Visualization

In [0]:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import sys
import seaborn as sns
```

In [0]:

```
df = pd.read_csv("pokemon_alopez247.csv", sep=",")
```

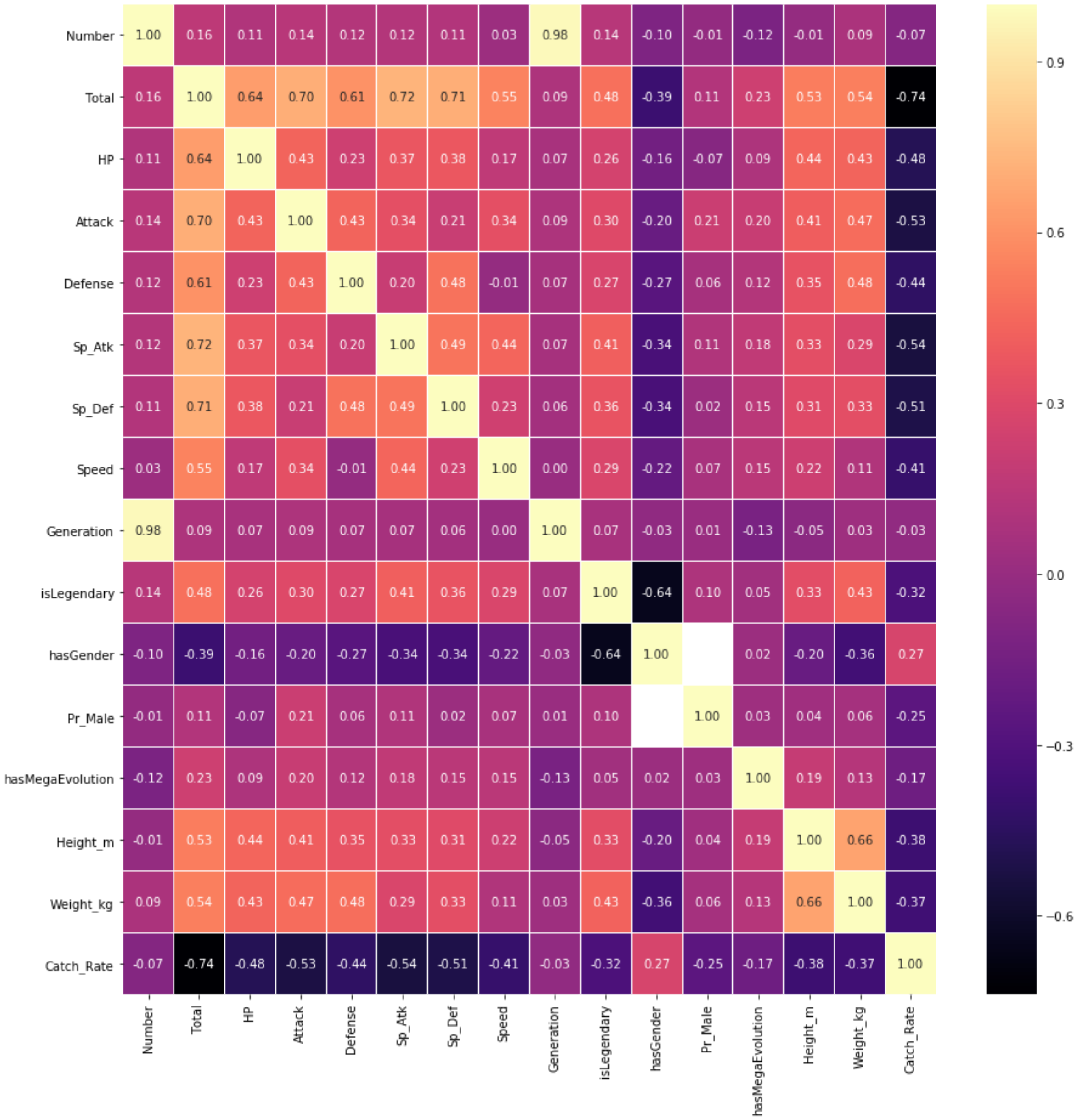
In [3]:

```
print("There are", len(df.columns), "columns:")
for x in df.columns:
    sys.stdout.write(str(x)+", ")
```

```
There are 23 columns:
Number, Name, Type_1, Type_2, Total, HP, Attack, Defense, Sp_Atk, Sp_Def, Speed, Gen
eration, isLegendary, Color, hasGender, Pr_Male, Egg_Group_1, Egg_Group_2, hasMegaEv
olution, Height_m, Weight_kg, Catch_Rate, Body_Style,
```

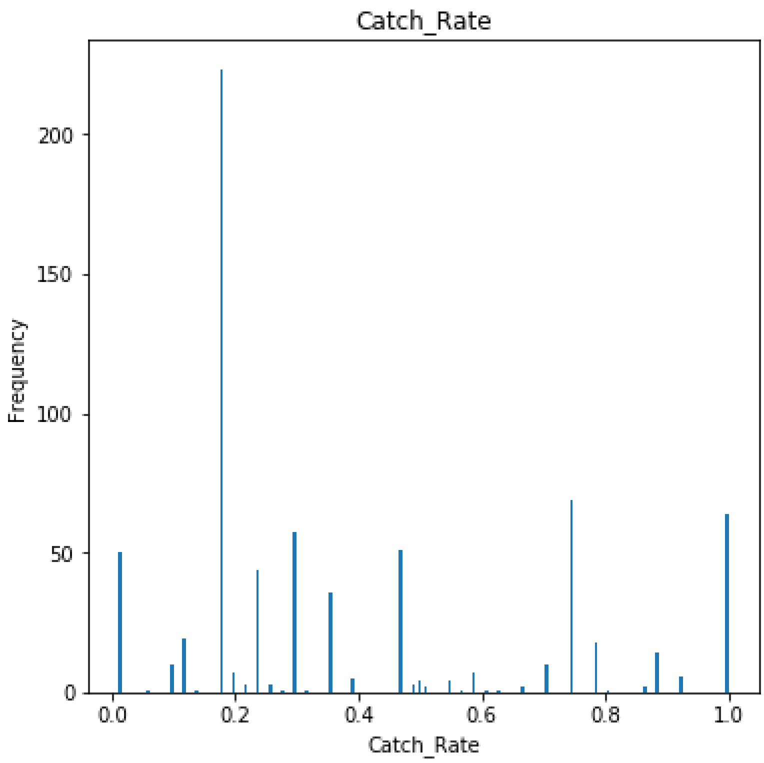
In [4]:

```
fig, ax = plt.subplots(figsize=(15, 15))
sns.heatmap(df.corr(), ax=ax, annot=True, linewidths=0.05, fmt='.2f', cmap="magma")
plt.show()
```



In [5]:

```
Catch_Rate = df["Catch_Rate"]
Catch_Rate_ = Catch_Rate/255
Catch_Rate_.plot(kind='hist', bins=200, figsize=(6, 6))
plt.title("Catch_Rate")
plt.xlabel("Catch_Rate")
plt.ylabel("Frequency")
plt.show()
```



In [6]:

```
number = df["Number"]
print('Total number of Pokemons is', len(number))
Legendary = df["isLegendary"]
rate = np.mean(Legendary == True)
print('legendary rate=', rate)
```

```
Total number of Pokemons is 721
legendary rate= 0.0638002773925104
```

In [0]:

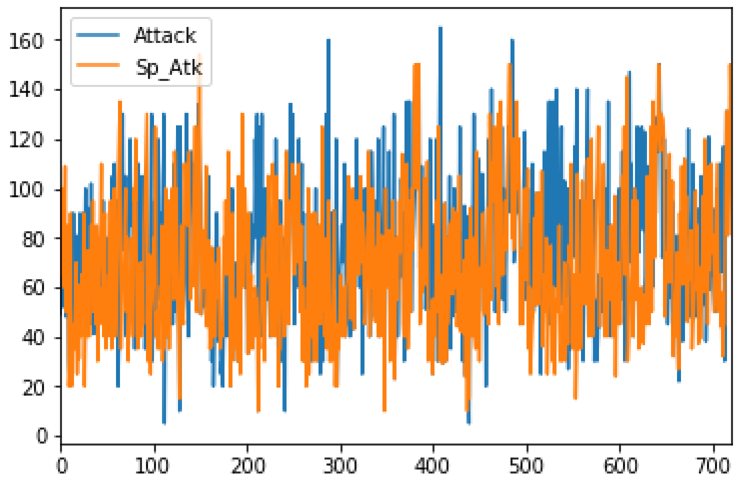
```
# Unnecessary columns
# Number and Name are just identifiers
# Total is a aggregation of others columns
clean_df = df.drop(columns=['Number', 'Name', 'Total'])
```

In [8]:

```
df2 = df.loc[:, ["Attack", "Sp_Atk"]]
df2.plot()
```

Out[8]:

<matplotlib.axes._subplots.AxesSubplot at 0x7fd6f6e2b5f8>

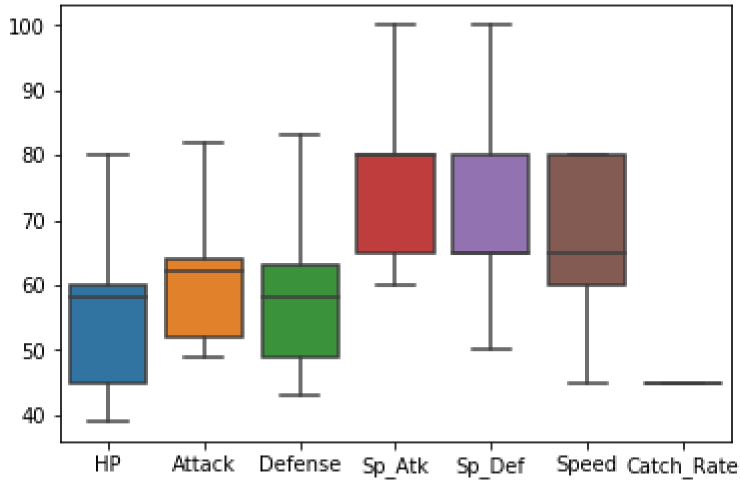


In [9]:

