HomeWork 2
BDS

Group 2
Rithu Anand Krishnan
Manvi Mahajan
Aman Bhardwaj

Q1

In [1]:
```
import pandas as pd
import numpy as np
```

In [2]:
```
#Reading values from DF1 and storing it in df1

df1 = pd.read_csv(r'Lab2_Data/DF1',index_col=0, sep=',')
df1
```

Out[2]:

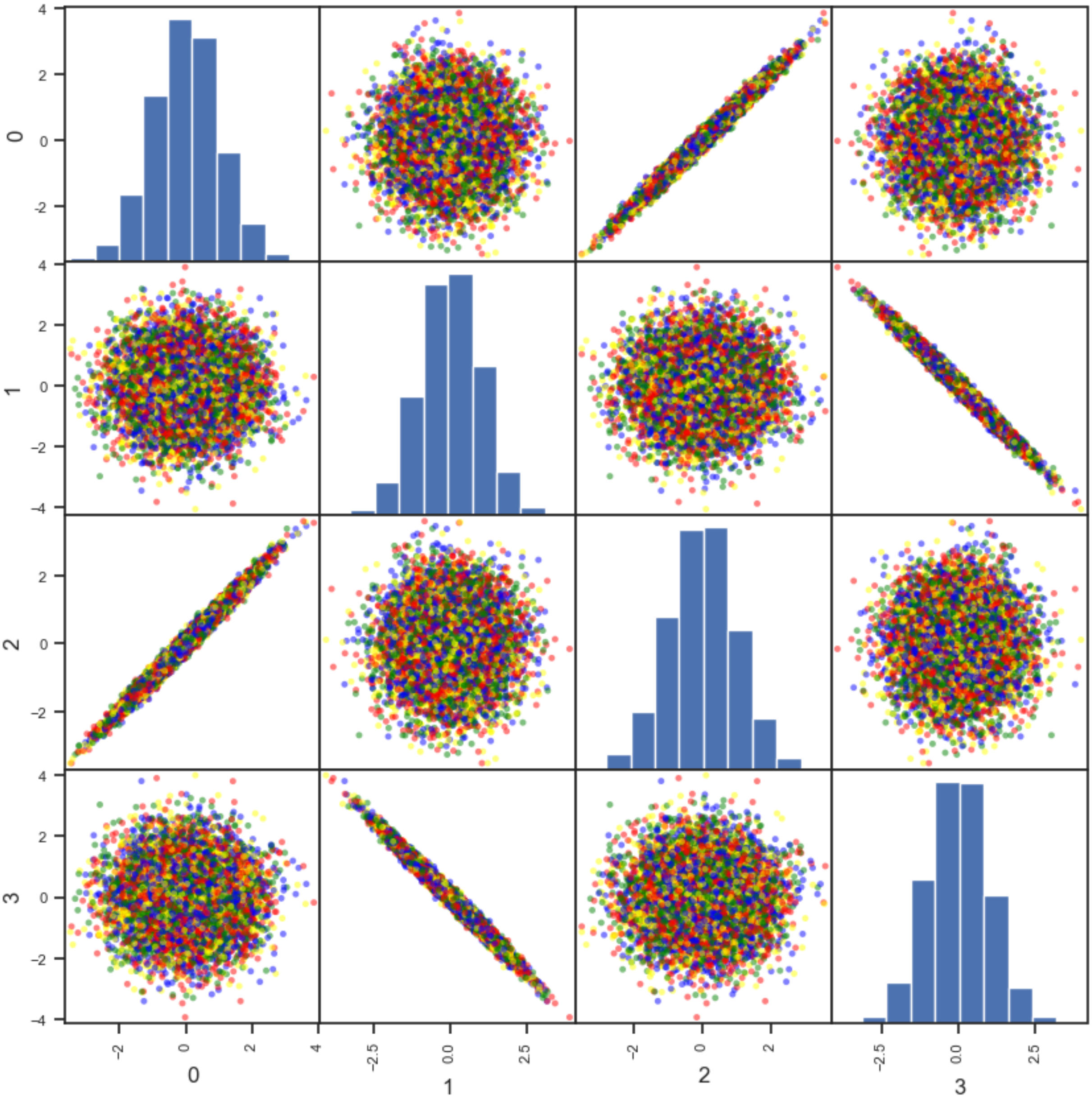
	0	1	2	3
0	1.038502	0.899865	0.835053	-0.971528
1	0.320455	-0.647459	0.149079	0.352593
2	0.055480	2.234771	0.271672	-2.108739
3	-0.007260	-0.524299	-0.126550	0.670827
4	-1.237390	-1.377017	-1.049932	1.342079
...	...	...	...	...
9995	-0.632309	-0.145873	-0.797517	0.436184
9996	0.679417	-0.530216	0.526470	0.439397
9997	0.890697	-2.210855	1.072751	2.285372
9998	0.475293	0.490971	0.536909	-0.195772
9999	1.207406	0.819239	1.230797	-0.752397

10000 rows × 4 columns

In [16]:
```
import random
colors = list()
palette = {0: "red", 1: "green", 2: "blue", 3: "yellow"}

for c in range(0,10000): colors.append(palette[random.randint(0, 3)])
```

In [18]:
```
import matplotlib.pyplot as plt
%matplotlib inline


print("Plot to display DataSet df1 values")


pd.plotting.scatter_matrix(df1,  figsize = (10,10), color=colors, s=50)
plt.show()
```

