COMPARING REGRESSION MODELS

Hello everyone.This is a notebook comparing various regression models such as Ridge,Knn,Bayesian Regression,Decision Tree and SVM. *It is extremely beneficial for beginners to take a close look at the notebook so as to get an insight as to how different algorithms work and also which algorithms can perform better in some cases depending upon cases*

[1]:
```
# This Python 3 environment comes with many helpful analytics libraries␣
↪installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/
↪docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list␣
↪the files in the input directory

from subprocess import check_output
print(check_output(["ls", "../input"]).decode("utf8"))

# Any results you write to the current directory are saved as output.
```

```
movie_metadata.csv
```

[2]:
```
# Importing packages

import os
import pandas as pd
from pandas import DataFrame,Series
from sklearn import tree
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.preprocessing import StandardScaler
import statsmodels.formula.api as smf
```

```
import statsmodels.api as sm
from mpl_toolkits.mplot3d import Axes3D
import seaborn as sns
from sklearn import neighbors
from sklearn import linear_model
%matplotlib inline
```

[3]:
```
f = pd.read_csv("../input/movie_metadata.csv")
```

[4]:
```
data=DataFrame(f)
data.head()[:2]
```

[4]:
```
   color   director_name  num_critic_for_reviews  duration  \
0  Color   James Cameron                   723.0     178.0
1  Color  Gore Verbinski                   302.0     169.0

   director_facebook_likes  actor_3_facebook_likes      actor_2_name  \
0                      0.0                   855.0  Joel David Moore
1                    563.0                  1000.0     Orlando Bloom

   actor_1_facebook_likes        gross                           genres  \
0                  1000.0  760505847.0  Action|Adventure|Fantasy|Sci-Fi
1                 40000.0  309404152.0         Action|Adventure|Fantasy

          …           num_user_for_reviews language  country  content_rating  \
0         …                         3054.0  English      USA           PG-13
1         …                         1238.0  English      USA           PG-13

        budget  title_year actor_2_facebook_likes imdb_score  aspect_ratio  \
0  237000000.0      2009.0                  936.0        7.9          1.78
1  300000000.0      2007.0                 5000.0        7.1          2.35

  movie_facebook_likes
0                33000
1                    0

[2 rows x 28 columns]
```

Getting non-object elements

[5]:
```
X_data=data.dtypes[data.dtypes!='object'].index
X_train=data[X_data]
X_train.head()[:2]
```

[5]:
```
   num_critic_for_reviews  duration  director_facebook_likes  \
0                   723.0     178.0                      0.0
1                   302.0     169.0                    563.0
```

```
   actor_3_facebook_likes  actor_1_facebook_likes        gross  \
0                   855.0                  1000.0  760505847.0
1                  1000.0                 40000.0  309404152.0

   num_voted_users  cast_total_facebook_likes  facenumber_in_poster  \
0           886204                       4834                   0.0
1           471220                      48350                   0.0

   num_user_for_reviews       budget  title_year  actor_2_facebook_likes  \
0                3054.0  237000000.0      2009.0                   936.0
1                1238.0  300000000.0      2007.0                  5000.0

   imdb_score  aspect_ratio  movie_facebook_likes
0         7.9          1.78                 33000
1         7.1          2.35                     0
```

[6]:
```
X_train.describe()
```

[6]:
```
       num_critic_for_reviews     duration  director_facebook_likes  \
count             4993.000000  5028.000000              4939.000000
mean               140.194272   107.201074               686.509212
std                121.601675    25.197441              2813.328607
min                  1.000000     7.000000                 0.000000
25%                 50.000000    93.000000                 7.000000
50%                110.000000   103.000000                49.000000
75%                195.000000   118.000000               194.500000
max                813.000000   511.000000             23000.000000

       actor_3_facebook_likes  actor_1_facebook_likes         gross  \
count             5020.000000             5036.000000  4.159000e+03
mean               645.009761             6560.047061  4.846841e+07
std               1665.041728            15020.759120  6.845299e+07
min                  0.000000                0.000000  1.620000e+02
25%                133.000000              614.000000  5.340988e+06
50%                371.500000              988.000000  2.551750e+07
75%                636.000000            11000.000000  6.230944e+07
max              23000.000000           640000.000000  7.605058e+08

       num_voted_users  cast_total_facebook_likes  facenumber_in_poster  \
count     5.043000e+03                5043.000000           5030.000000
mean      8.366816e+04                9699.063851              1.371173
std       1.384853e+05               18163.799124              2.013576
min       5.000000e+00                   0.000000              0.000000
25%       8.593500e+03                1411.000000              0.000000
50%       3.435900e+04                3090.000000              1.000000
75%       9.630900e+04               13756.500000              2.000000
```

```
max       1.689764e+06             656730.000000             43.000000

       num_user_for_reviews        budget   title_year  \
count           5022.000000  4.551000e+03  4935.000000
mean             272.770808  3.975262e+07  2002.470517
std              377.982886  2.061149e+08    12.474599
min                1.000000  2.180000e+02  1916.000000
25%               65.000000  6.000000e+06  1999.000000
50%              156.000000  2.000000e+07  2005.000000
75%              326.000000  4.500000e+07  2011.000000
max             5060.000000  1.221550e+10  2016.000000

       actor_2_facebook_likes   imdb_score  aspect_ratio  movie_facebook_likes
count             5030.000000  5043.000000   4714.000000           5043.000000
mean              1651.754473     6.442138      2.220403           7525.964505
std               4042.438863     1.125116      1.385113          19320.445110
min                  0.000000     1.600000      1.180000              0.000000
25%                281.000000     5.800000      1.850000              0.000000
50%                595.000000     6.600000      2.350000            166.000000
75%                918.000000     7.200000      2.350000           3000.000000
max             137000.000000     9.500000     16.000000         349000.000000
```