```
sns.boxplot(data=df.drop(['isLegendary', 'Generation', 'Number', 'Total', 'Color', 'hasGender', 'P
                          'Egg_Group_1', 'Egg_Group_2', 'hasMegaEvolution', 'Height_m', 'Weight_kg
plt.show
```

Out[9]:

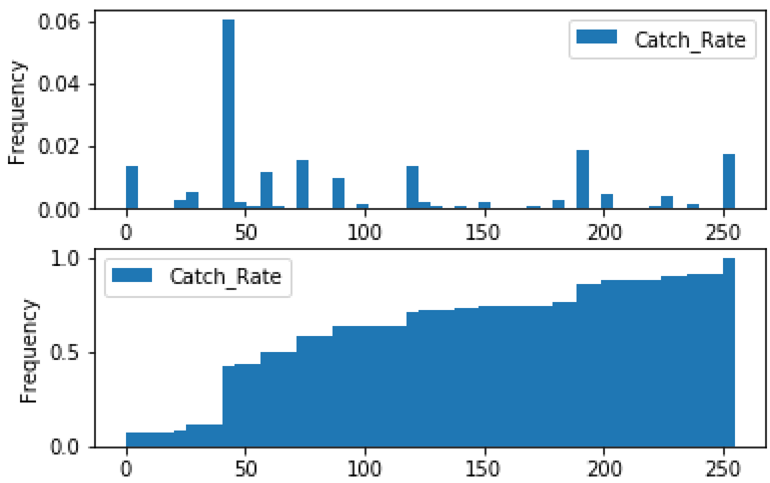
```
<function matplotlib.pyplot.show>
```



In [10]:

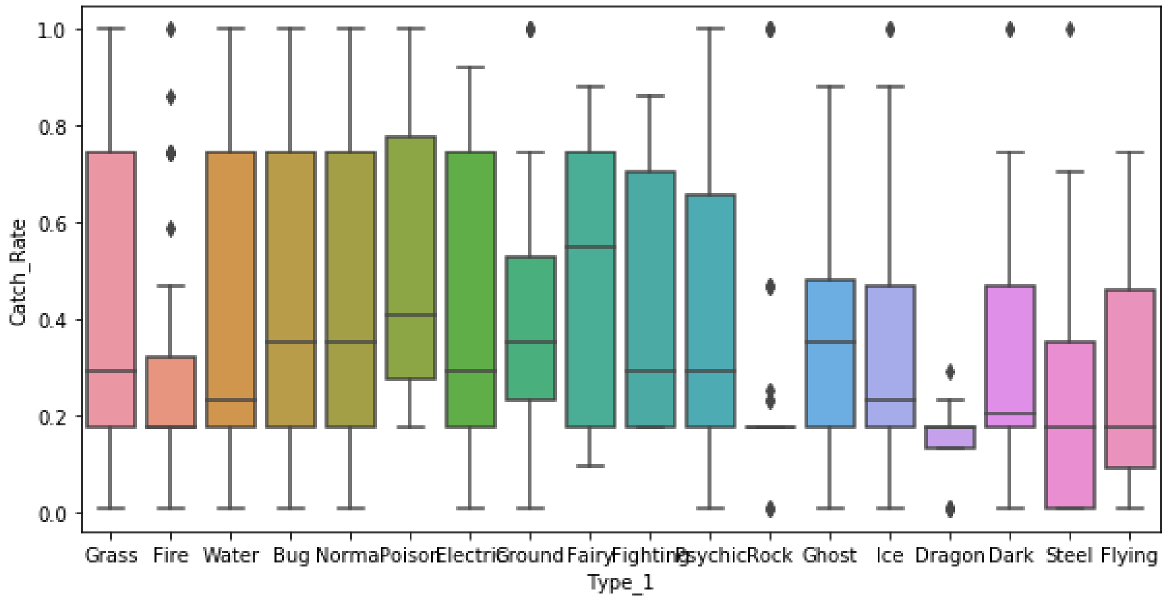
```
fig, axes = plt.subplots(nrows=2, ncols=1)
df.plot(kind="hist", y="Catch_Rate", bins=50, range=(0, 255), normed=True, ax=axes[0])
df.plot(kind="hist", y="Catch_Rate", bins=50, range=(0, 255), normed=True, ax=axes[1], cumulative=
plt.show()
# there's a sudden increase around 0.16 percentage of Catch_Rate
```

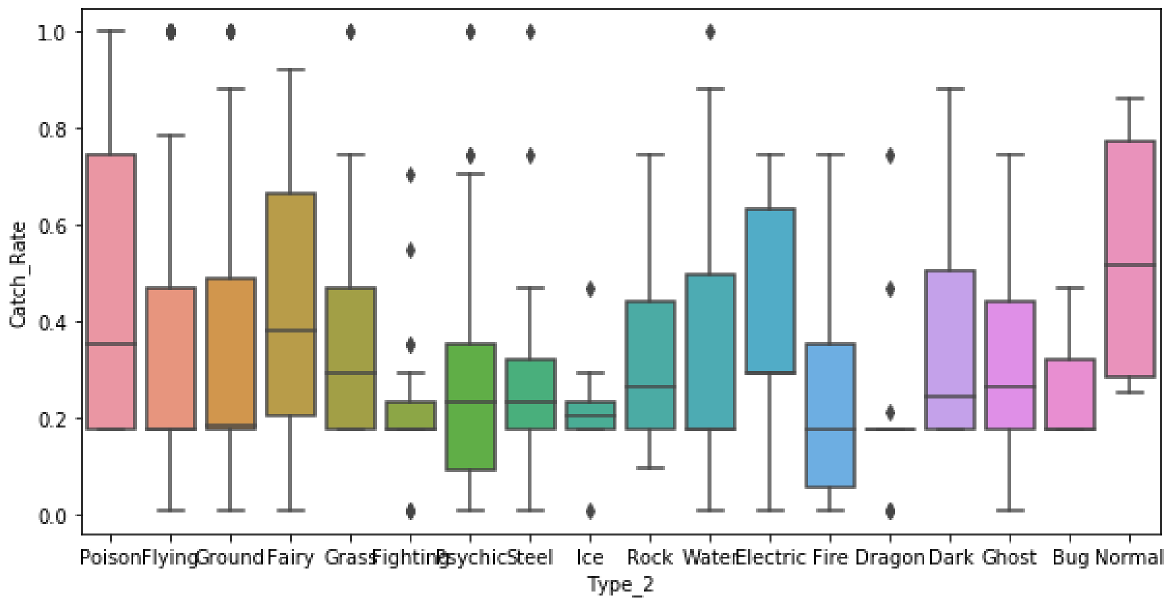
```
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: MatplotlibDepr
ecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use
'density' instead.
  alternative="'density'", removal="3.1")
/usr/local/lib/python3.6/dist-packages/matplotlib/axes/_axes.py:6521: MatplotlibDepr
ecationWarning:
The 'normed' kwarg was deprecated in Matplotlib 2.1 and will be removed in 3.1. Use
'density' instead.
  alternative="'density'", removal="3.1")
```



In [11]:

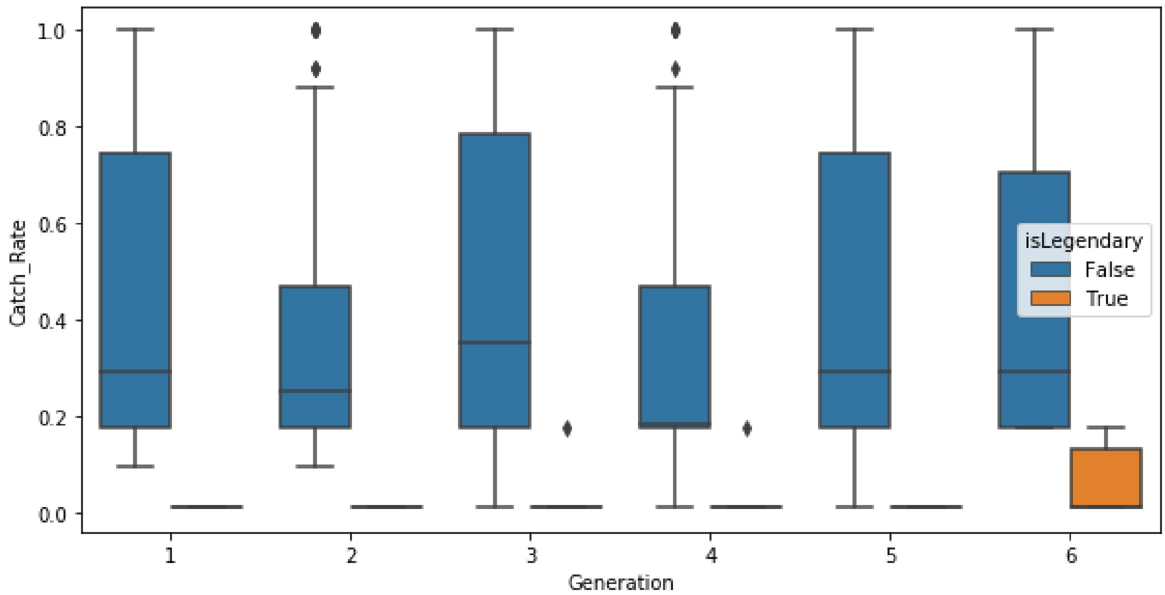
```
plt.figure(figsize=(10, 5))
ax = sns.boxplot(x='Type_1', y=Catch_Rate_, data=df)
plt.figure(figsize=(10, 5))
ax = sns.boxplot(x='Type_2', y=Catch_Rate_, data=df)
```





In [12]:

```
# If generation and islegendary relate to Catch_rate
plt.figure(figsize=(10, 5))
ax = sns.boxplot(x='Generation', y=Catch_Rate_, hue='isLegendary', data=df)
```



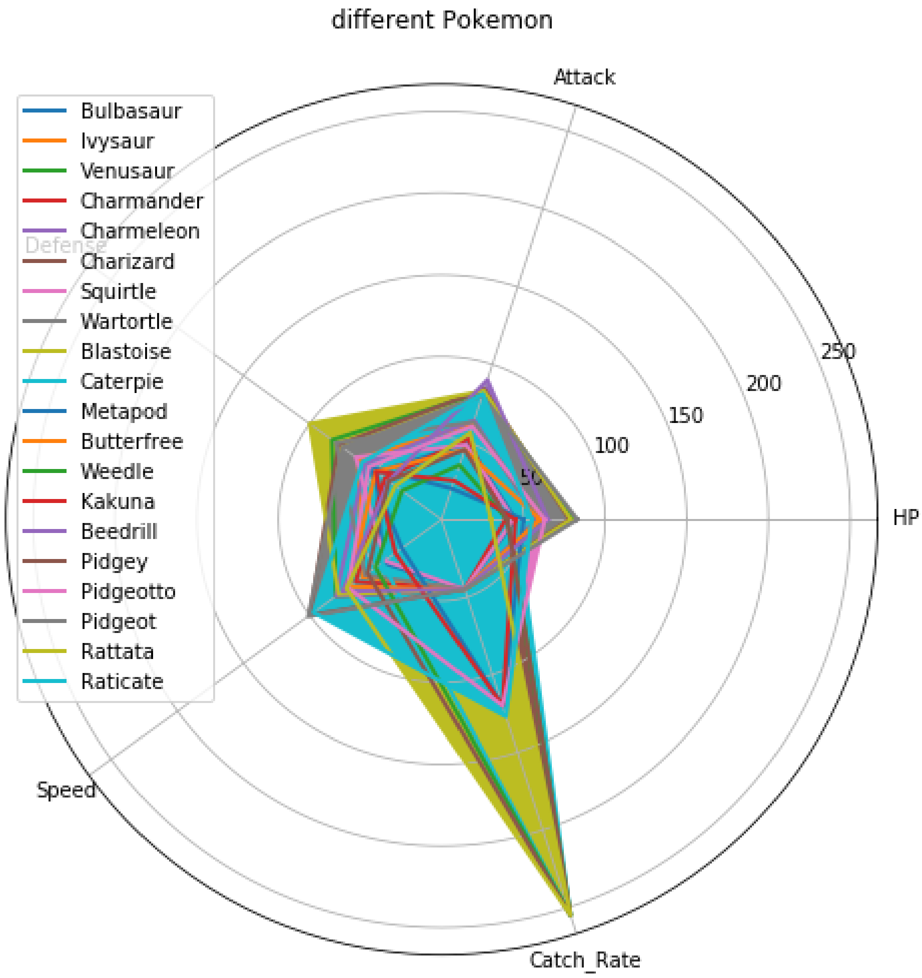
In [13]:

```
result1 = df.drop(['Type_1', 'Type_2', 'Sp_Atk', 'Sp_Def', 'isLegendary', 'Generation', 'Number',
                   'Pr_Male', 'Egg_Group_1', 'Egg_Group_2', 'hasMegaEvolution', 'Height_m', 'Weigh
print(result1)
```

```
          Name  HP  Attack  Defense  Speed  Catch_Rate
0     Bulbasaur  45      49       49     45          45
1       Ivysaur  60      62       63     60          45
2      Venusaur  80      82       83     80          45
3    Charmander  39      52       43     65          45
4    Charmeleon  58      64       58     80          45
5     Charizard  78      84       78    100          45
6      Squirtle  44      48       65     43          45
7     Wartortle  59      63       80     58          45
8     Blastoise  79      83      100     78          45
9      Caterpie  45      30       35     45         255
10      Metapod  50      20       55     30         120
11   Butterfree  60      45       50     70          45
12       Weedle  40      35       30     50         255
13       Kakuna  45      25       50     35         120
14     Beedrill  65      90       40     75          45
15       Pidgey  40      45       40     56         255
16    Pidgeotto  63      60       55     71         120
17      Pidgeot  83      80       75    101          45
18      Rattata  30      56       35     72         255
19     Raticate  55      81       60     97         127
```

In [14]:

```
def result_pic(result):
    labels = ['HP', 'Attack', 'Defense', 'Speed', 'Catch_Rate']
    kinds = list(result.iloc[:, 0])

    result = pd.concat([result, result[['HP']]], axis=1)
    centers = np.array(result.iloc[:, 1:])

    n = len(labels)
    angle = np.linspace(0, 2 * np.pi, n, endpoint=False)
    angle = np.concatenate((angle, [angle[0]]))
    fig = plt.figure(figsize=(8, 8))
    ax = fig.add_subplot(111, polar=True)

    for i in range(len(kinds)):
        ax.plot(angle, centers[i], linewidth=2, label=kinds[i])
        ax.fill(angle, centers[i])

    ax.set_thetagrids(angle * 180 / np.pi, labels)
    plt.title('different Pokemon')
    plt.legend(loc='upper left')
    plt.show()


if __name__ == '__main__':
    result = result1
    result_pic(result)
```


different Pokemon
Attack
HP
Catch_Rate
Speed
Defense
50
100
150
200
250
Bulbasaur
Ivysaur
Venusaur
Charmander
Charmeleon
Charizard
Squirtle
Wartortle
Blastoise
Caterpie
Metapod
Butterfree
Weedle
Kakuna
Beedrill
Pidgey
Pidgeotto
Pidgeot
Rattata
Raticate

In [15]:

```
sns.pairplot(data=df.drop(['isLegendary', 'Generation', 'Number', 'Total', 'Color', 'hasGender', '
                            'Egg_Group_2', 'hasMegaEvolution', 'Height_m', 'Weight_kg', 'Body_Style
plt.show
```

```
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:140: RuntimeWarning: D
egrees of freedom <= 0 for slice
  keepdims=keepdims)
/usr/local/lib/python3.6/dist-packages/numpy/core/_methods.py:132: RuntimeWarning: i
nvalid value encountered in double_scalars
  ret = ret.dtype.type(ret / rcount)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kde.py:488: Runtime
Warning: invalid value encountered in true_divide
  binned = fast_linbin(X, a, b, gridsize) / (delta * nobs)
/usr/local/lib/python3.6/dist-packages/statsmodels/nonparametric/kdetools.py:34: Run
timeWarning: invalid value encountered in double_scalars
  FAC1 = 2*(np.pi*bw/RANGE)**2
```

Out[15]:

<function matplotlib.pyplot.show>

In [16]:

```
df.describe()
```

Out[16]:

	Number	Total	HP	Attack	Defense	Sp_Atk	Sp_Def	
count	721.00000	721.000000	721.000000	721.000000	721.000000	721.000000	721.000000	721
mean	361.00000	417.945908	68.380028	75.013870	70.808599	68.737864	69.291262	65
std	208.27906	109.663671	25.848272	28.984475	29.296558	28.788005	27.015860	27
min	1.00000	180.000000	1.000000	5.000000	5.000000	10.000000	20.000000	5
25%	181.00000	320.000000	50.000000	53.000000	50.000000	45.000000	50.000000	45
50%	361.00000	424.000000	65.000000	74.000000	65.000000	65.000000	65.000000	65
75%	541.00000	499.000000	80.000000	95.000000	85.000000	90.000000	85.000000	85
max	721.00000	720.000000	255.000000	165.000000	230.000000	154.000000	230.000000	160

In [0]:

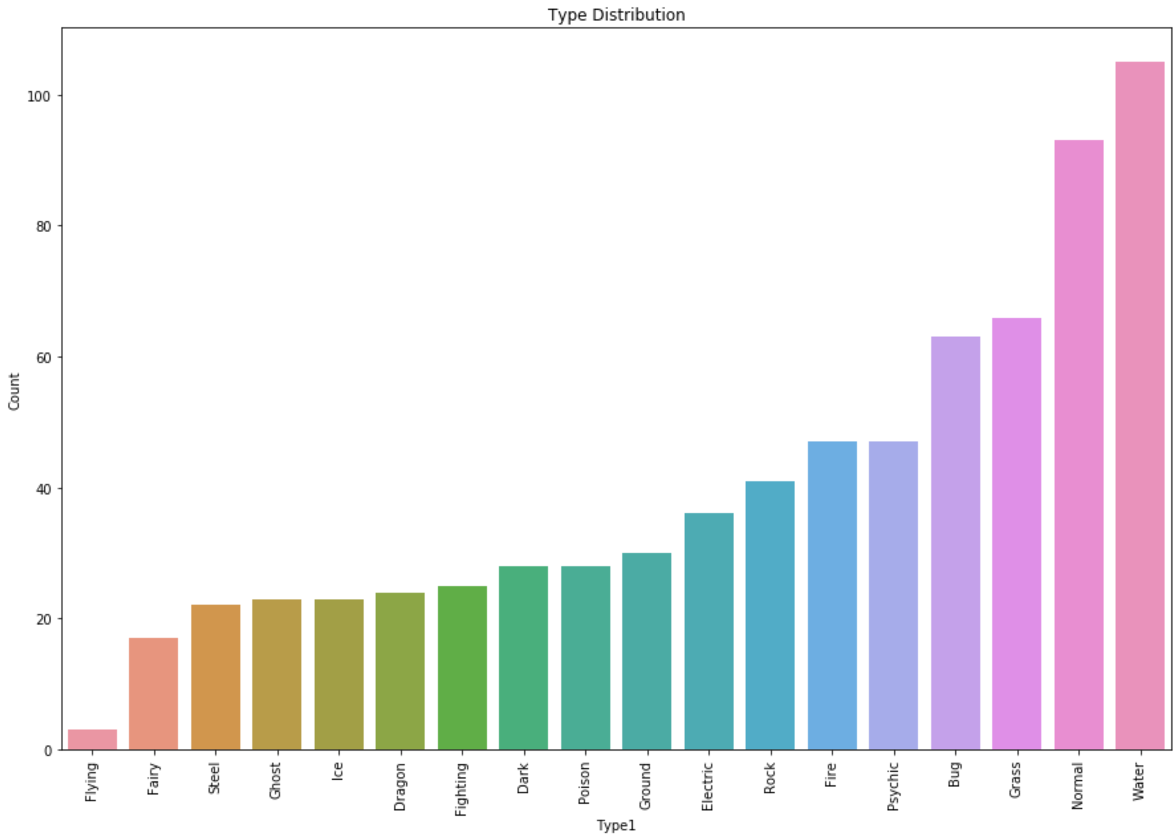
```
df1 = df.groupby('Type_1')['Type_1'].count().reset_index(name='Count')
df1 = df1.sort_values(by='Count')
```

In [18]:

```
plt.figure(figsize=(15, 10))
sns.barplot(x=df1['Type_1'], y=df1['Count'])
plt.xticks(rotation=90)
plt.xlabel('Type1')
plt.ylabel('Count')
plt.title('Type Distribution')
```

Out[18]:

```
Text(0.5, 1.0, 'Type Distribution')
```



In [0]:

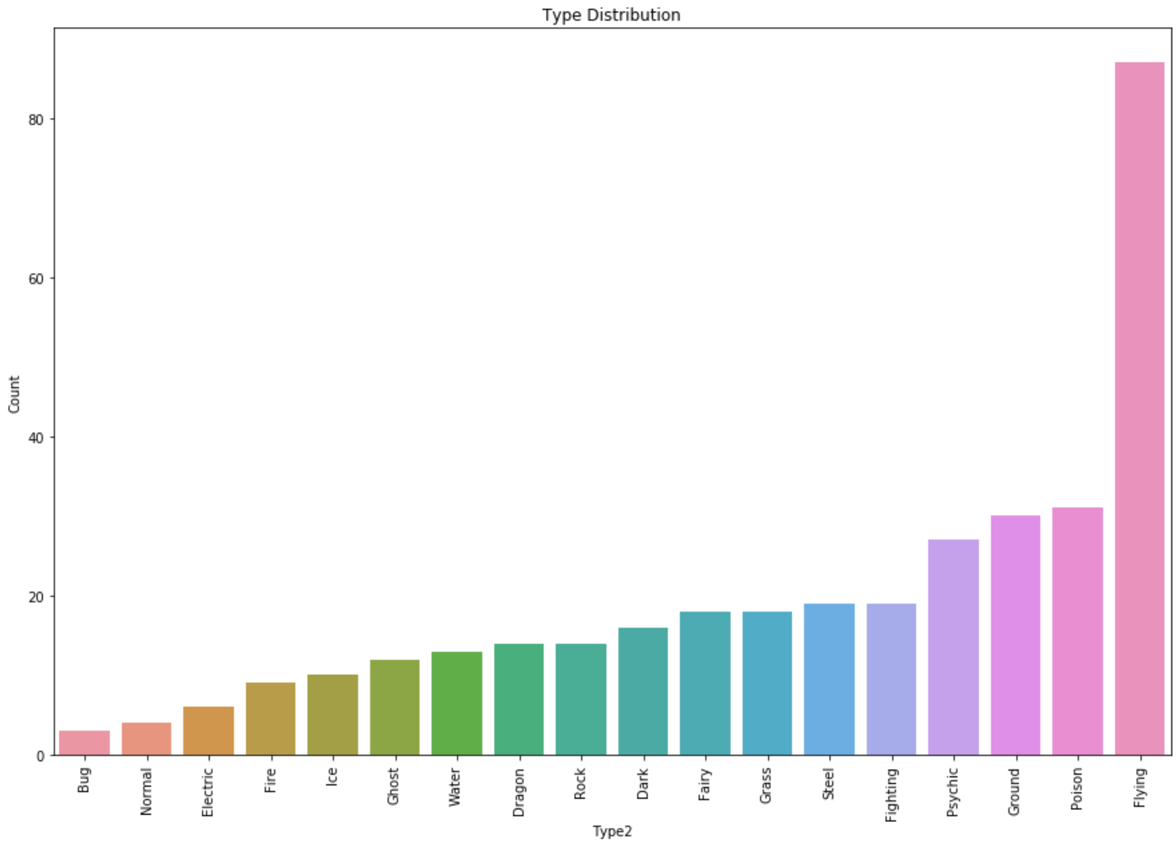
```
df2 = df.groupby('Type_2')['Type_2'].count().reset_index(name='Count')
df2 = df2.sort_values(by='Count')
```

In [20]:

```
plt.figure(figsize=(15, 10))
sns.barplot(x=df2['Type_2'], y=df2['Count'])
plt.xticks(rotation=90)
plt.xlabel('Type2')
plt.ylabel('Count')
plt.title('Type Distribution')
```

Out[20]:

```
Text(0.5, 1.0, 'Type Distribution')
```

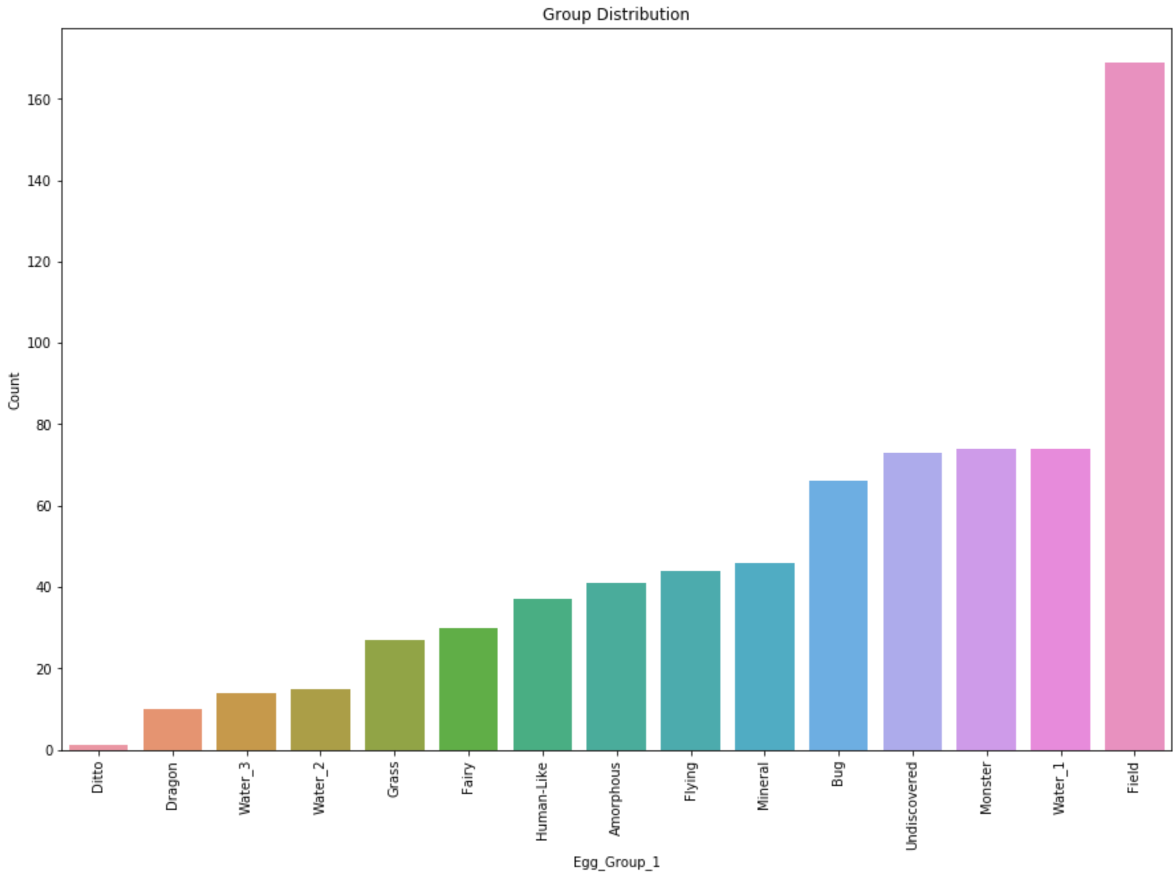


In [21]:

```
df3 = df.groupby('Egg_Group_1')['Egg_Group_1'].count().reset_index(name='Count')
df3 = df3.sort_values(by='Count')
plt.figure(figsize=(15, 10))
sns.barplot(x=df3['Egg_Group_1'], y=df3['Count'])
plt.xticks(rotation=90)
plt.xlabel('Egg_Group_1')
plt.ylabel('Count')
plt.title('Group Distribution')
```

Out[21]:

```
Text(0.5, 1.0, 'Group Distribution')
```

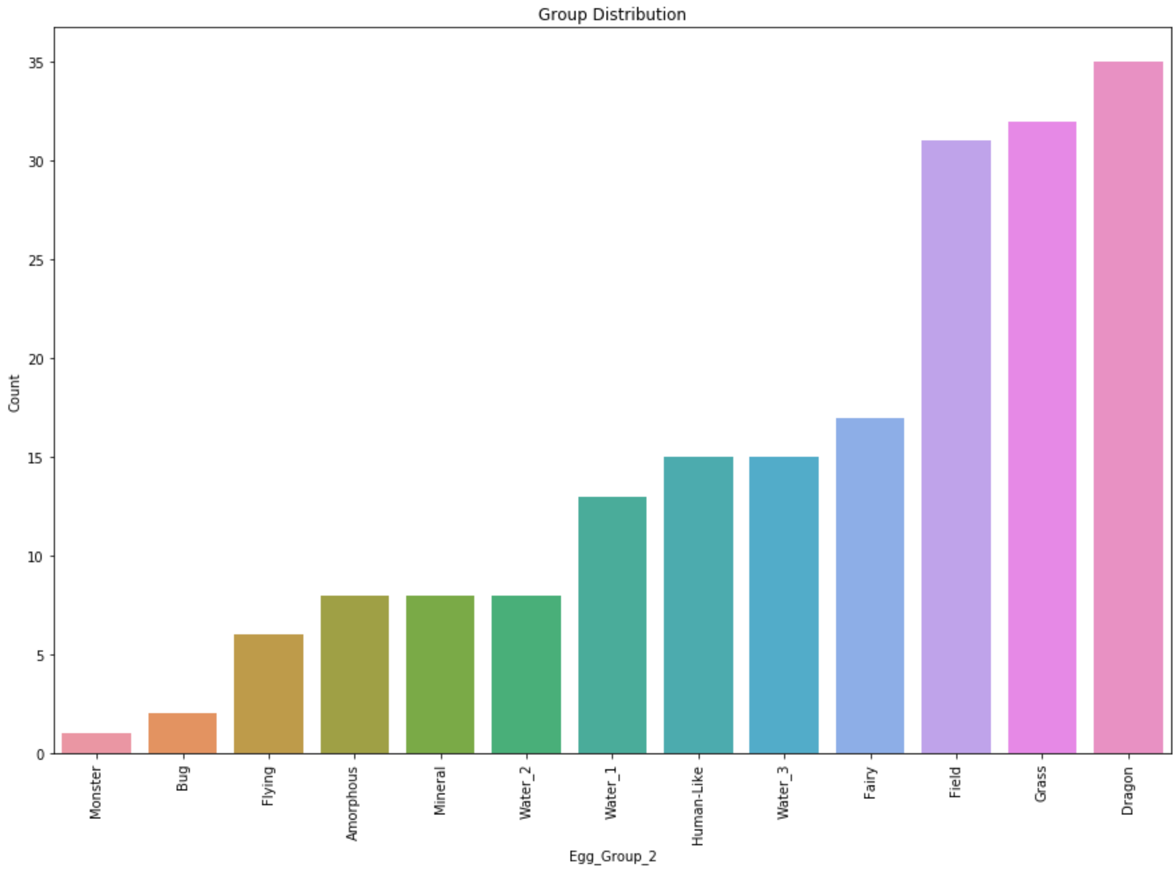


In [22]:

```
df4 = df.groupby('Egg_Group_2')['Egg_Group_2'].count().reset_index(name='Count')
df4 = df4.sort_values(by='Count')
plt.figure(figsize=(15, 10))
sns.barplot(x=df4['Egg_Group_2'], y=df4['Count'])
plt.xticks(rotation=90)
plt.xlabel('Egg_Group_2')
plt.ylabel('Count')
plt.title('Group Distribution')
```

Out[22]:

```
Text(0.5, 1.0, 'Group Distribution')
```



In [0]:

```
import sys
import os

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
import seaborn as sns

import sklearn
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import r2_score
from sklearn.model_selection import train_test_split, KFold
from sklearn.decomposition import PCA

import torch
import torch.nn.functional as F
import torch.nn as nn
import torch.utils.data as utils
```

In [0]:

```
df = pd.read_csv("pokemon_alopez247.csv", sep=",")
```

In [25]:

```
print(df.head(6))
print(df.info())
```

```
   Number        Name Type_1  ... Weight_kg  Catch_Rate     Body_Style
0       1   Bulbasaur  Grass  ...       6.9          45      quadruped
1       2     Ivysaur  Grass  ...      13.0          45      quadruped
2       3    Venusaur  Grass  ...     100.0          45      quadruped
3       4  Charmander   Fire  ...       8.5          45  bipedal_tailed
4       5  Charmeleon   Fire  ...      19.0          45  bipedal_tailed
5       6   Charizard   Fire  ...      90.5          45  bipedal_tailed

[6 rows x 23 columns]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 721 entries, 0 to 720
Data columns (total 23 columns):
Number              721 non-null int64
Name                721 non-null object
Type_1              721 non-null object
Type_2              350 non-null object
Total               721 non-null int64
HP                  721 non-null int64
Attack              721 non-null int64
Defense             721 non-null int64
Sp_Atk              721 non-null int64
Sp_Def              721 non-null int64
Speed               721 non-null int64
Generation          721 non-null int64
isLegendary         721 non-null bool
Color               721 non-null object
hasGender           721 non-null bool
Pr_Male             644 non-null float64
Egg_Group_1         721 non-null object
Egg_Group_2         191 non-null object
hasMegaEvolution    721 non-null bool
Height_m            721 non-null float64
Weight_kg           721 non-null float64
Catch_Rate          721 non-null int64
Body_Style          721 non-null object
dtypes: bool(3), float64(3), int64(10), object(7)
memory usage: 114.8+ KB
None
```

Prediction

1. Data preprocessed

Convert text labels to one hot code.

选择one hot的原因 [Reason (http://queirozf.com/entries/one-hot-encoding-a-feature-on-a-pandas-dataframe-an-example)](http://queirozf.com/entries/one-hot-encoding-a-feature-on-a-pandas-dataframe-an-example)

In [0]:

```
df1 = df.copy()

df1 = pd.get_dummies(data=df1, columns=['Type_1', 'Type_2'], prefix='Type')
df1 = pd.get_dummies(data=df1, columns=['Color'], prefix='Color')
df1 = pd.get_dummies(data=df1, columns=['Egg_Group_1', 'Egg_Group_2'], prefix='Egg_Group')
df1 = pd.get_dummies(data=df1, columns=['Body_Style'], prefix='Body_Style')
df1 = df1.fillna(0)
```

Generate X and y from the dataset.

In [27]:

```
# X = df1.loc[:, df1.columns != 'Name']
X = df1.drop(['Name', 'Number', 'Catch_Rate'], axis=1).to_numpy().astype(float)
labels = df1.drop(['Name', 'Number', 'Catch_Rate'], axis=1).columns
y = df1.Catch_Rate.to_numpy() / 255
n_sample = X.shape[0]
n_channel = X.shape[1]
print(f'Number of samples: {n_sample}\nNumber of features: {n_channel}')
```

```
Number of samples: 721
Number of features: 102
```

2. Linear prediction

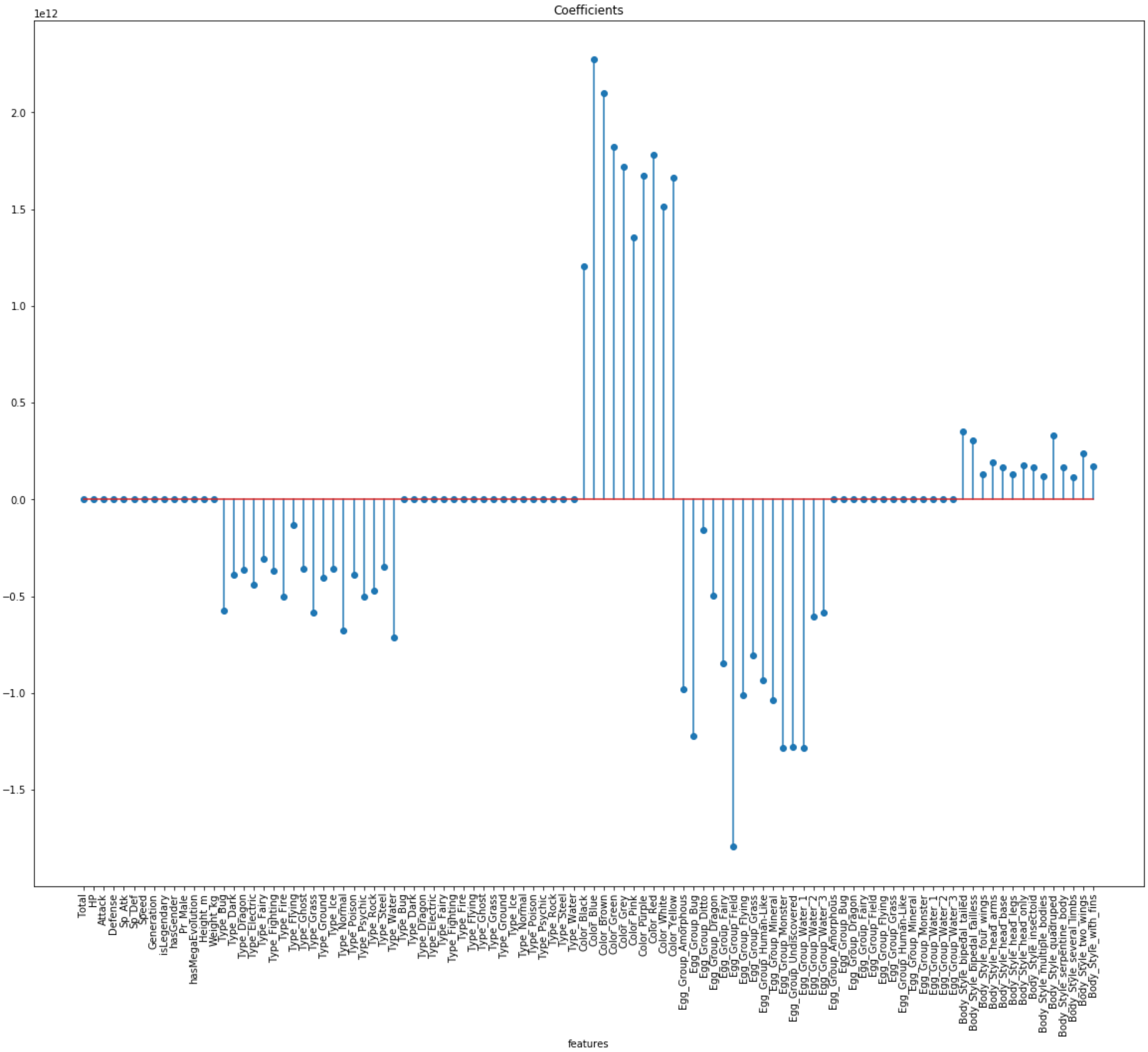
a. Basic linear regression

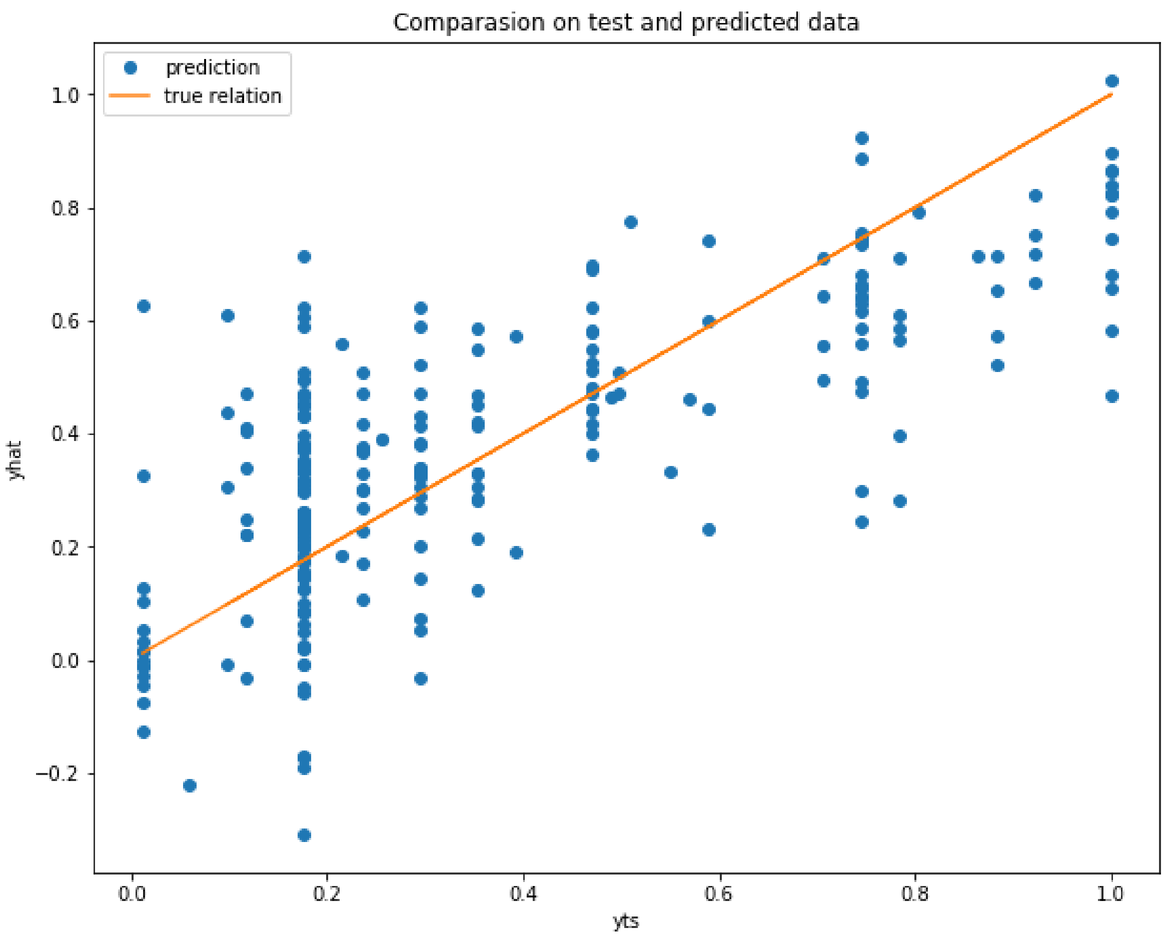
First do the normalization. Then do the linear regression.

In [28]:

```
scaling = StandardScaler()
scaling.fit(X)
X_n = scaling.transform(X)


def score(model, x, y):
    return 1 - np.mean((model.predict(x) - y)**2)


n_run = 5
train_scores = []
test_scores = []
coeffs = []
for i in range(n_run):
    Xtr, Xts, ytr, yts = train_test_split(X_n, y, test_size=0.33, shuffle=True)
    regr = LinearRegression()
    regr.fit(Xtr, ytr)
    yhat = regr.predict(Xts)
    train_scores.append(score(regr, Xtr, ytr))
    test_scores.append(score(regr, Xts, yts))
    coeffs.append(regr)

# print(test_scores)
print(f"Train score is {np.max(train_scores)}, test score is {np.max(test_scores)}.")
best = np.argmax(test_scores)
best_coef = coeffs[best].coef_
# print(best_coef)
plt.figure(figsize=(20, 16))
plt.stem(best_coef)
plt.xlabel('features')
# plt.ylabel('$log(coef)$')
plt.title('Coefficients')
plt.xticks(range(n_channel), labels, rotation=90)
plt.show()

plt.figure(figsize=(10, 8))
plt.plot(yts, coeffs[best].predict(Xts), 'o')
plt.plot(yts,yts)
plt.xlabel('yts')
plt.ylabel('yhat')
plt.title('Comparasion on test and predicted data')
plt.legend(['prediction', 'true relation'])
plt.show()
```