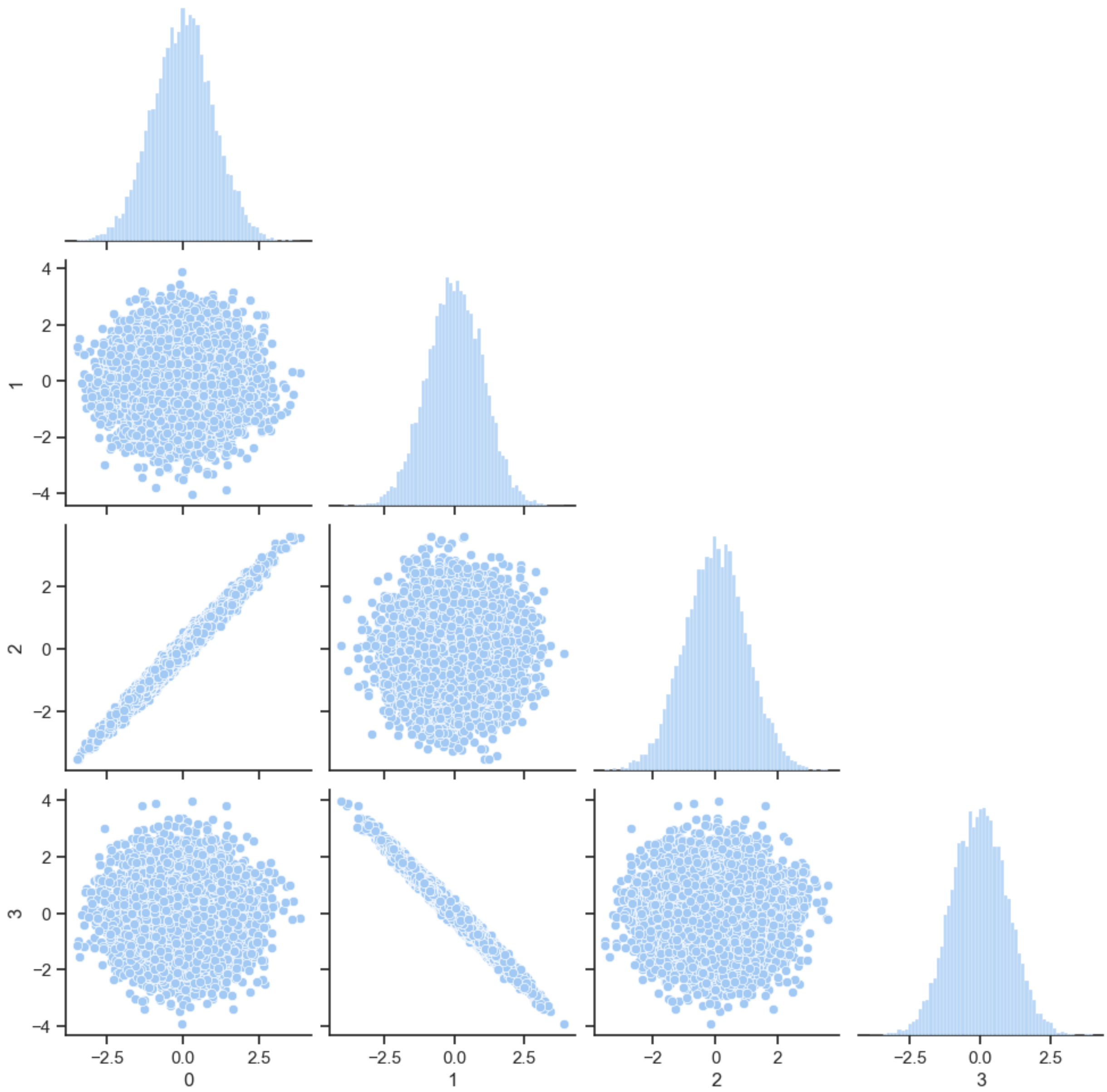
```
Plot to display DataSet df1 values
```



In [37]:

```
#Using seaborn to plot co-relation
import seaborn as sns

sns.set(style="ticks", color_codes=True)

print("Plot to display correlation with Seaborn")

palette = sns.color_palette("bright")
sns.set_palette("pastel")
sns.pairplot(df1, corner=True)
plt.show()
```

```
Plot to display correlation with Seaborn
```



In [40]:
```
#Computing Covariance of df1

df_cov = np.cov(df1)
```

In [39]:
```
df_covariance = np.cov(df1,rowvar=False)
print('The covariance matrix of df1 is:')
print(df_covariance)
```

```
The covariance matrix of df1 is:
[[ 1.00155793 -0.00401176  0.99162409  0.00412485]
 [-0.00401176  1.00537841 -0.00409877 -0.99545662]
 [ 0.99162409 -0.00409877  1.00158867  0.00408108]
 [ 0.00412485 -0.99545662  0.00408108  1.00516828]]
```

In [43]:
```
#Choose a symmetric matrix
sym_matrix = [[1.0, 0.0, 0.0],\
              [0.0, 1.0, 0.4], \
              [0.0, 0.4, 1.0]]
print("Sample symmetric matrix")
print(sym_matrix)
```

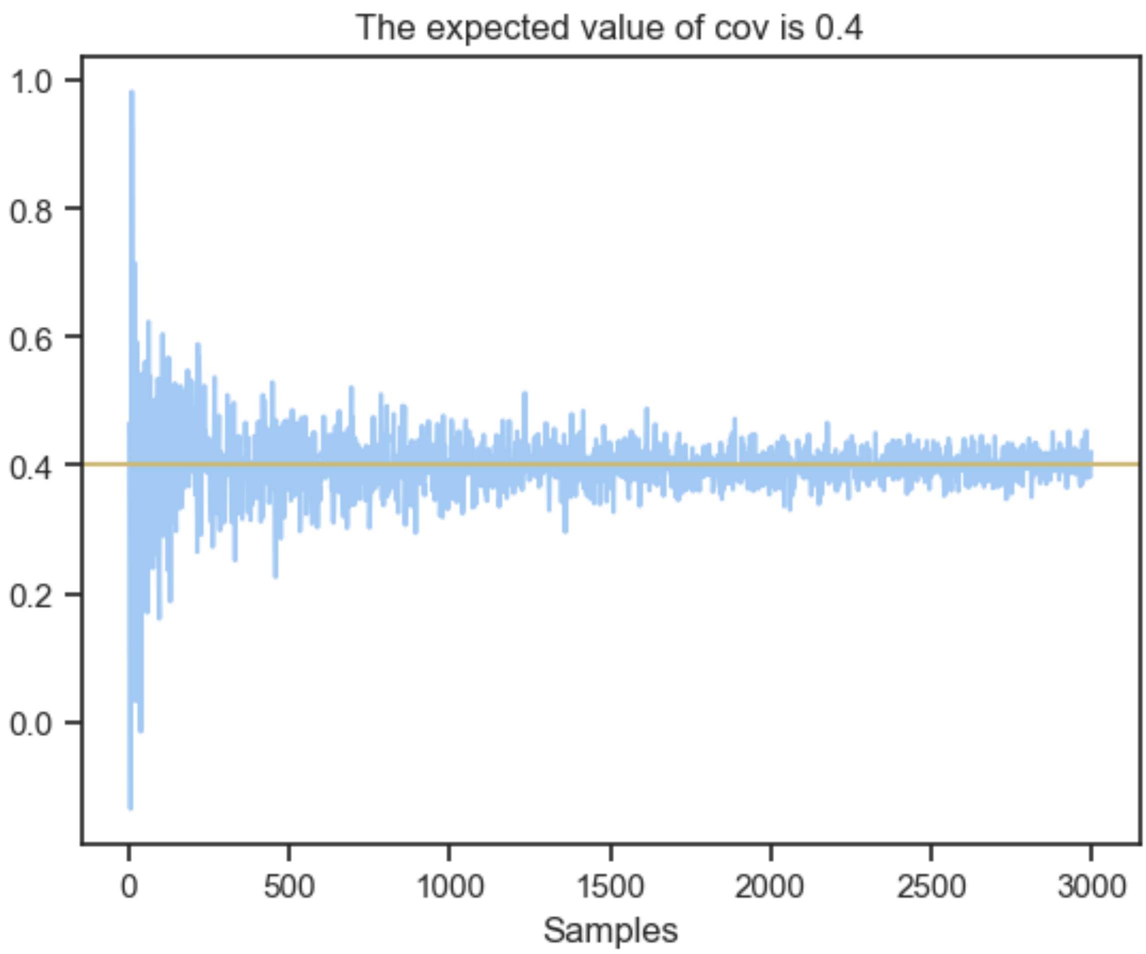
```
Sample symmetric matrix
[[1.0, 0.0, 0.0], [0.0, 1.0, 0.4], [0.0, 0.4, 1.0]]
```