[7]:
```
# Finding all the columns with NULL values

np.sum(X_train.isnull())
```

[7]:
```
num_critic_for_reviews        50
duration                      15
director_facebook_likes      104
actor_3_facebook_likes        23
actor_1_facebook_likes         7
gross                        884
num_voted_users                0
cast_total_facebook_likes      0
facenumber_in_poster          13
num_user_for_reviews          21
budget                       492
title_year                   108
actor_2_facebook_likes        13
imdb_score                     0
aspect_ratio                 329
movie_facebook_likes           0
dtype: int64
```

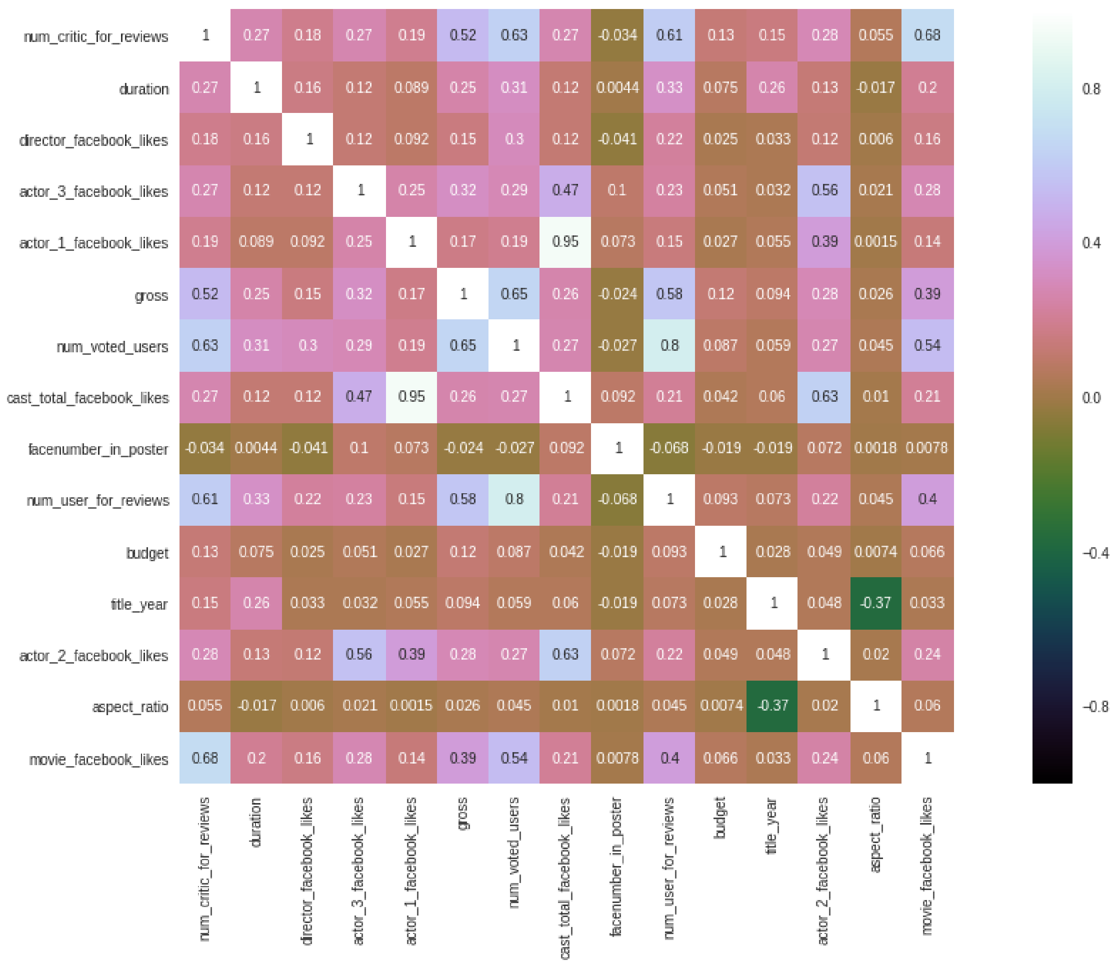
[8]:
```
# Filling all Null values
X_train=X_train.fillna(0)
columns=X_train.columns.tolist()
```

```
y=X_train['imdb_score']
X_train.drop(['imdb_score'],axis=1,inplace=True)
X_train.head()[:2]
```

[8]:
```
   num_critic_for_reviews  duration  director_facebook_likes  \
0                   723.0     178.0                      0.0
1                   302.0     169.0                    563.0

   actor_3_facebook_likes  actor_1_facebook_likes        gross  \
0                   855.0                  1000.0  760505847.0
1                  1000.0                 40000.0  309404152.0

   num_voted_users  cast_total_facebook_likes  facenumber_in_poster  \
0           886204                       4834                   0.0
1           471220                      48350                   0.0

   num_user_for_reviews       budget  title_year  actor_2_facebook_likes  \
0                3054.0  237000000.0      2009.0                   936.0
1                1238.0  300000000.0      2007.0                  5000.0

   aspect_ratio  movie_facebook_likes
0          1.78                 33000
1          2.35                     0
```

