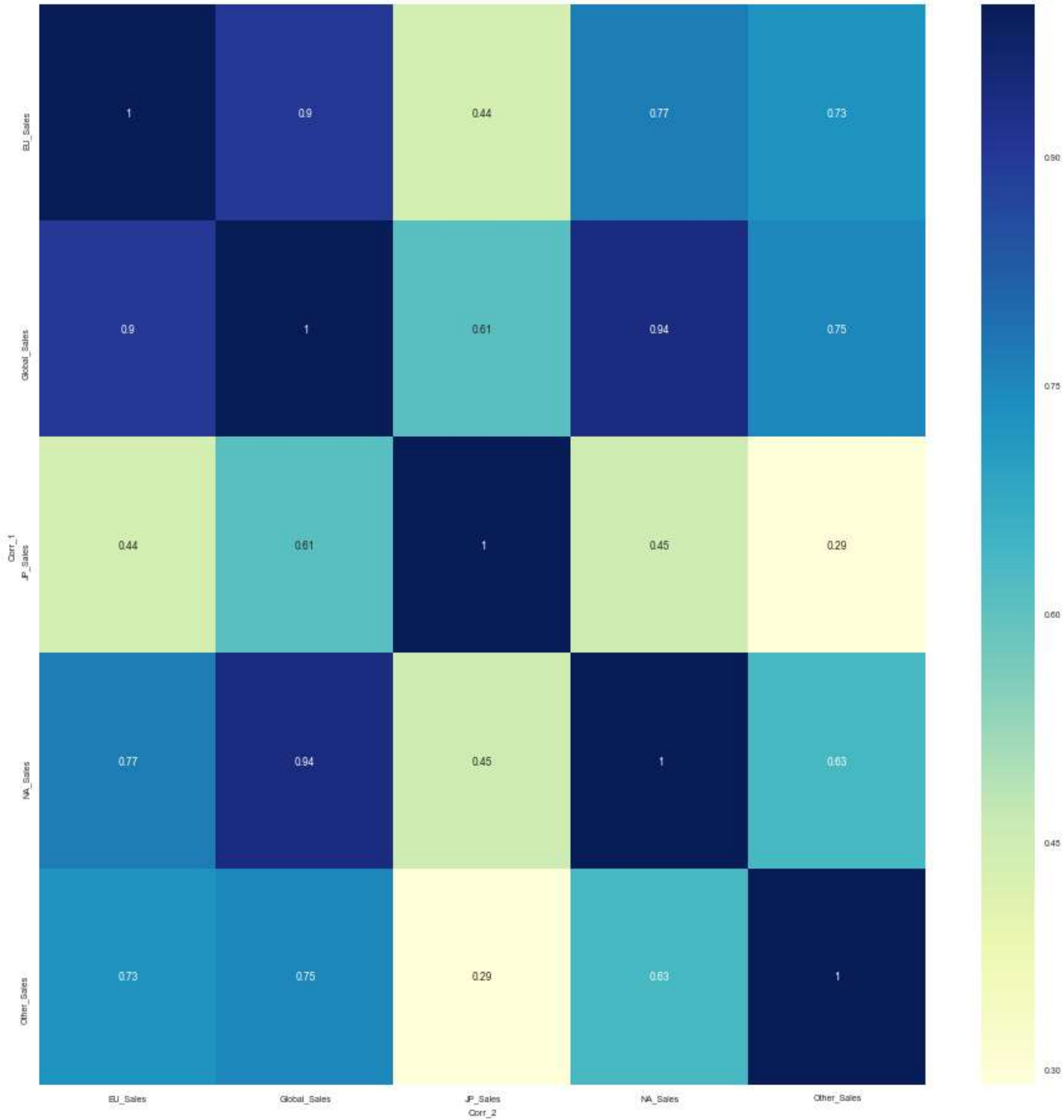
Corr_2 Corr_1	EU_Sales	Global_Sales	JP_Sales	NA_Sales	Other_Sales
EU_Sales	1.000000	0.902836	0.435584	0.767727	0.726385
Global_Sales	0.902836	1.000000	0.611816	0.941047	0.748331
JP_Sales	0.435584	0.611816	1.000000	0.449787	0.290186
NA_Sales	0.767727	0.941047	0.449787	1.000000	0.634737
Other_Sales	0.726385	0.748331	0.290186	0.634737	1.000000

In [16]:

```
plt.figure(figsize=(16,16))
sns.heatmap(corr_data,cmap = "YlGnBu", annot=True)
```