In [46]:
```
zero_list = [0, 0, 0]
```

In [52]:
```
#Using multivariate_normal to derive the covariance
covariance_x = range(0, 3000, 2)
covariance_y =[np.cov(np.random.multivariate_normal(zero_list, size=n, co
```

```
/var/folders/gp/xs_xkf814zx898m2wdld0hw40000gn/T/ipykernel_96606/12622978
08.py:2: RuntimeWarning: Degrees of freedom <= 0 for slice
  covariance_y =[np.cov(np.random.multivariate_normal(zero_list, size=n,
cov=sym_matrix),rowvar=False)[1][2] for n in covariance_x]
```

In [53]:
```
#From the plot it is clear that as the number of the samples we use incre
plt.xlabel('Samples')
plt.title('The expected value of cov is 0.4')
plt.plot(x,y)
plt.axhline(0.4, 0, 3000, color='y')
plt.show()
```



Due to the significant covariance between a variable and itself, the covariances along the diagonal tend to be one. Regarding the other high covariances, we can observe a strong correlation between the variances of the corresponding pairs of variables from the plots. This results in exceptionally densely clustered points along a negatively or positive axis, depending on the covariance sign. There is typically no correlation between the distribution of the covariances with relatively low values and the random groupings of points in the plots.

Question 2:

In [59]:
```
df2 = pd.read_csv(r'Lab2_Data/DF2', index_col=[0], sep=',')
df2
```

Out[59]:

	0	1
0	1.038502	0.835053
1	0.320455	0.149079
2	0.055480	0.271672
3	-0.007260	-0.126550
4	-1.237390	-1.049932
...	...	...
9995	-0.632309	-0.797517
9996	0.679417	0.526470
9997	0.890697	1.072751
9998	0.475293	0.536909
9999	1.207406	1.230797

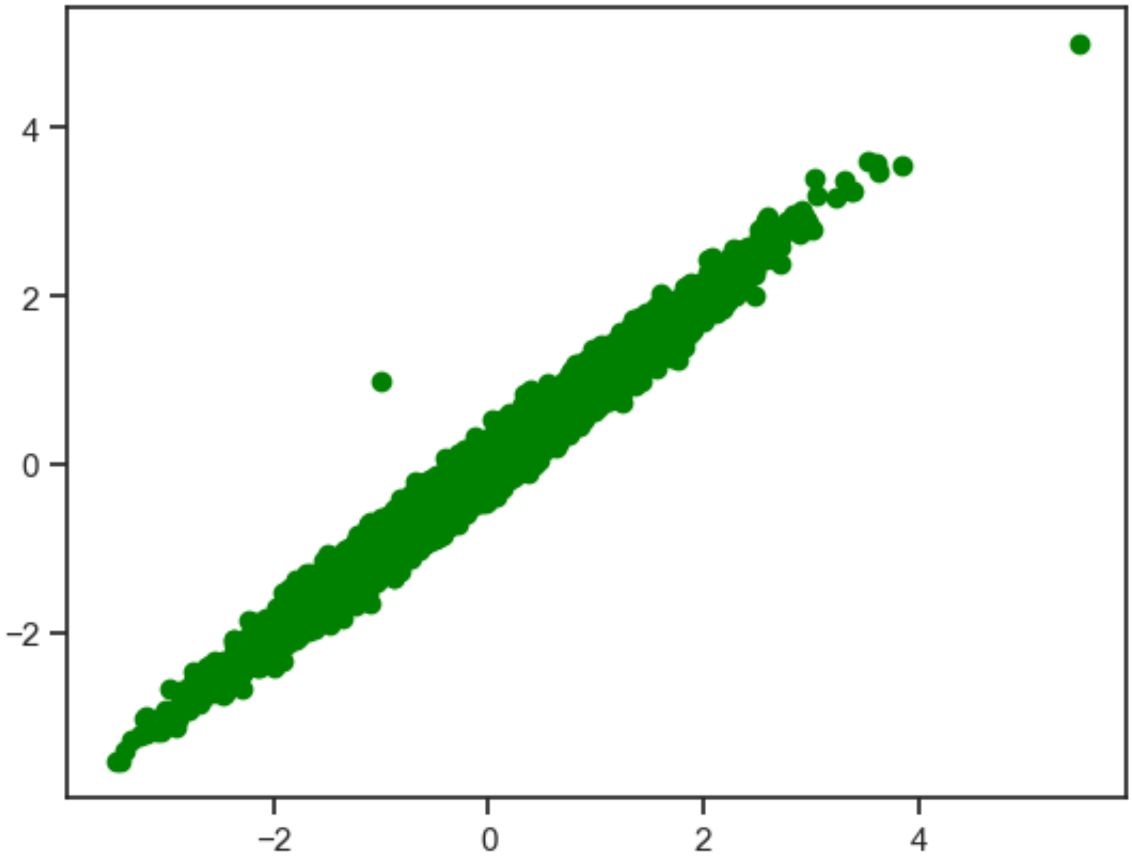
10000 rows × 2 columns

In [60]:

```
#To plot the dataframe
first_variable = df2.iloc[:,0]
second_variable = df2.iloc[:,1]
```

In [61]:

```
plt.scatter(first_variable,second_variable, color='green')
plt.show()
```



In [63]:
```
#calculating the covariance based on the above scatterplot inorder to qua
#the variability between the datasets and outliers

df2_cov = np.cov(df2, rowvar=False)
```