[9]:
```
# GETTING Correllation matrix
corr_mat=X_train.corr(method='pearson')
plt.figure(figsize=(20,10))
sns.heatmap(corr_mat,vmax=1,square=True,annot=True,cmap='cubehelix')
```

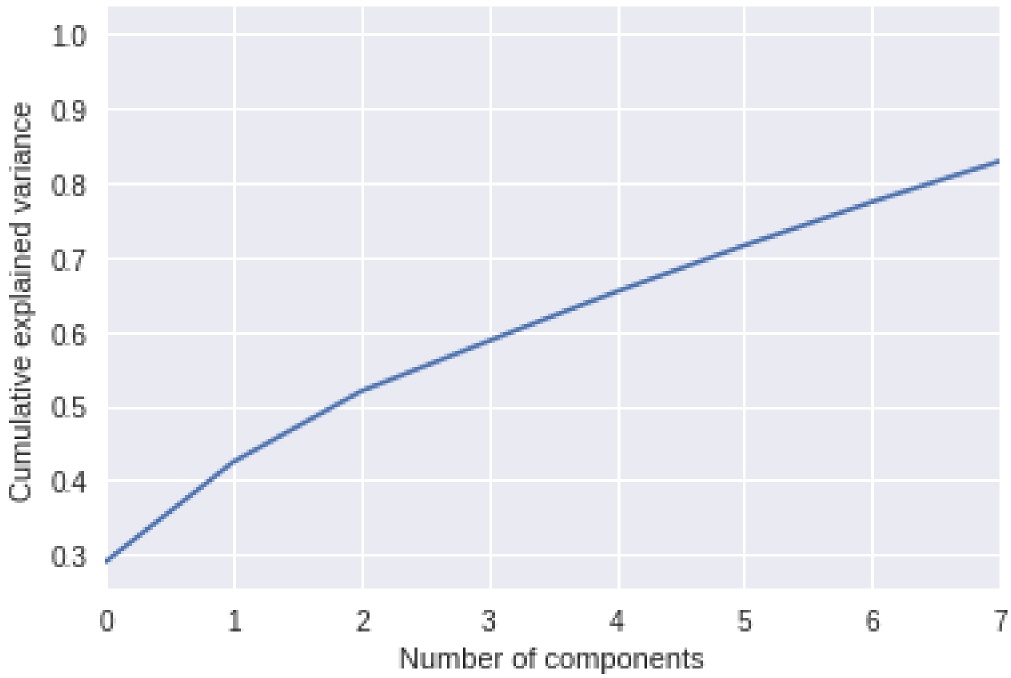
[9]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f3c0fa12cc0>
```



[10]:
```
X_Train=X_train.values
X_Train=np.asarray(X_Train)

# Finding normalised array of X_Train
X_std=StandardScaler().fit_transform(X_Train)
```

[11]:
```
from sklearn.decomposition import PCA
pca = PCA().fit(X_std)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,7,1)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
```

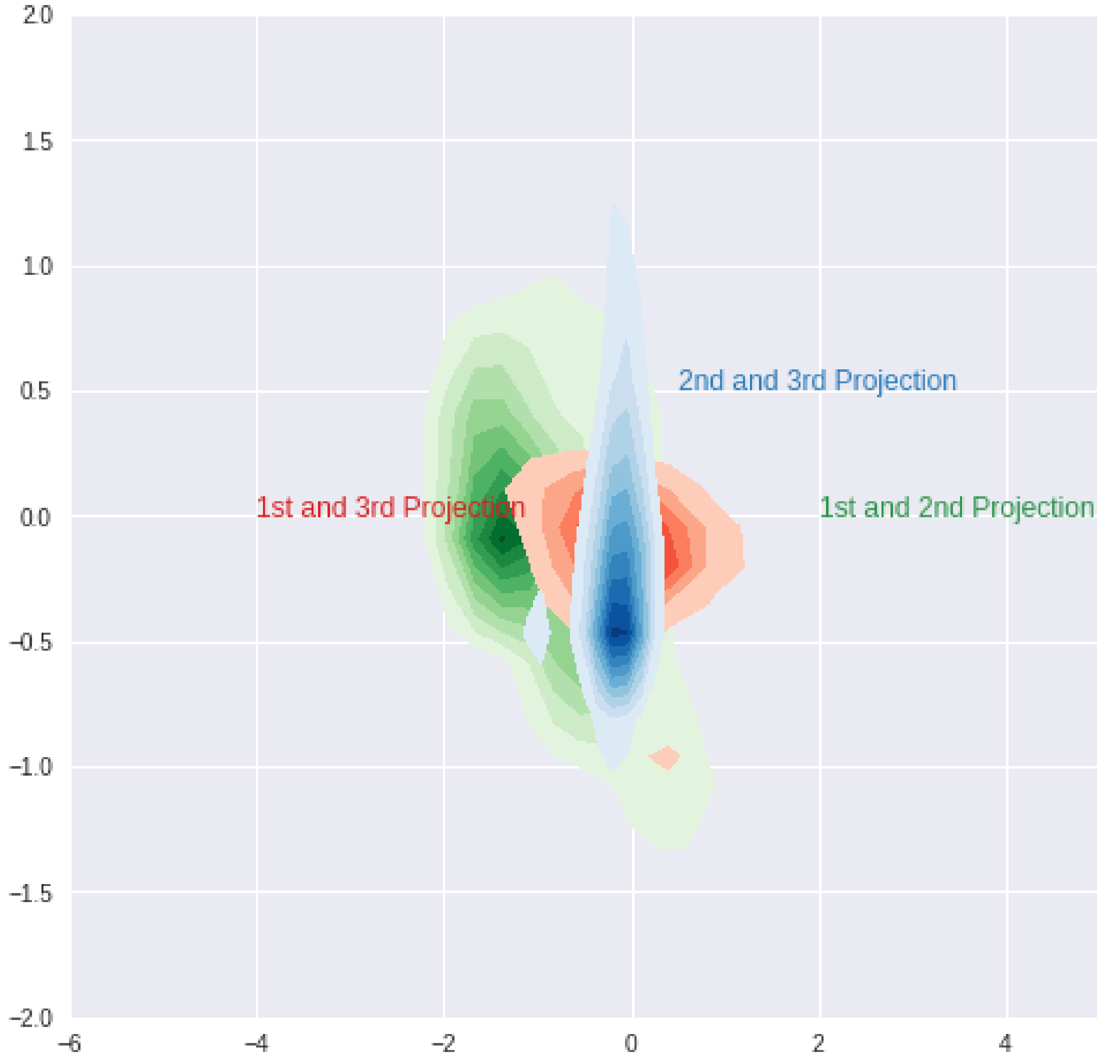
[11]: <matplotlib.text.Text at 0x7f3c05789278>



Since 5 components can explain more than 70% of the variance, we choose the number of the components to be 5

[12]:
```
from sklearn.decomposition import PCA
sklearn_pca=PCA(n_components=5)
X_Train=sklearn_pca.fit_transform(X_std)