Out[16]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a1e6df9278>
```

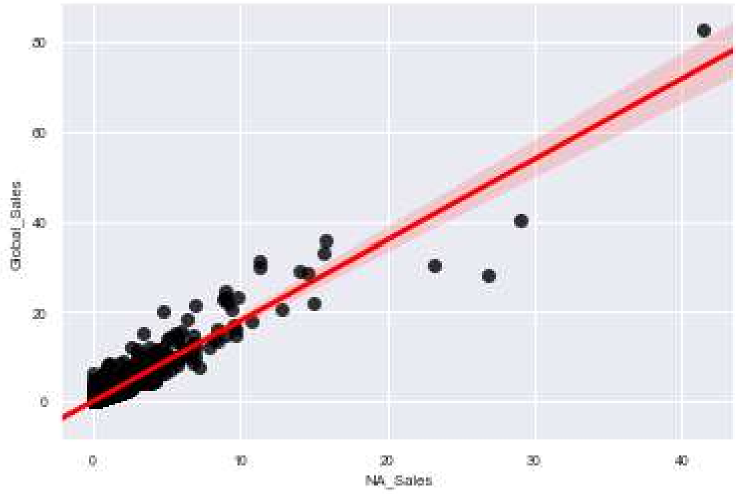


In [17]:

```
sns.regplot(x="NA_Sales", y="Global_Sales", data=df, scatter_kws={"color": "black"}
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2a1e6e2a9b0>
```



Training

I want to predict something useing NA sales and Global sales as these have the highest correlation. These are columns 6 and 10 in my dataframe.

In [56]:

```
X = df.iloc[:, 6].values
y = df.iloc[:, 10].values
```

In [57]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.10, random_
```

In [59]:

```
#I have to reshape these in order to run the regressor.
X_train = X_train.reshape((14938,1))
y_train = y_train.reshape((14938,1))
X_test = X_test.reshape((1660,1))
y_test = y_test.reshape((1660,1))
```

In [60]:

```
regressor = LinearRegression()
regressor.fit(X_test, y_test)
```

Out[60]:

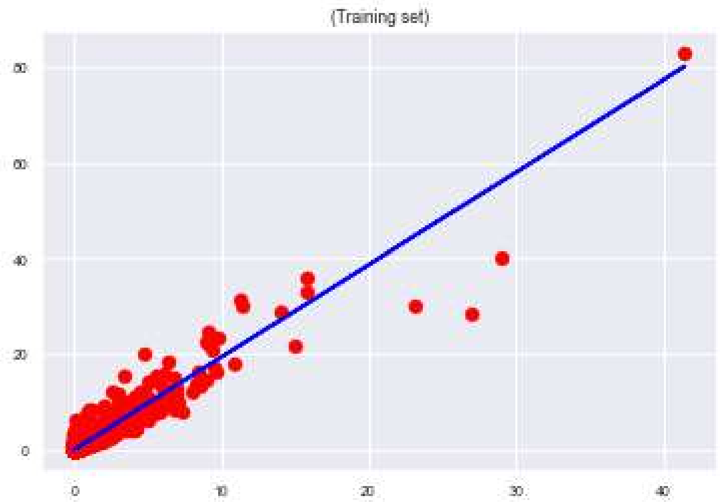
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=
False)
```

In [25]:

```
y_pred = regressor.predict(X_test)
```

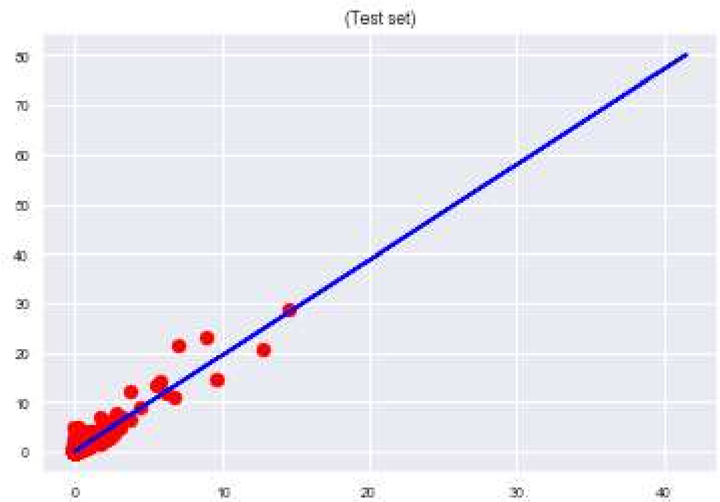
In [32]:

```
#For some reason i can't set labels to the axi. Errormessage: 'str' object is not c
# X-axis = North America sales, Y-axis = Global Sales
plt.scatter(X_train, y_train,color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('(Training set)')
plt.show()
```



In [31]:

```
plt.scatter(X_test, y_test,color='red')
plt.plot(X_train, regressor.predict(X_train), color='blue')
plt.title('(Test set)')
plt.show()
```



In [36]:

```
print("Training set score: {:.2f}".format(regressor.score(X_train,y_train)))
print("Test set score: {:.2f}".format(regressor.score(X_test,y_test)))
```

```
Training set score: 0.88
Test set score: 0.90
```

Pretty good scores for the Linear regressor. Now i'm going to try a decision tree.

In [38]:

```
X = X.reshape((16598,1))
y = y.reshape((16598,1))

TreeReg = DecisionTreeRegressor(random_state=0)
TreeReg.fit(X,y)
```

Out[38]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=No
ne,
           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,
           presort=False, random_state=0, splitter='best')
```

In [40]:

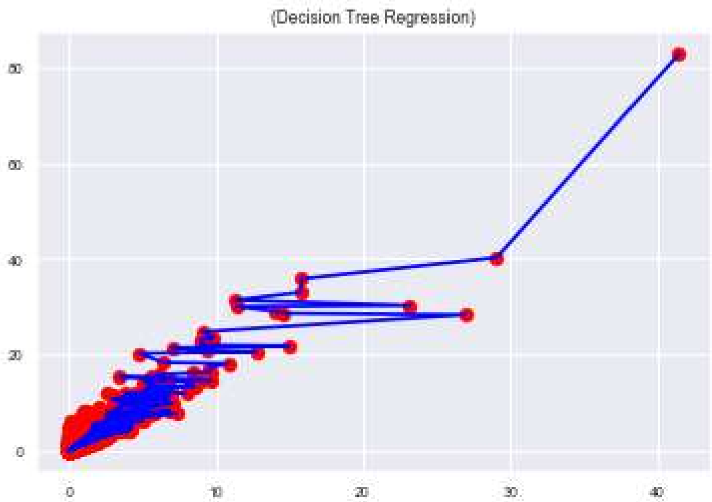
```
y_pred = TreeReg.predict(6.5)
y_pred
```

Out[40]:

```
array([18.36])
```

In [48]:

```
#Same as with the linear regression, i can't set labels to the axi.
plt.scatter(X, y, color = 'red')
plt.plot(X, TreeReg.predict(X), color = 'blue')
plt.title('(Decision Tree Regression)')
plt.show()
```



In [ ]:

```

```

In [49]:

```
print("Decision tree score: {:.2f}".format(TreeReg.score(X,y)))
```

```
Decision tree score: 0.95
```

0.95 is very good, but can SVR do better?

In [50]:

```
SVRregressor = SVR(kernel = 'rbf')
SVRregressor.fit(X,y)
```

```
E:\Anaconda\lib\site-packages\sklearn\utils\validation.py:578: DataCon
versionWarning: A column-vector y was passed when a 1d array was expec
ted. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

Out[50]:

```
SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='au
to',
  kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

In [51]:

```
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
```

In [52]:

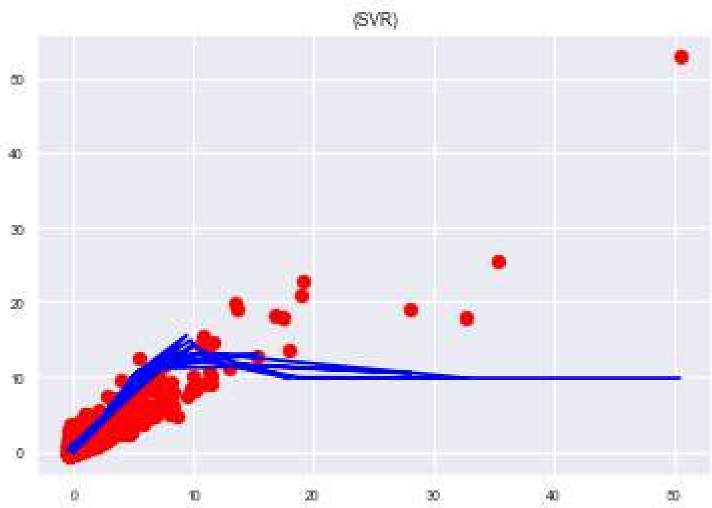
```
y_pred = sc_y.inverse_transform(SVRregressor.predict(sc_X.transform(np.array([[6.5]
y_pred
```

Out[52]:

```
array([18.12001474])
```

In [54]:

```
plt.scatter(X, y, color = 'red')
plt.plot(X, SVRregressor.predict(X), color = 'blue')
plt.title('(SVR)')
plt.show()
```



In [55]:

```
print("SVR score: {:.2f}".format(SVRregressor.score(X,y)))
```

```
SVR score: 0.31
```

SVR doesn't seem to have a good score, so i'll use the decision tree and linear regression to draw a conclusion.

In [ ]:

```
```