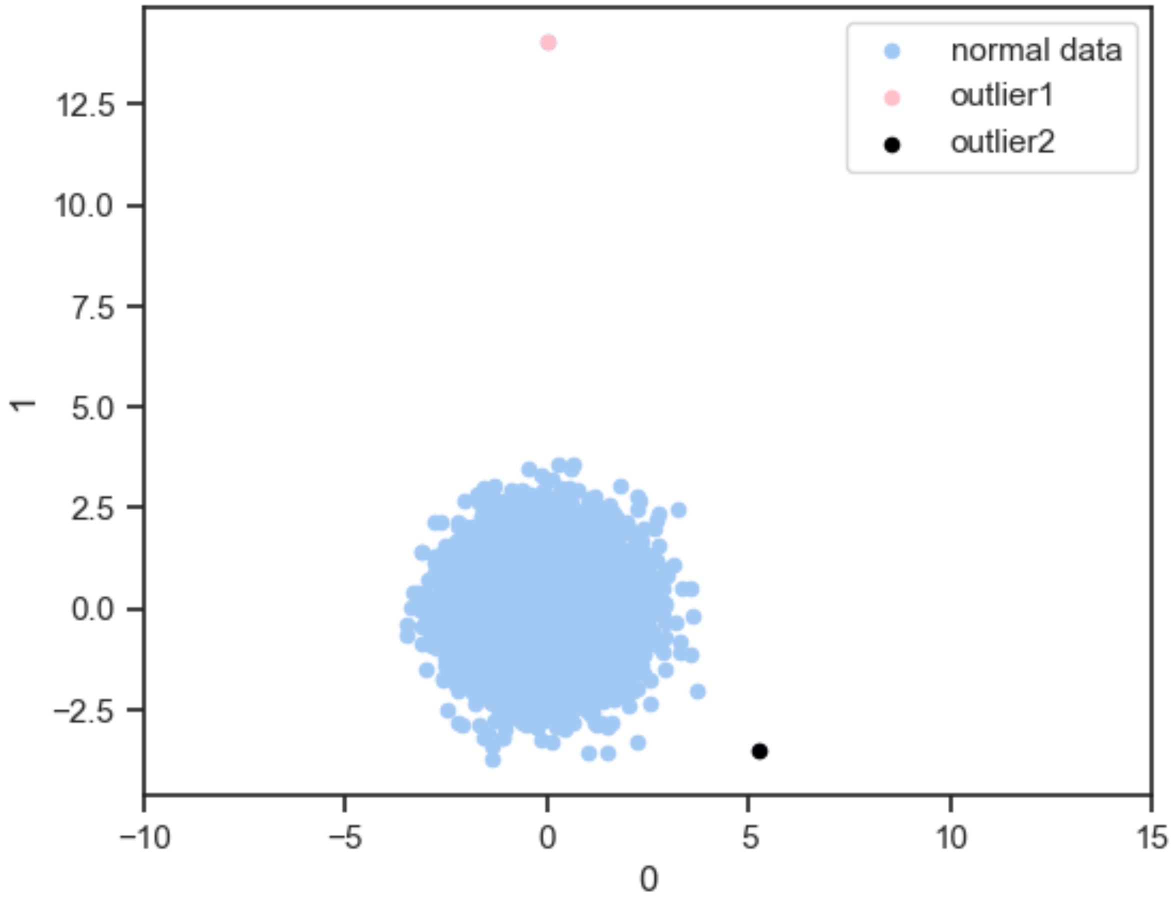
In [65]:
```
#calculating the eigenvalues and right eigenvectors for the given square
#(covariance matrix)

i,j = np.linalg.eig(df2_cov)
```

In [73]:
```
 #calculating covariance value
 #Extract the diagonal array through the concept of eigen values matrix
var = i**(-1/2)
Q = np.diag(var) @ j.T
new_Data = Q @ df2.T
new_Data = new_Data.T
#Estimating the dot product
```

In [82]:
```
#Getting values of outliers
outlier1 = df2.index[(df2['0']==-1) & (df2['1']==1)]
outlier2 = df2.index[(df2['0']==5.5) & (df2['1']==5)]
```

In [83]:
```
ax_val = new_Data.plot(x=0, y=1, kind="scatter")
new_Data.loc[outlier1].plot(x=0, y=1, kind="scatter", ax=ax_val, c="pink"
new_Data.loc[outlier2].plot(x=0, y=1, kind="scatter", ax=ax_val, c="black
ax_val.legend(['normal data','outlier1','outlier2'])
ax_val.set_aspect('equal')
ax_val.set_xlim(-10,15)
#now plotting the scatter plot to depict the outlier points
#Using the above eigen values, covariance and dot product to show that (5
#is nearer than (-1,1)
```

Out[83]: (-10.0, 15.0)



Question 3

In [84]:
```
import numpy as np
import pandas as pd
import glob
import os

from scipy.stats import percentileofscore
from collections import defaultdict
from operator import add
from pyspark.mllib.feature import HashingTF, IDF
import math, random

year = input("Enter year:")
k = int(input("Enter k:"))
#year = './Names/yob' + year + '.txt'


#df = pd.read_csv(year, index_col=None, header=0)
#year=2015
chosenYear = pd.read_csv("Names/yob%s.txt" % year ,names = ["Name", "Gend

#df.loc[df['Name'] == name]

chosenYear['Name'].value_counts().head(k)
```

```
Enter year:2015
Enter k:10
```

Out[84]:
```
Emma       2
Chase      2
Karson     2
Nino       2
Lyrick     2
Sully      2
Nix        2
Taygen     2
Stone      2
Wyatt      2
Name: Name, dtype: int64
```

In [85]:
```
chosenYear.head(5)
```

Out[85]:

	Name	Gender	Number
0	Emma	F	20355
1	Olivia	F	19553
2	Sophia	F	17327
3	Ava	F	16286
4	Isabella	F	15504

In [86]:
```
#part 2

path = r'Names/'                     # use your path
all_files = glob.glob(os.path.join(path, "*.txt"))

li = []

for filename in all_files:
    date = filename.replace('Names/yob','').replace('.txt','')

    df = pd.read_csv(filename, index_col=None, header=0,names = ["Name",
    df['year'] = date
    li.append(df)

df = pd.concat(li, axis=0, ignore_index=True)
name = input("Enter name to evaluate for men and women:")
namesumF = df.loc[df['Name'] == name].loc[df['Gender']=='F'].Number.sum()
print("The name ",name," is repated",namesumF, " times for Females")

namesumM = df.loc[df['Name'] == name].loc[df['Gender']=='M'].Number.sum()
print("The name ",name," is repated",namesumM, " times for Males")
```

```
Enter name to evaluate for men and women:Sara
The name  Sara  is repated 419025  times for Females
The name  Sara  is repated 1236  times for Males
```