sns.set(style='darkgrid')
f, ax = plt.subplots(figsize=(8, 8))
# ax.set_aspect('equal')
ax = sns.kdeplot(X_Train[:,0], X_Train[:,1], cmap="Greens",
          shade=True, shade_lowest=False)
ax = sns.kdeplot(X_Train[:,1], X_Train[:,2], cmap="Reds",
          shade=True, shade_lowest=False)
ax = sns.kdeplot(X_Train[:,2], X_Train[:,3], cmap="Blues",
          shade=True, shade_lowest=False)
red = sns.color_palette("Reds")[-2]
blue = sns.color_palette("Blues")[-2]
green = sns.color_palette("Greens")[-2]
ax.text(0.5, 0.5, "2nd and 3rd Projection", size=12, color=blue)
ax.text(-4, 0.0, "1st and 3rd Projection", size=12, color=red)
ax.text(2, 0, "1st and 2nd Projection", size=12, color=green)
plt.xlim(-6,5)
plt.ylim(-2,2)
```

[12]: (-2, 2)



[13]:
```
number_of_samples = len(y)
np.random.seed(0)
random_indices = np.random.permutation(number_of_samples)
num_training_samples = int(number_of_samples*0.75)
x_train = X_Train[random_indices[:num_training_samples]]
y_train=y[random_indices[:num_training_samples]]
x_test=X_Train[random_indices[num_training_samples:]]
y_test=y[random_indices[num_training_samples:]]
y_Train=list(y_train)
```

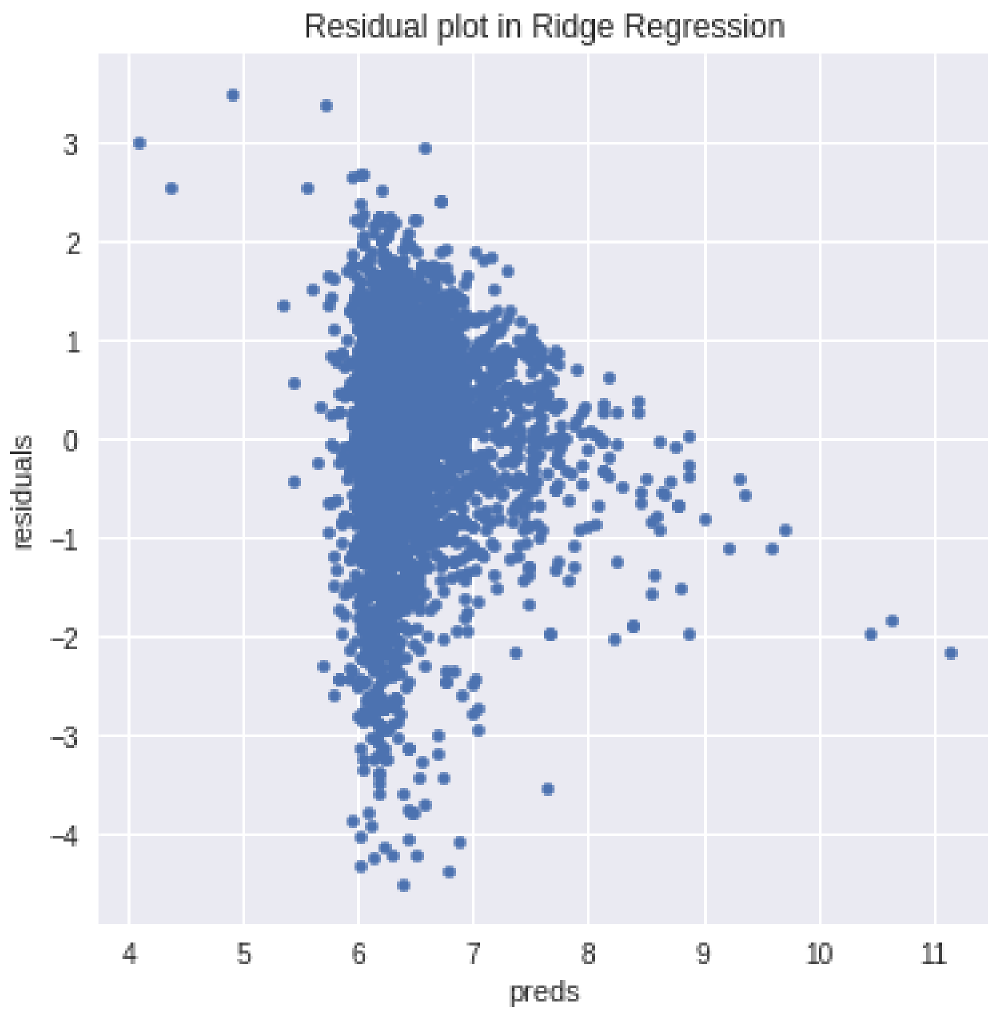
Ridge Regression

[14]:
```
model=linear_model.Ridge()
model.fit(x_train,y_train)
y_predict=model.predict(x_train)