```
Train score is 0.974636388478952, test score is 0.9616833020611844.
```


Coefficients
1e12
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Total
HP
Attack
Defense
Sp_Atk
Sp_Def
Speed
Generation
isLegendary
hasGender
Pr_Male
hasMegaEvolution
Height_m
Weight_kg
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Color_Black
Color_Blue
Color_Brown
Color_Green
Color_Grey
Color_Pink
Color_Purple
Color_Red
Color_White
Color_Yellow
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Ditto
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Undiscovered
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Body_Style_bipedal_tailed
Body_Style_bipedal_tailless
Body_Style_four_wings
Body_Style_head_arms
Body_Style_head_base
Body_Style_head_legs
Body_Style_head_only
Body_Style_insectoid
Body_Style_multiple_bodies
Body_Style_quadruped
Body_Style_serpentine_body
Body_Style_several_limbs
Body_Style_two_wings
Body_Style_with_fins
features

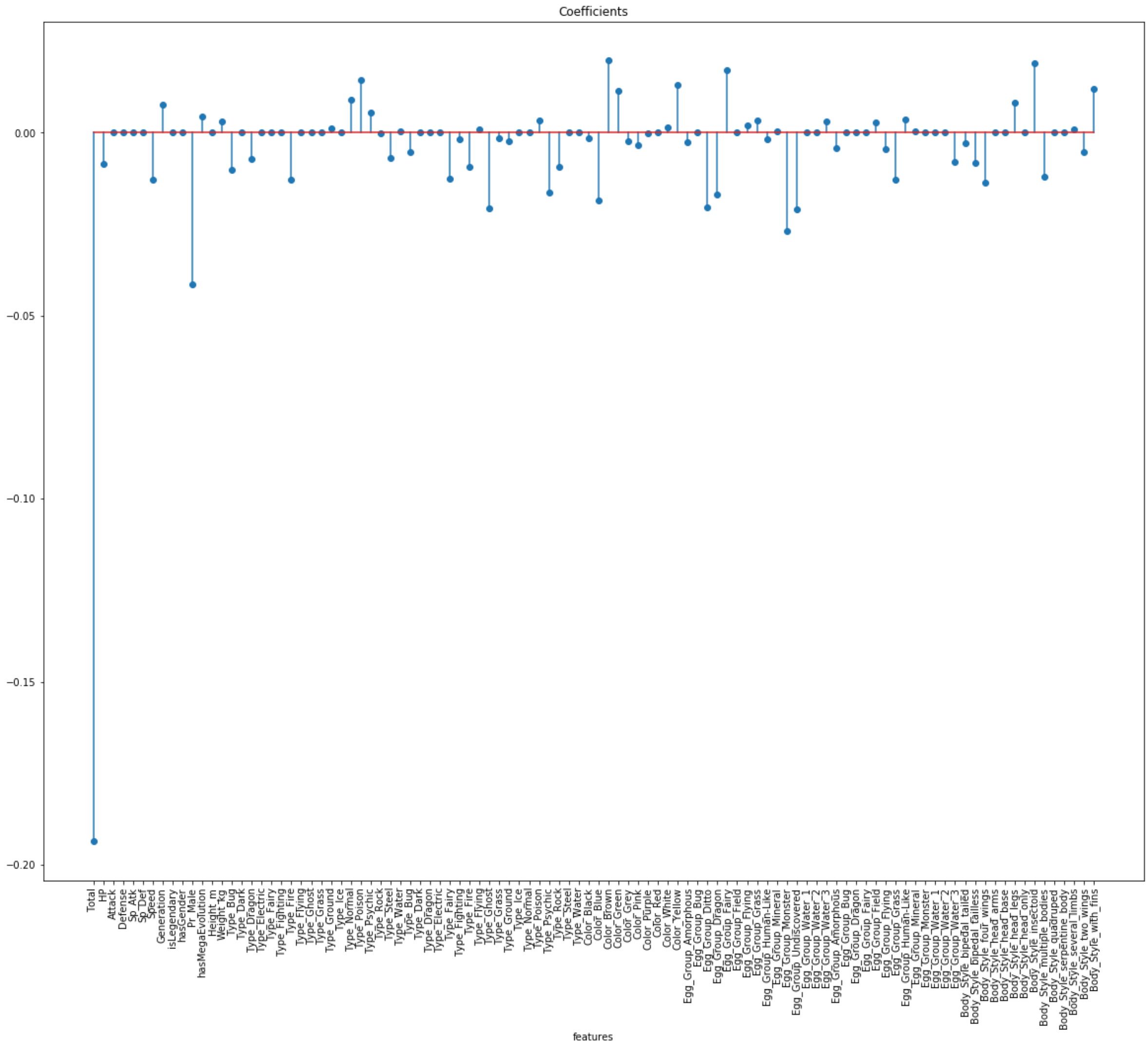

Comparasion on test and predicted data
prediction
true relation
yhat
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0
yts

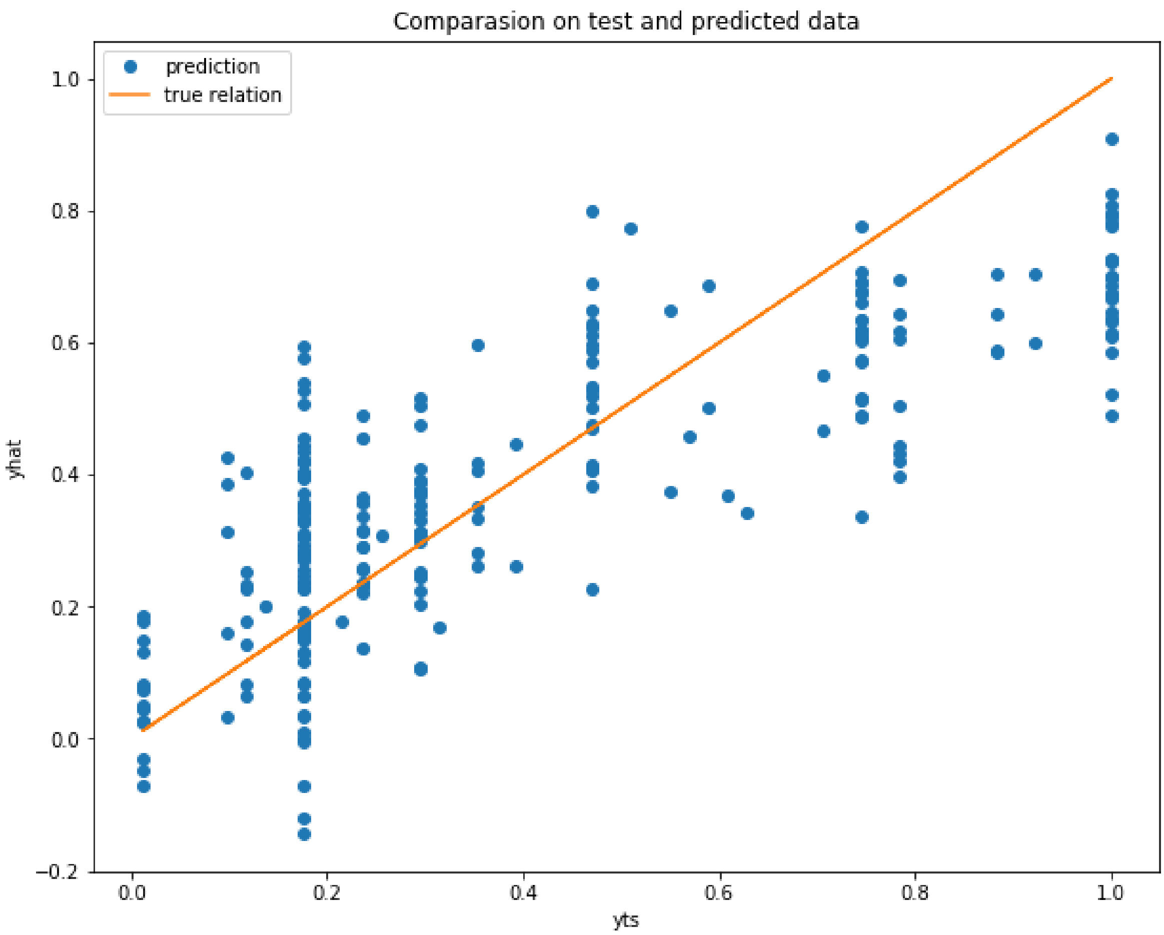
b. Linear regression with LASSO

In [29]:

```
nalpha = 10
alphas = np.logspace(-3, 2, nalpha)

train_scores = []
test_scores = []
coeffs = []

for alpha in alphas:
    for i in range(n_run):
        Xtr, Xts, ytr, yts = train_test_split(X_n, y, test_size=0.33, shuffle=True)
        #     print(f"calculating alpha for {alpha}")
        regr = Lasso(alpha=alpha)
        regr.fit(Xtr, ytr)
        #     print(f"Done {alpha}")
        yts_pred = regr.predict(Xts)
        train_scores.append(score(regr, Xtr, ytr))
        test_scores.append(score(regr, Xts, yts))
        coeffs.append(regr)

print(f"Best train score is {np.max(train_scores)}, test  score is {np.max(test_scores)}.")
best = np.argmax(test_scores)
best_coef = coeffs[best].coef_
# print(best_coef)
prominent_labels = labels[np.abs(best_coef).argsort()[-6:]][::-1]
print(f"Six most prominent features are {', '.join(prominent_labels)}")
plt.figure(figsize=(20, 16))
plt.stem(best_coef)
plt.xlabel('features')
# plt.ylabel('$log(coef)$')
plt.title('Coefficients')
plt.xticks(range(n_channel), labels, rotation=90)
plt.show()

plt.figure(figsize=(10, 8))
plt.plot(yts, coeffs[best].predict(Xts), 'o')
plt.plot(yts,yts)
plt.xlabel('yts')
plt.ylabel('yhat')
plt.title('Comparasion on test and predicted data')
plt.legend(['prediction', 'true relation'])
plt.show()
```