In [87]:
```
#Calculate frequency of name but per year
def nameFreqByYear(name):

    #getting people with same name and gender in chosen year
    maleCol = chosenYear['Number'].loc[(chosenYear['Name'] == name) & (ch
    femaleCol = chosenYear['Number'].loc[(chosenYear['Name'] == name) & (

    if maleCol.empty is False:
        maleCount = maleCol.values[0]
    else:
        maleCount = 0

    if femaleCol.empty is False:
        femaleCount = femaleCol.values[0]
    else:
        femaleCount = 0

    return maleCount,femaleCount
```

In [88]:
```
#method to calculate relative frequency as mapped to total names
def relativeFreq(name):
    maleCount,femaleCount = nameFreqByYear(name)

    sumByYear = np.sum(chosenYear,axis=0)[2]
    ansMale = maleCount/sumByYear
    ansFemale = femaleCount/sumByYear

    return ansMale,ansFemale
```

In [89]:
```
#method for frequency per year
def freqPerYear(name,flag):
    ret = []
    for year in range(1880, 2016):
        chosenYear['year'] = year
        if not flag:
            maleFrequency, femaleFrequency = nameFreqByYear(name)
            chosenYear.loc[(chosenYear['Name'] == name) & (chosenYear['ye
            chosenYear.loc[(chosenYear['Name'] == name) & (chosenYear['ye
        else:
            maleFrequency, femaleFrequency = relativeFreq(name)
            chosenYear.loc[(chosenYear['Name'] == name) & (chosenYear['ye
            chosenYear.loc[(chosenYear['Name'] == name) & (chosenYear['ye


    ret.append(chosenYear.loc[(chosenYear['Name'] == name) & (chosenYear[
    ret = pd.concat(ret)
    return ret
```

In [90]:
```
#answer for part 3
relativeFrequencyForName = freqPerYear('Sara', flag=True)
print("The relative frequency for chosen name that is Sara in chosen year
```

```
The relative frequency for chosen name that is Sara in chosen year:
      year  Name  Frequency Gender
161   2015  Sara   0.000535      F
```

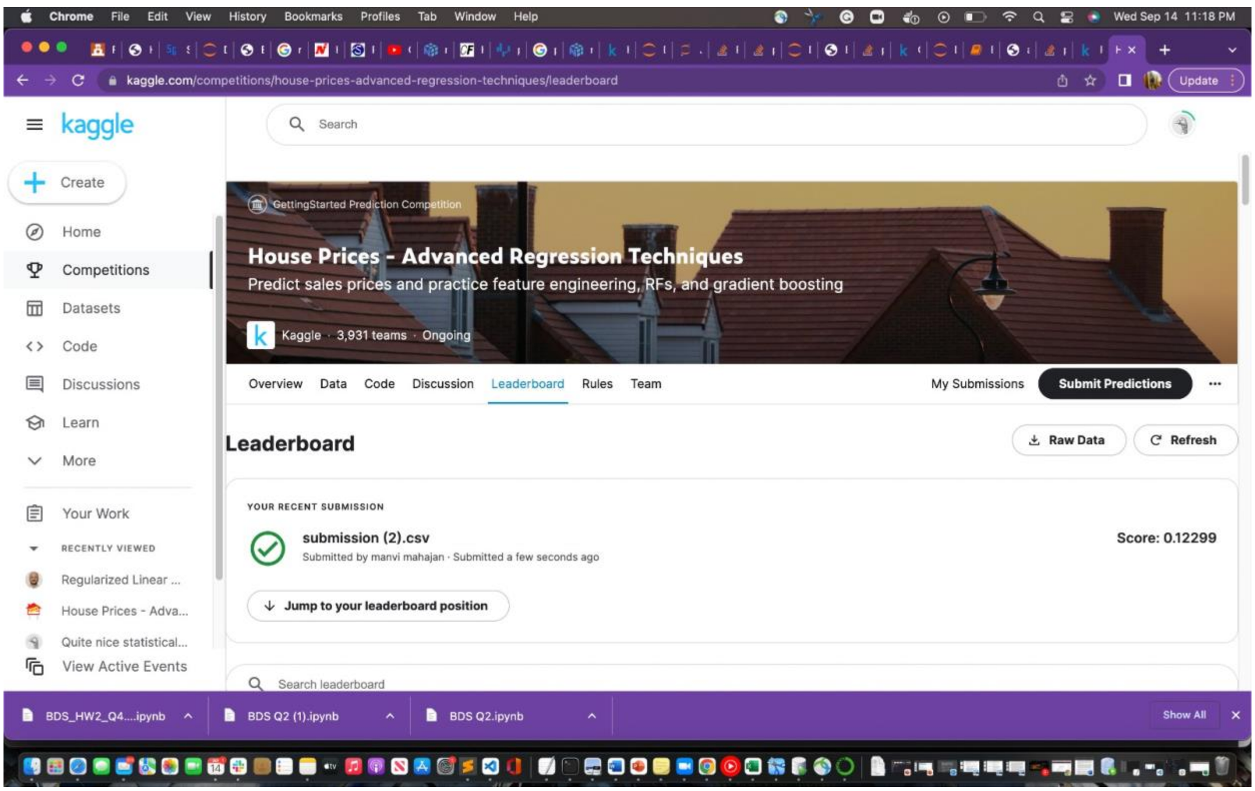
In [91]:
```
#Part 4


#method to loop over given year file
def getDataForYear(year):
    ret = pd.read_csv("Names/yob%s.txt" % year ,names = ["Name", "Gender"
    return ret


#calculates names that were popular for M and then swicthed to F and vice
def namesPopularitySwitch():
    ret = []

    #loop over every year
    for year in range(1880,2016):
        temp = getDataForYear(year)
        temp['Year'] = year
        ret.append(temp)


    ret = pd.concat(ret)

    #assign change column
    ret["change"] = ret["Gender"].map({'M':1,'F':-1})
    ret["change"] = ret["change"] * ret["Number"]

    #drop year info
    ret = ret.groupby(["Name","Year"]).sum()
    ret = ret.reset_index().drop('Year',1)

    #assign extreme values
    ret = ret.groupby('Name').agg({'change':['min','max']})
    ret.columns = ['min', 'max']


    #create new switch column if there was flag switch in name and remove
    ret['switch'] = (np.sign(ret["min"] * ret['max']) == -1)
    ret = ret[ret['switch']].reset_index()

    return ret['Name'].values


print(namesPopularitySwitch())
```

```
['Aalijah' 'Aamari' 'Aaren' ... 'Zy' 'Zyaire' 'Zyian']
```

```
/var/folders/gp/xs_xkf814zx898m2wdld0hw40000gn/T/ipykernel_96606/31997127
52.py:29: FutureWarning: In a future version of pandas all arguments of D
ataFrame.drop except for the argument 'labels' will be keyword-only.
  ret = ret.reset_index().drop('Year',1)
```