error=0
for i in range(len(y_Train)):
    error+=(abs(y_Train[i]-y_predict[i])/y_Train[i])
train_error_ridge=error/len(y_Train)*100
print("Train error = "'{}'.format(train_error_ridge)+" percent in Ridge␣
 ↪Regression")

Y_test=model.predict(x_test)
y_Predict=list(y_test)

error=0
for i in range(len(y_test)):
    error+=(abs(y_Predict[i]-Y_test[i])/y_Predict[i])
test_error_ridge=error/len(Y_test)*100
print("Test error = "'{}'.format(test_error_ridge)+" percent in Ridge␣
 ↪Regression")
```

```
Train error = 13.914226734021002 percent in Ridge Regression
Test error = 15.299716605526257 percent in Ridge Regression
```

[15]:
```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)

preds = pd.DataFrame({"preds":model.predict(x_train), "true":y_train})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
plt.title("Residual plot in Ridge Regression")
```

[15]:
```
<matplotlib.text.Text at 0x7f3c040fd0f0>
```



Knn Algorithm

[16]:
```
n_neighbors=5
knn=neighbors.KNeighborsRegressor(n_neighbors,weights='uniform')
knn.fit(x_train,y_train)
y1_knn=knn.predict(x_train)
y1_knn=list(y1_knn)

error=0
for i in range(len(y_train)):
    error+=(abs(y1_knn[i]-y_Train[i])/y_Train[i])
train_error_knn=error/len(y_Train)*100
print("Train error = "+'{}'.format(train_error_knn)+" percent"+" in Knn␣
 ↪algorithm")

y2_knn=knn.predict(x_test)
```

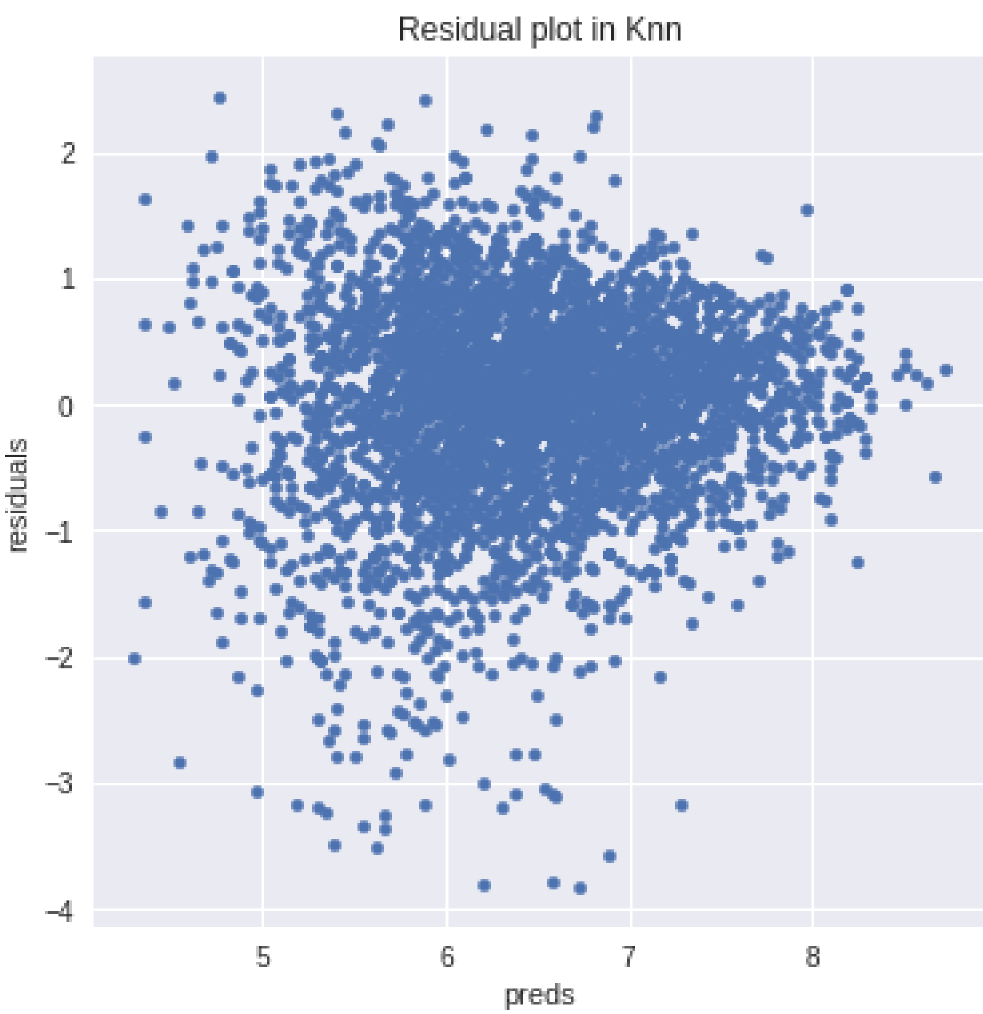
```
y2_knn=list(y2_knn)
error=0
for i in range(len(y_test)):
    error+=(abs(y2_knn[i]-Y_test[i])/Y_test[i])
test_error_knn=error/len(Y_test)*100
print("Test error = "'{}'.format(test_error_knn)+" percent"+" in knn algorithm")
```