```
Best train score is 0.9732474155729648, test  score is 0.9677208499056017.
Six most prominent features are Total, Pr_Male, Egg_Group_Monster, Egg_Group_Undisco
vered, Type_Ghost, Egg_Group_Ditto
```


Coefficients
0.00
−0.05
−0.10
−0.15
−0.20
Total
HP
Attack
Defense
Sp_Atk
Sp_Def
Speed
Generation
isLegendary
hasGender
Pr_Male
hasMegaEvolution
Height_m
Weight_kg
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Color_Black
Color_Blue
Color_Brown
Color_Green
Color_Grey
Color_Pink
Color_Purple
Color_Red
Color_White
Color_Yellow
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Ditto
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Undiscovered
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Body_Style_bipedal_tailed
Body_Style_bipedal_tailless
Body_Style_four_wings
Body_Style_head_arms
Body_Style_head_base
Body_Style_head_legs
Body_Style_head_only
Body_Style_insectoid
Body_Style_multiple_bodies
Body_Style_quadruped
Body_Style_serpentine_body
Body_Style_several_limbs
Body_Style_two_wings
Body_Style_with_fins
features

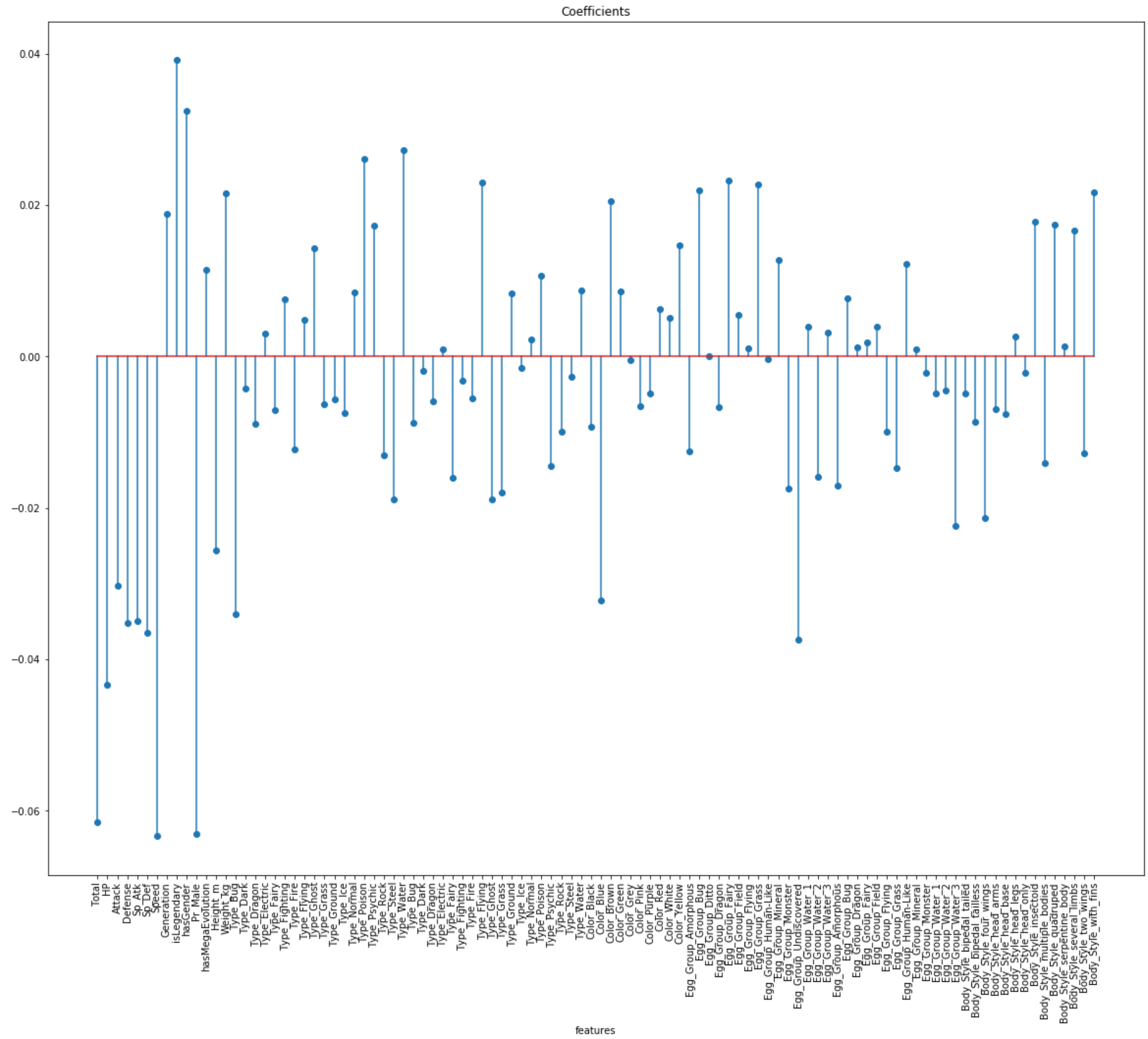

Comparasion on test and predicted data
prediction
true relation
yhat
yts
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
0.0
0.2
0.4
0.6
0.8
1.0

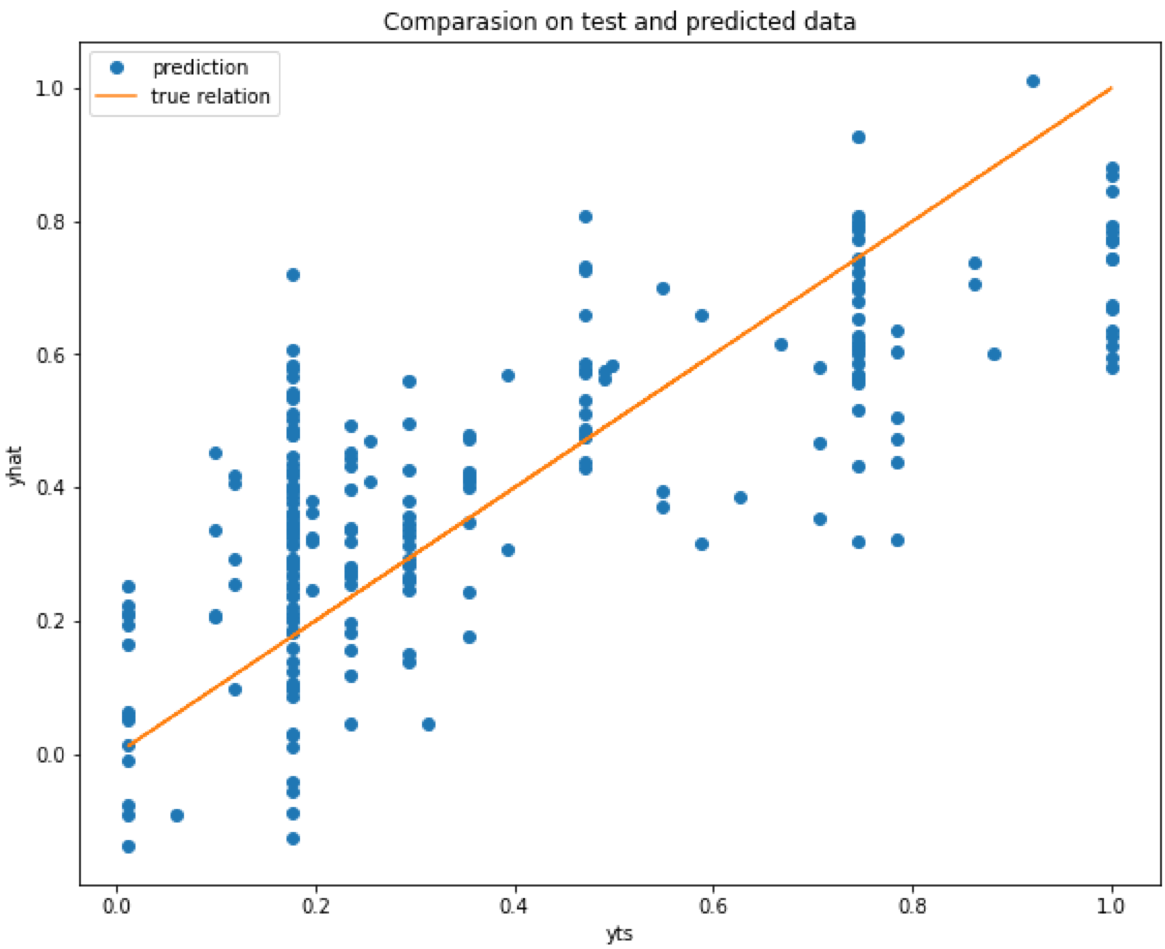
c. Linear regression with Ridge

In [30]:

```
nalpha = 20
alphas = np.logspace(-2, 2, nalpha)

train_scores = []
test_scores = []
coeffs = []

for alpha in alphas:
    for i in range(n_run):
        Xtr, Xts, ytr, yts = train_test_split(X_n, y, test_size=0.33, shuffle=True)
        regr = Ridge(alpha=alpha)
        regr.fit(Xtr, ytr)
        train_scores.append(score(regr, Xtr, ytr))
        test_scores.append(score(regr, Xts, yts))
        coeffs.append(regr)

print(f"Best train score is {np.max(train_scores)}, test score is {np.max(test_scores)}.")
best = np.argmax(test_scores)
best_coef = coeffs[best].coef_
# print(best_coef)
prominent_labels = labels[np.abs(best_coef).argsort()[-6:]][::-1]
print(f"Six most prominent features are {', '.join(prominent_labels)}")
plt.figure(figsize=(20, 16))
plt.stem(best_coef)
plt.xlabel('features')
# plt.ylabel('$log(coef)$')
plt.title('Coefficients')
plt.xticks(range(n_channel), labels, rotation=90)
plt.show()

plt.figure(figsize=(10, 8))
plt.plot(yts, coeffs[best].predict(Xts), 'o')
plt.plot(yts,yts)
plt.xlabel('yts')
plt.ylabel('yhat')
plt.title('Comparasion on test and predicted data')
plt.legend(['prediction', 'true relation'])
plt.show()
```

```
Best train score is 0.9758948312606975, test score is 0.9668559838842328.
Six most prominent features are Speed, Pr_Male, Total, HP, isLegendary, Egg_Group_Un
discovered
```


Coefficients
0.04
0.02
0.00
−0.02
−0.04
−0.06
Total
HP
Attack
Defense
Sp_Atk
Sp_Def
Speed
Generation
isLegendary
hasGender
Pr_Male
hasMegaEvolution
Height_m
Weight_kg
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Type_Bug
Type_Dark
Type_Dragon
Type_Electric
Type_Fairy
Type_Fighting
Type_Fire
Type_Flying
Type_Ghost
Type_Grass
Type_Ground
Type_Ice
Type_Normal
Type_Poison
Type_Psychic
Type_Rock
Type_Steel
Type_Water
Color_Black
Color_Blue
Color_Brown
Color_Green
Color_Grey
Color_Pink
Color_Purple
Color_Red
Color_White
Color_Yellow
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Ditto
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Undiscovered
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Egg_Group_Amorphous
Egg_Group_Bug
Egg_Group_Dragon
Egg_Group_Fairy
Egg_Group_Field
Egg_Group_Flying
Egg_Group_Grass
Egg_Group_Human-Like
Egg_Group_Mineral
Egg_Group_Monster
Egg_Group_Water_1
Egg_Group_Water_2
Egg_Group_Water_3
Body_Style_bipedal_tailed
Body_Style_bipedal_tailless
Body_Style_four_wings
Body_Style_head_arms
Body_Style_head_base
Body_Style_head_legs
Body_Style_head_only
Body_Style_insectoid
Body_Style_multiple_bodies
Body_Style_quadruped
Body_Style_serpentine_body
Body_Style_several_limbs
Body_Style_two_wings
Body_Style_with_fins
features


Comparasion on test and predicted data
prediction
true relation
yhat
1.0
0.8
0.6
0.4
0.2
0.0
yts

d. Linear regression with PCA

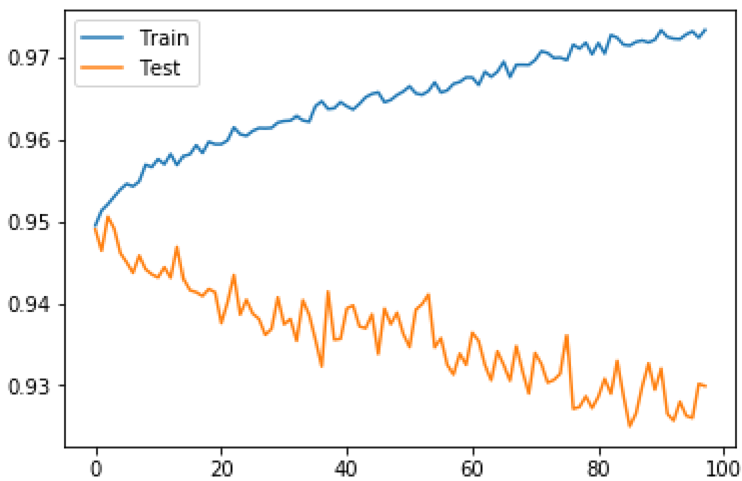
In [0]:

```
n_run = 10

# Number of PCs to try
ncomp_test = np.arange(2, 100)
num_nc = len(ncomp_test)

train_scores = []
test_scores = []
coeffs = []

for ncomp in ncomp_test:
    for i in range(n_run):
        Xtr, Xts, ytr, yts = train_test_split(X_n, y, test_size=0.33, shuffle=True)
        pca = PCA(n_components=ncomp, svd_solver='randomized', whiten=True)
        Xtr_transform = pca.fit_transform(Xtr)

        regr = LinearRegression()
        regr.fit(Xtr_transform, ytr)
        Xts_transform = pca.fit_transform(Xts)

        train_scores.append(score(regr, Xtr_transform, ytr))
        test_scores.append(score(regr, Xts_transform, yts))
        coeffs.append(regr.coef_)
```

In [32]:

```
# best = np.argmax(acc)
print(f"Best train  score is {np.max(train_scores)}, test score is {np.max(test_scores)}.")
# print(train_scores)
# print(test_scores)
train_scores = np.array(train_scores).reshape(num_nc, -1)
test_scores = np.array(test_scores).reshape(num_nc, -1)

# keep biggest 6 and compute their mean
train_mean = np.sort(train_scores)[:, :4:-1].mean(axis=1)
test_mean = np.sort(test_scores)[:, :4:-1].mean(axis=1)
plt.plot(train_mean)
plt.plot(test_mean)
plt.legend(['Train', 'Test'])
plt.show()
```

```
Best train  score is 0.9749101980907205, test score is 0.9535702528151085.
```



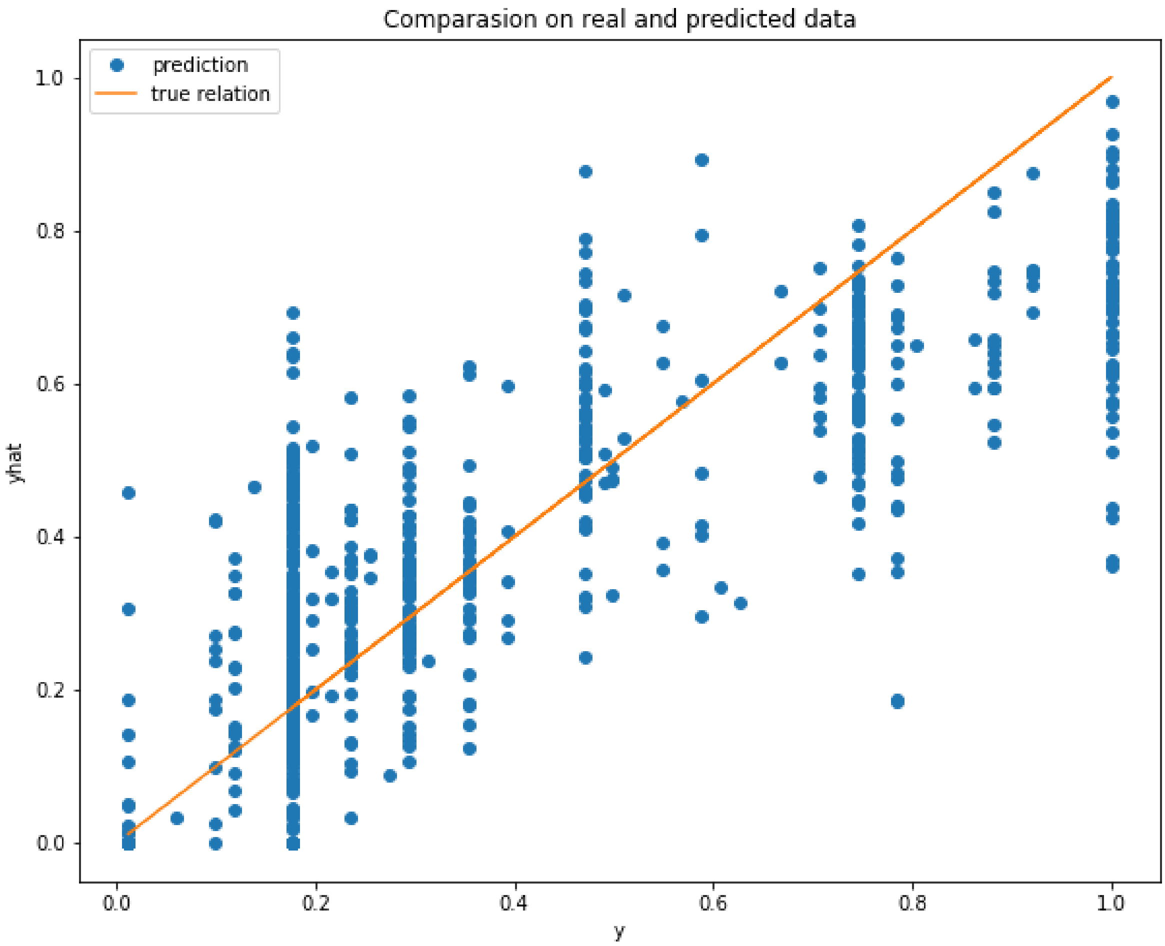
e. With neural network

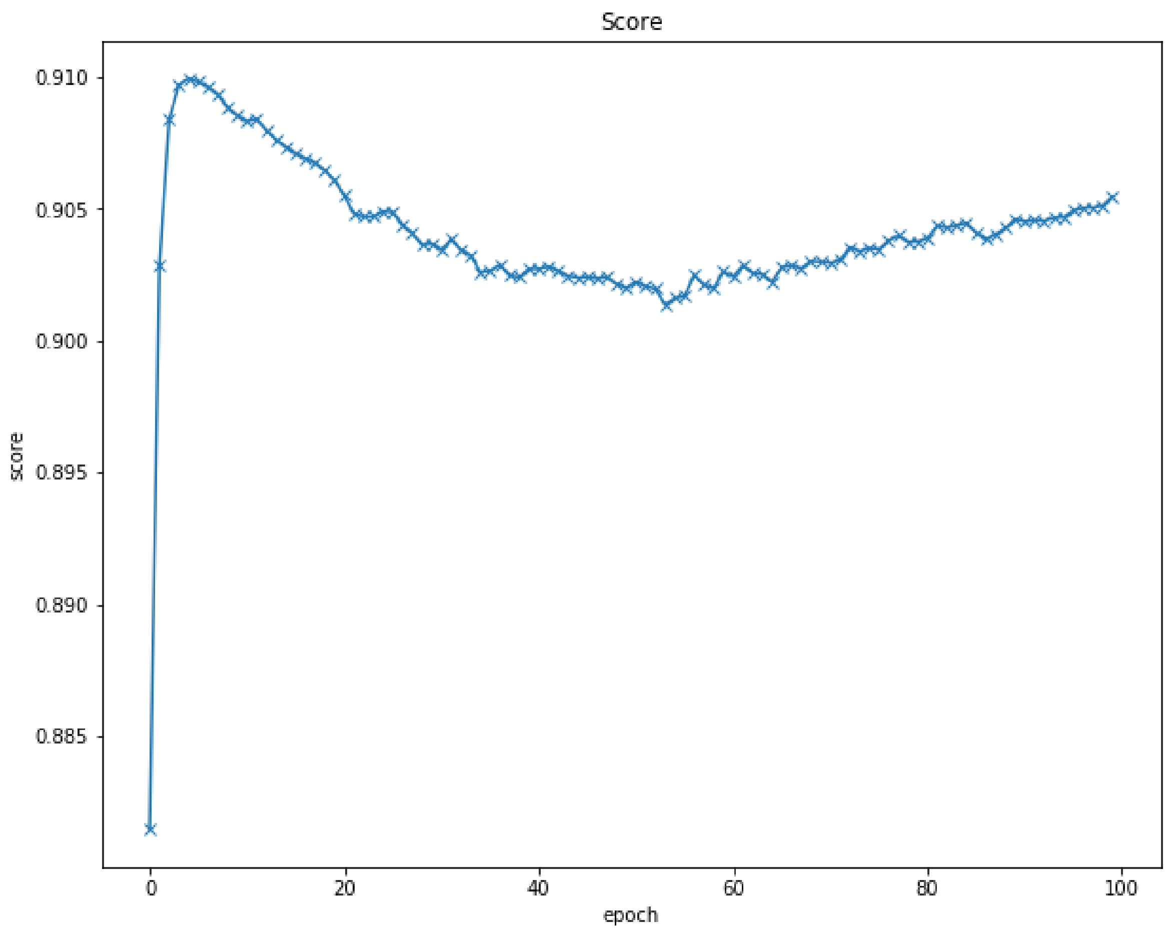
In [33]:

```
class Net(nn.Module):
    def __init__(self, n):
        super(Net, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(n, 1000),
            nn.ReLU(),
            nn.Linear(1000,1000),
            nn.ReLU(),
            nn.Linear(1000,1),
            nn.ReLU()
        )


    def forward(self, x):
        return self.net(x)


lr = 0.001

epochs = 100
batch_size = 50
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

dataset = utils.TensorDataset(torch.Tensor(X_n), torch.Tensor(y))
dataloader = utils.DataLoader(dataset, batch_size=50, shuffle=True)

print(f"Running on device {device}")

# Use GPU
model = Net(X_n.shape[1]).to(device)
optimizer = torch.optim.SGD(model.parameters(), lr=lr)
criterion = nn.MSELoss()
train_scores = []
losses = []

for epoch in range(1, epochs + 1):
    epoch_loss = 0
    score = 0
    model = model.train()
    for index, data in enumerate(dataloader):
        x_, y_ = data
        x_, y_ = x_.to(device), y_.to(device)
        pred = model(x_).view(-1)
#         print(y_.shape, pred.shape)
        loss = criterion(pred, y_)
#         loss = torch.mean((pred - y_)**2)
        epoch_loss += loss.item()

        # print(f"{index*batch_size:.4f} --- loss: {loss.item()/batch_size:.6f}")
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
    with torch.no_grad():
        X_val = torch.FloatTensor(X_n).to(device)
        result = model(X_val)
        pred = result.cpu().numpy()
        # score = r2_score(y, pred)
```

```
        score = 1 - np.mean((y - pred[0, :])**2)
    train_scores.append(score)
    losses.append(epoch_loss / len(dataset))
#     print(f'Epoch {epoch} finished! Loss: {epoch_loss / len(dataset):.4f} Score: {score:.6f}')
    torch.save(model.state_dict(), 'saved.model')


with torch.no_grad():
      X_val = torch.FloatTensor(X_n).to(device)
      result = model(X_val)
      pred = result.cpu().numpy()
      plt.figure(figsize=(10, 8))
      plt.plot(y, pred, 'o')
      plt.plot(y,y)
      plt.xlabel('y')
      plt.ylabel('yhat')
      plt.title('Comparasion on real and predicted data')
      plt.legend(['prediction', 'true relation'])
      plt.show()

plt.figure(figsize=(10, 8))
plt.plot(train_scores,'x-')
# plt.plot(losses)
plt.xlabel('epoch')
plt.ylabel('score')
plt.title('Score')
# plt.legend(['Score', 'Loss'])

plt.show()
```

```
Running on device cuda:0
```


Comparasion on real and predicted data
prediction
true relation
yhat
y
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0


Score
score
epoch
0.910
0.905
0.900
0.895
0.890
0.885
0
20
40
60
80
100