In [ ]:

Screenshot from Kaggle for Group 2 for question 4



In [1]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib
import xgboost as xgb
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from scipy.stats import skew
from scipy.stats.stats import pearsonr

%config InlineBackend.figure_format = 'png' #set 'png' here when working
%matplotlib inline
from sklearn.model_selection import KFold, cross_val_score
```

In [38]:
```
train = pd.read_csv(r'train.csv')
test = pd.read_csv(r'test.csv')
```

In [39]:
```
train
```

Out[39]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Lan
0	1	60	RL	65.0	8450	Pave	NaN	Reg	
1	2	20	RL	80.0	9600	Pave	NaN	Reg	
2	3	60	RL	68.0	11250	Pave	NaN	IR1	
3	4	70	RL	60.0	9550	Pave	NaN	IR1	
4	5	60	RL	84.0	14260	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	
1455	1456	60	RL	62.0	7917	Pave	NaN	Reg	
1456	1457	20	RL	85.0	13175	Pave	NaN	Reg	
1457	1458	70	RL	66.0	9042	Pave	NaN	Reg	
1458	1459	20	RL	68.0	9717	Pave	NaN	Reg	
1459	1460	20	RL	75.0	9937	Pave	NaN	Reg	

1460 rows × 81 columns

In [40]:
```
test
```

Out[40]:

	Id	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	Lan
0	1461	20	RH	80.0	11622	Pave	NaN	Reg	
1	1462	20	RL	81.0	14267	Pave	NaN	IR1	
2	1463	60	RL	74.0	13830	Pave	NaN	IR1	
3	1464	60	RL	78.0	9978	Pave	NaN	IR1	
4	1465	120	RL	43.0	5005	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	...	
1454	2915	160	RM	21.0	1936	Pave	NaN	Reg	
1455	2916	160	RM	21.0	1894	Pave	NaN	Reg	
1456	2917	20	RL	160.0	20000	Pave	NaN	Reg	
1457	2918	85	RL	62.0	10441	Pave	NaN	Reg	
1458	2919	60	RL	74.0	9627	Pave	NaN	Reg	

1459 rows × 80 columns

In [41]:
```
all_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],test.loc[
```

In [42]:
```
all_data
```

Out[42]:

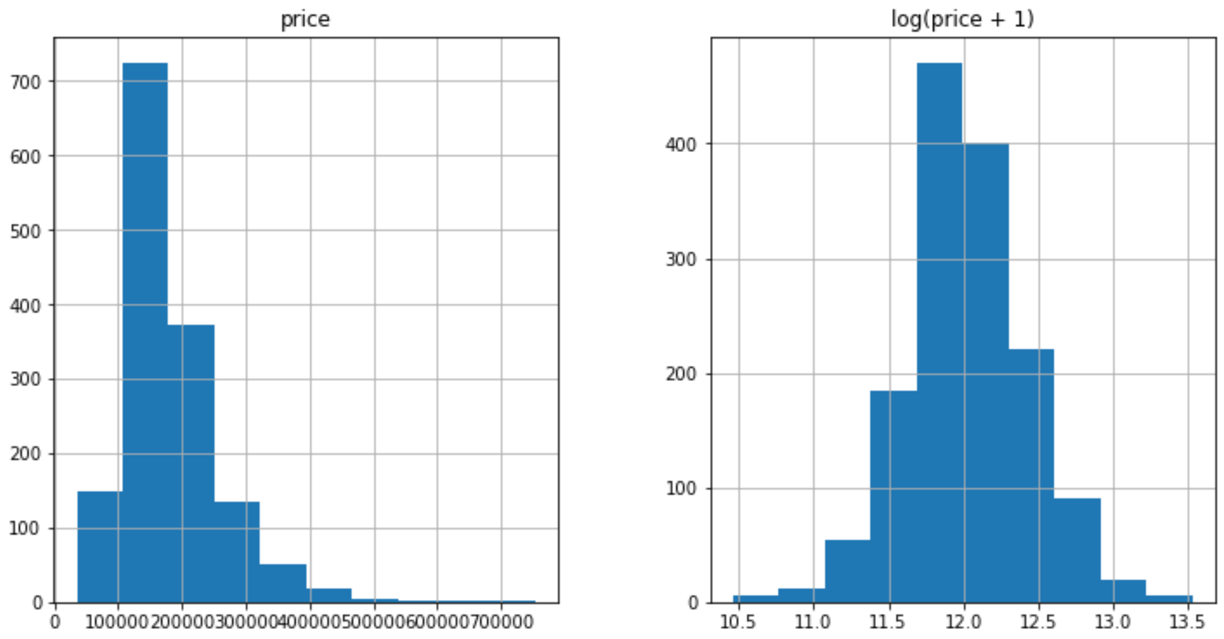
	MSSubClass	MSZoning	LotFrontage	LotArea	Street	Alley	LotShape	LandContc
0	60	RL	65.0	8450	Pave	NaN	Reg	
1	20	RL	80.0	9600	Pave	NaN	Reg	
2	60	RL	68.0	11250	Pave	NaN	IR1	
3	70	RL	60.0	9550	Pave	NaN	IR1	
4	60	RL	84.0	14260	Pave	NaN	IR1	
...	...	...	...	...	...	...	...	
1454	160	RM	21.0	1936	Pave	NaN	Reg	
1455	160	RM	21.0	1894	Pave	NaN	Reg	
1456	20	RL	160.0	20000	Pave	NaN	Reg	
1457	85	RL	62.0	10441	Pave	NaN	Reg	
1458	60	RL	74.0	9627	Pave	NaN	Reg	

2919 rows × 79 columns

In [43]:
```
# model regression . fit
# model lasso . fit
```

In [44]:
```
#features more normal
matplotlib.rcParams['figure.figsize'] = (12.0, 6.0)
prices = pd.DataFrame({"price":train["SalePrice"], "log(price + 1)":np.lo
prices.hist()
```

Out[44]:
```
array([[<AxesSubplot:title={'center':'price'}>,
        <AxesSubplot:title={'center':'log(price + 1)'}>]], dtype=object)
```



In [45]:
```
train["SalePrice"] = np.log1p(train["SalePrice"])
```

In [46]:
```
train["SalePrice"] #define target
```

Out[46]:
```
0       12.247699
1       12.109016
2       12.317171
3       11.849405
4       12.429220
          ...
1455    12.072547
1456    12.254868
1457    12.493133
1458    11.864469
1459    11.901590
Name: SalePrice, Length: 1460, dtype: float64
```

In [47]:
```
numeric_feats = all_data.dtypes[all_data.dtypes != "object"].index
```

In [48]:
```
numeric_feats
```

Out[48]:
```
Index(['MSSubClass', 'LotFrontage', 'LotArea', 'OverallQual', 'OverallCon
d',
       'YearBuilt', 'YearRemodAdd', 'MasVnrArea', 'BsmtFinSF1', 'BsmtFinS
F2',
       'BsmtUnfSF', 'TotalBsmtSF', '1stFlrSF', '2ndFlrSF', 'LowQualFinSF'
,
       'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath', 'HalfBath
',
       'BedroomAbvGr', 'KitchenAbvGr', 'TotRmsAbvGrd', 'Fireplaces',
       'GarageYrBlt', 'GarageCars', 'GarageArea', 'WoodDeckSF', 'OpenPorc
hSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'MiscVal'
,
       'MoSold', 'YrSold'],
      dtype='object')
```

In [49]:
```
skewed_feats = train[numeric_feats].apply(lambda x: skew(x.dropna())) #co
skewed_feats = skewed_feats[skewed_feats > 0.75]
skewed_feats = skewed_feats.index