```
Train error = 10.812937212714084 percent in Knn algorithm
Test error = 6.878221673331934 percent in knn algorithm
```

[17]:

```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)
preds = pd.DataFrame({"preds":knn.predict(x_train), "true":y_train})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
plt.title("Residual plot in Knn")
```

[17]: <matplotlib.text.Text at 0x7f3bfc306160>



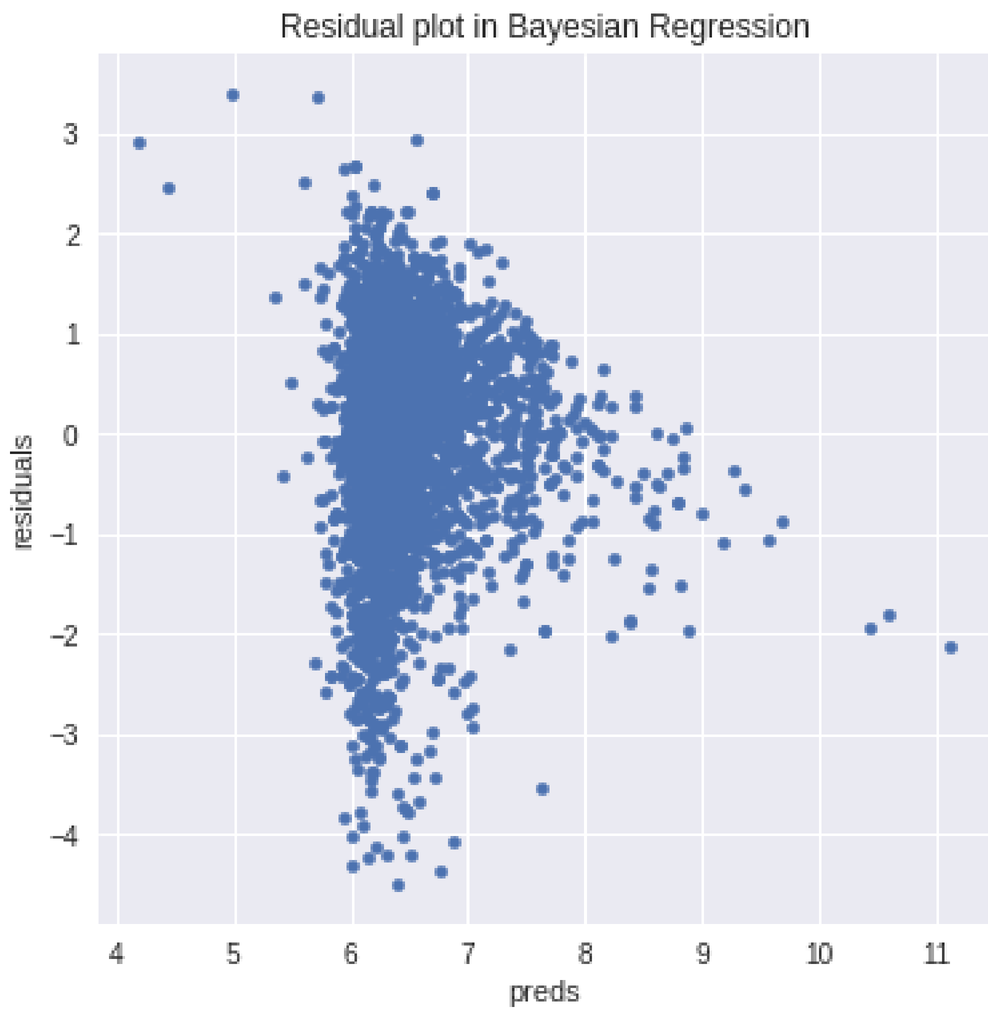
Bayesian Regression

[18]:
```
reg = linear_model.BayesianRidge()
reg.fit(x_train,y_train)
y1_reg=reg.predict(x_train)
y1_reg=list(y1_reg)
y2_reg=reg.predict(x_test)
y2_reg=list(y2_reg)

error=0
for i in range(len(y_train)):
    error+=(abs(y1_reg[i]-y_Train[i])/y_Train[i])
train_error_bay=error/len(y_Train)*100
print("Train error = "+'{}'.format(train_error_bay)+" percent"+" in Bayesian␣
 ↪Regression")

error=0
for i in range(len(y_test)):
    error+=(abs(y2_reg[i]-Y_test[i])/Y_test[i])
test_error_bay=(error/len(Y_test))*100
print("Test error = "+'{}'.format(test_error_bay)+" percent"+" in Bayesian␣
 ↪Regression")
```

```
Train error = 13.91749661366315 percent in Bayesian Regression
Test error = 0.025287435537397897 percent in Bayesian Regression
```

[19]:
```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)
preds = pd.DataFrame({"preds":reg.predict(x_train), "true":y_train})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
plt.title("Residual plot in Bayesian Regression")
```

[19]:
```
<matplotlib.text.Text at 0x7f3bfc2f80b8>
```



Decision Tree Regressor

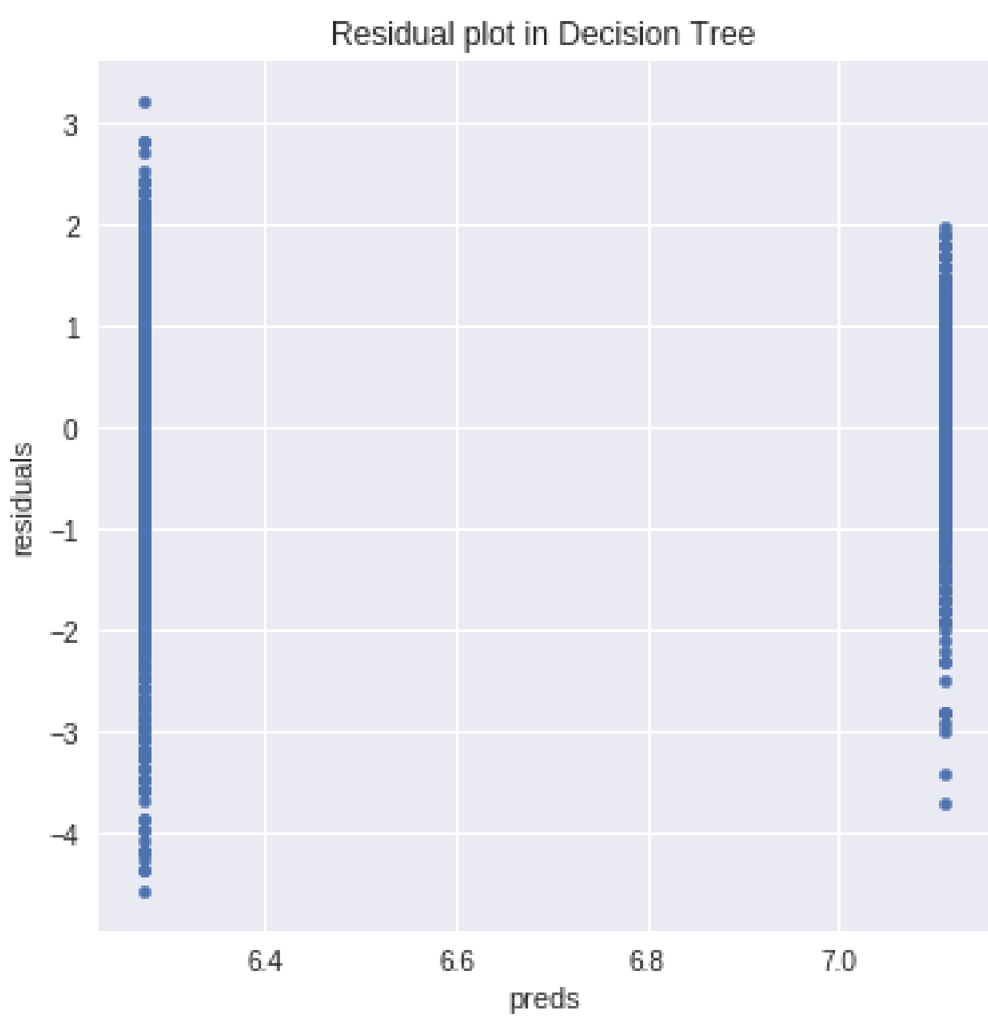
[20]:
```
dec = tree.DecisionTreeRegressor(max_depth=1)
dec.fit(x_train,y_train)
y1_dec=dec.predict(x_train)
y1_dec=list(y1_dec)
y2_dec=dec.predict(x_test)
y2_dec=list(y2_dec)

error=0
for i in range(len(y_train)):
    error+=(abs(y1_dec[i]-y_Train[i])/y_Train[i])
train_error_tree=error/len(y_Train)*100
print("Train error = "+'{}'.format(train_error_tree)+" percent"+" in Decision␣
 ↪Tree Regressor")
```

```
error=0
for i in range(len(y_test)):
    error+=(abs(y1_dec[i]-Y_test[i])/Y_test[i])
test_error_tree=error/len(Y_test)*100
print("Test error = "'{}'.format(test_error_tree)+" percent in Decision Tree␣
  ↪Regressor")
```

```
Train error = 14.590941891509965 percent in Decision Tree Regressor
Test error = 5.816650087351861 percent in Decision Tree Regressor
```

[21]:
```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)
preds = pd.DataFrame({"preds":dec.predict(x_train), "true":y_train})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
plt.title("Residual plot in Decision Tree")
```

[21]: <matplotlib.text.Text at 0x7f3bfc22f7b8>



SVM

[22]:
```
svm_reg=svm.SVR()
svm_reg.fit(x_train,y_train)
y1_svm=svm_reg.predict(x_train)
y1_svm=list(y1_svm)
y2_svm=svm_reg.predict(x_test)
y2_svm=list(y2_svm)

error=0
for i in range(len(y_train)):
    error+=(abs(y1_svm[i]-y_Train[i])/y_Train[i])
train_error_svm=error/len(y_Train)*100
print("Train error = "+'{}'.format(train_error_svm)+" percent"+" in SVM␣
  ↪Regressor")

error=0
for i in range(len(y_test)):
    error+=(abs(y2_svm[i]-Y_test[i])/Y_test[i])
test_error_svm=error/len(Y_test)*100
print("Test error = "'{}'.format(test_error_svm)+" percent in SVM Regressor")
```