all_data[skewed_feats] = np.log1p(all_data[skewed_feats])
```

In [50]:
```
all_data[skewed_feats]
```

Out[50]:

	MSSubClass	LotFrontage	LotArea	MasVnrArea	BsmtFinSF1	BsmtFinSF2	BsmtU
0	4.110874	4.189655	9.042040	5.283204	6.561031	0.0	5.01
1	3.044522	4.394449	9.169623	0.000000	6.886532	0.0	5.65
2	4.110874	4.234107	9.328212	5.093750	6.188264	0.0	6.07
3	4.262680	4.110874	9.164401	0.000000	5.379897	0.0	6.29
4	4.110874	4.442651	9.565284	5.860786	6.486161	0.0	6.19
...	...	...	...	...	...	...	
1454	5.081404	3.091042	7.568896	0.000000	0.000000	0.0	6.30
1455	5.081404	3.091042	7.546974	0.000000	5.533389	0.0	5.68
1456	3.044522	5.081404	9.903538	0.000000	7.110696	0.0	0.00
1457	4.454347	4.143135	9.253591	0.000000	5.823046	0.0	6.35
1458	4.110874	4.317488	9.172431	4.553877	6.632002	0.0	5.47

2919 rows × 21 columns

In [51]:
```
all_data = pd.get_dummies(all_data)
```

In [52]:
```
#filling NA's with the mean of the column:
all_data = all_data.fillna(all_data.mean())
```

In [53]:
```
#creating matrices for sklearn:
X_train = all_data[:train.shape[0]]
X_test = all_data[train.shape[0]:]
y = train.SalePrice
```

In [54]:
```
all_data
```

Out[54]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRem
0	4.110874	4.189655	9.042040	7	5	2003	
1	3.044522	4.394449	9.169623	6	8	1976	
2	4.110874	4.234107	9.328212	7	5	2001	
3	4.262680	4.110874	9.164401	7	5	1915	
4	4.110874	4.442651	9.565284	8	5	2000	
...	...	...	...	...	...	...	
1454	5.081404	3.091042	7.568896	4	7	1970	
1455	5.081404	3.091042	7.546974	4	5	1970	
1456	3.044522	5.081404	9.903538	5	7	1960	
1457	4.454347	4.143135	9.253591	5	5	1992	
1458	4.110874	4.317488	9.172431	7	5	1993	

2919 rows × 288 columns

In [55]:
```
from sklearn.linear_model import Ridge, RidgeCV, ElasticNet, LassoCV, Las
from sklearn.model_selection import cross_val_score

def rmse_cv(model):
    rmse= np.sqrt(-cross_val_score(model, X_train, y, scoring="neg_mean_s
    return(rmse)
```

In [56]:
```
model_ridge = Ridge()
```

In [57]:
```
model_ridge
```

Out[57]: Ridge()

In [58]:
```
alphas=[0.1]
cv_ridge = [rmse_cv(Ridge(alpha =alpha)).mean()
            for alpha in alphas]
```

RMSE Value - Ridge

In [59]:
```
cv_ridge
```

Out[59]: [0.13777538277187865]

RMSE Value - Lasso

In [60]:
```
model_lasso = LassoCV(alphas = [0.1]).fit(X_train, y)
```

In [61]:
```
rmse_cv(model_lasso).mean()
```

Out[61]: 0.20921930047608214

Obseerved that ridge performed better for alpha

Optimizing Alpha

In [62]:
```
# For Ridge
alphas=np.arange(5, 15, 1).tolist() #running on alpha from 5 to 15 at a 1
cv_ridge = [rmse_cv(Ridge(alpha =alpha)).mean()
            for alpha in alphas]
```

In [63]:
```
min_index=cv_ridge.index(min(cv_ridge)) #min cv see for index
cv_ridge[min_index] #this is minimum cv score we could achieve
```

Out[63]: 0.12733734668670776

In [64]:
```
alphas[min_index] #min CV achieved at alpha = 10
```

Out[64]: 10

In [65]:
```
# For Lasso
alphas=np.arange(0, 0.001, 0.0001).tolist() #running on alpha from 0 to 0
lasso_cv=[]
alpha_curr=[]
for i in alphas:
    model_lasso_cv = rmse_cv(LassoCV(alphas=[i]).fit(X_train, y)).mean()
    alpha_curr.append(i)
    lasso_cv.append(model_lasso_cv)
```

```
/Users/vipulsahni/opt/anaconda3/lib/python3.9/site-packages/sklearn/linea
r_model/_coordinate_descent.py:633: UserWarning: Coordinate descent with
alpha=0 may lead to unexpected results and is discouraged.
  model = cd_fast.enet_coordinate_descent_gram(
/Users/vipulsahni/opt/anaconda3/lib/python3.9/site-packages/sklearn/linea
r_model/_coordinate_descent.py:633: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations. Duality ga
p: 4.859170575181508, tolerance: 0.018912592760396085
  model = cd_fast.enet_coordinate_descent_gram(
/Users/vipulsahni/opt/anaconda3/lib/python3.9/site-packages/sklearn/linea
r_model/_coordinate_descent.py:633: UserWarning: Coordinate descent with
```