```
Train error = 12.036337747988636 percent in SVM Regressor
Test error = 5.403852057483367 percent in SVM Regressor
```

[23]:
```
matplotlib.rcParams['figure.figsize'] = (6.0, 6.0)
preds = pd.DataFrame({"preds":knn.predict(x_train), "true":y_train})
preds["residuals"] = preds["true"] - preds["preds"]
preds.plot(x = "preds", y = "residuals",kind = "scatter")
plt.title("Residual plot in SVM")
```

[23]: <matplotlib.text.Text at 0x7f3bfc172198>

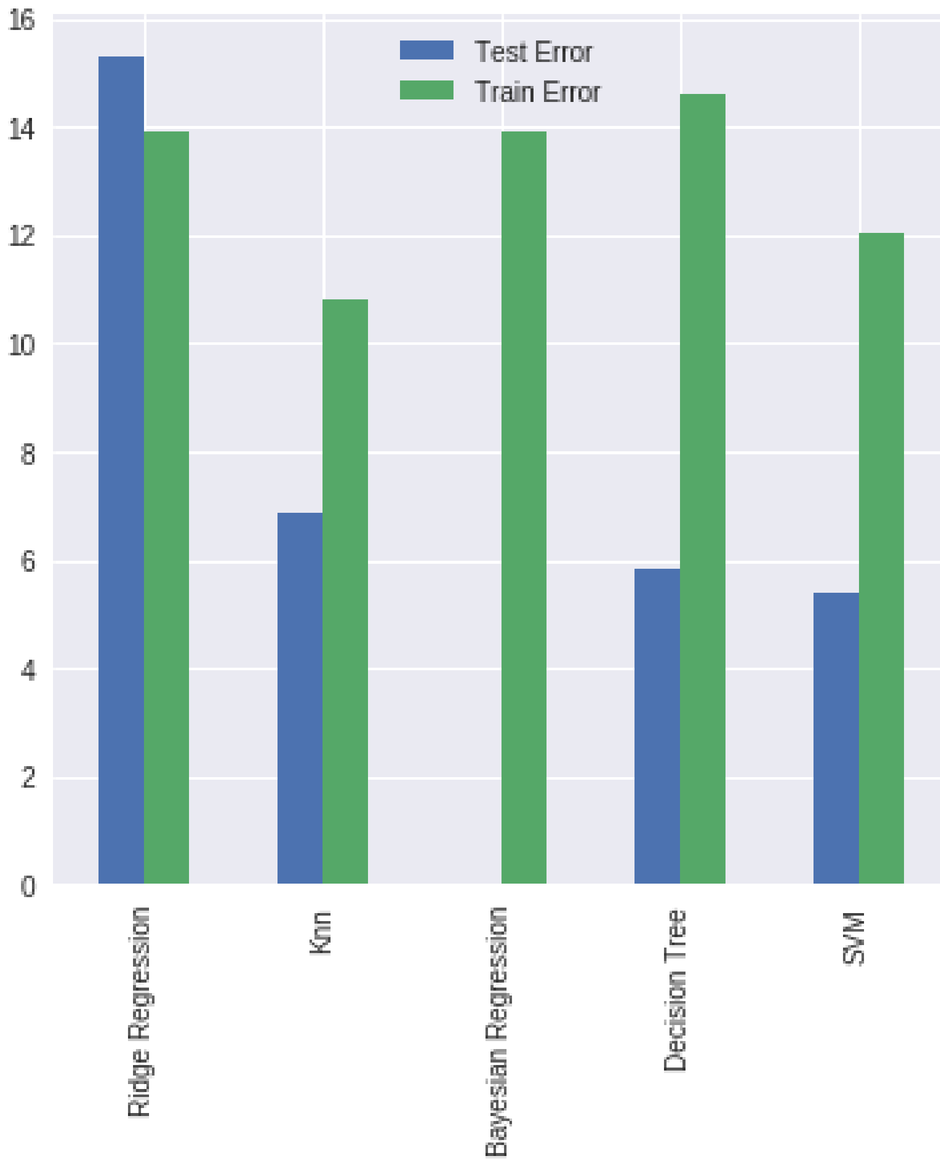
[24]:
```
train_error=[train_error_ridge,train_error_knn,train_error_bay,train_error_tree,train_error_sv
test_error=[test_error_ridge,test_error_knn,test_error_bay,test_error_tree,test_error_svm]

col={'Train Error':train_error,'Test Error':test_error}
models=['Ridge Regression','Knn','Bayesian Regression','Decision Tree','SVM']
df=DataFrame(data=col,index=models)
df
```

[24]:

	Test Error	Train Error
Ridge Regression	15.299717	13.914227
Knn	6.878222	10.812937
Bayesian Regression	0.025287	13.917497
Decision Tree	5.816650	14.590942
SVM	5.403852	12.036338

[25]:
```
df.plot(kind='bar')
```

[25]: <matplotlib.axes._subplots.AxesSubplot at 0x7f3bfc0af160>



Seems that KNN turned out to be the winner.Its because of the fact that there are very large number of data points and and also features are highly continuous *Moreover the dimentionality of the processed data is not too high*