```
r_model/_coordinate_descent.py:647: ConvergenceWarning: Objective did not
converge. You might want to increase the number of iterations, check the
scale of the features or consider increasing regularisation. Duality gap:
5.392e-01, tolerance: 1.800e-02
  model = cd_fast.enet_coordinate_descent(
```

In [66]:

```
min_index=lasso_cv.index(min(lasso_cv)) #min cv see for index
lasso_cv[min_index] #this is minimum cv score we could achieve with lasso
```

Out[66]: 0.12256735885048128

In [67]:

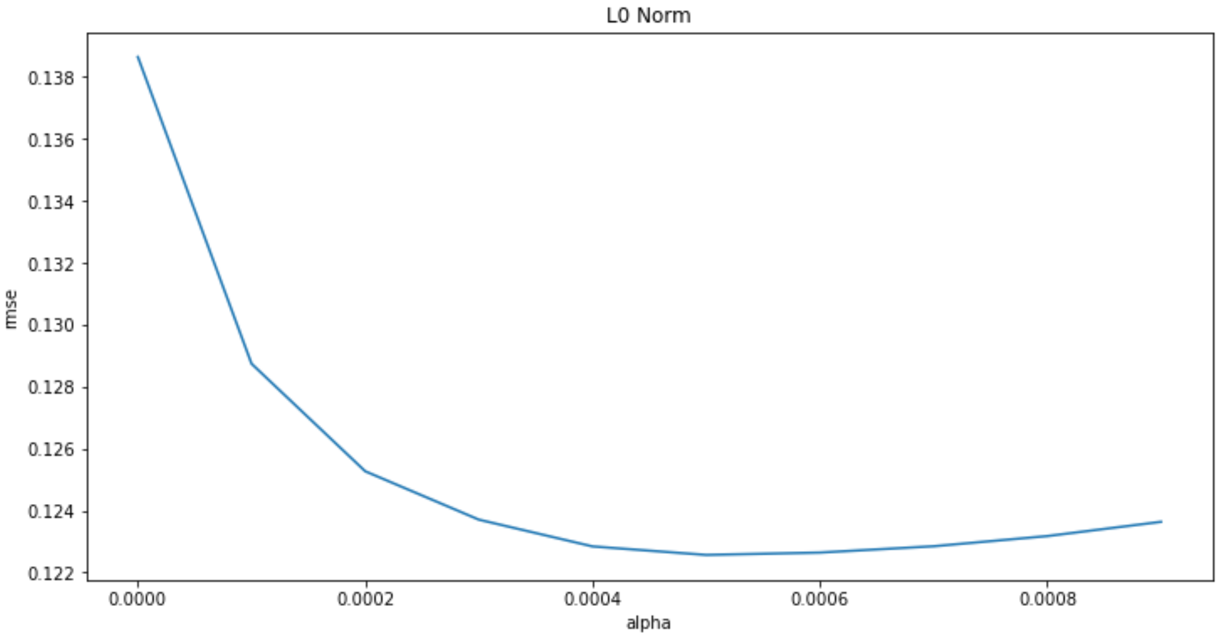
```
alpha_curr[min_index] #min CV achieved at alpha = 0.0005
```

Out[67]: 0.0005

In [68]:

```
#L0 Norm of the coefficients that lasso produces
lasso_cv = pd.Series(lasso_cv, index = alphas)
lasso_cv.plot(title = "L0 Norm")
plt.xlabel("alpha")
plt.ylabel("rmse")
```

Out[68]: Text(0, 0.5, 'rmse')



For Ridge, we were able to optimize alpha = 10, to get RMSE CV score of 0.127

For Lasso, alpha = 0.0005 to get RMSE CV score of 0.1225 - lower than Ridge

In [29]:

```
#Now checking predictions - Lasso
model_lasso = LassoCV(alphas = [0.0005]).fit(X_train, y)
model_lasso.predict(X_test)
```

Out[29]:
```
array([11.69490559, 11.92823243, 12.10183291, ..., 12.03779924,
       11.68640318, 12.33887195])
```

In [30]:
```
#Now checking predictions - Ridge
model_ridge = Ridge(alpha=0.1).fit(X_train, y) #0.1 because submitting to
```

In [31]:
```
pred_ridge=model_ridge.predict(X_test)
```

In [32]:
```
pred_ridge_df=pd.DataFrame(np.expm1(pred_ridge))
```

In [33]:
```
X_test['SalePrice']=pred_ridge_df[0]
X_test['Id'] = X_test.index
X_test['Id'] += 1461
```

```
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_18452/25514226
27.py:1: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test['SalePrice']=pred_ridge_df[0]
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_18452/25514226
27.py:2: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test['Id'] = X_test.index
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_18452/25514226
27.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_test['Id'] += 1461
```

Submission score=0.135

In [34]:
```
X_test[['Id','SalePrice']].to_csv('submission.csv') #0.135
```

Add features to ridge regression | Ensembling and Stacking

In [69]:
```
model_ridge = Ridge(alpha=10).fit(X_train, y)
model_lasso = LassoCV(alphas = [0.0005]).fit(X_train, y)
X_train['Lasso_val'] = model_lasso.predict(X_train)
X_train['Ridge_val'] = model_ridge.predict(X_train.loc[:,'MSSubClass':'Sa
```

```
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_18452/33504323
49.py:3: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Lasso_val'] = model_lasso.predict(X_train)
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_18452/33504323
49.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X_train['Ridge_val'] = model_ridge.predict(X_train.loc[:,'MSSubClass':'
SaleCondition_Partial'])
```

In [70]:

```
X_train
```

Out[70]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRem
0	4.110874	4.189655	9.042040	7	5	2003	
1	3.044522	4.394449	9.169623	6	8	1976	
2	4.110874	4.234107	9.328212	7	5	2001	
3	4.262680	4.110874	9.164401	7	5	1915	
4	4.110874	4.442651	9.565284	8	5	2000	
...	...	...	...	...	...	...	
1455	4.110874	4.143135	8.976894	6	5	1999	
1456	3.044522	4.454347	9.486152	6	6	1978	
1457	4.262680	4.204693	9.109746	7	9	1941	
1458	3.044522	4.234107	9.181735	5	6	1950	
1459	3.044522	4.330733	9.204121	5	6	1965	

1460 rows × 290 columns

In [165…

```
X_train.head()
```

Out[165]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemod
0	4.110874	4.189655	9.042040	7	5	2003	2
1	3.044522	4.394449	9.169623	6	8	1976	1
2	4.110874	4.234107	9.328212	7	5	2001	2
3	4.262680	4.110874	9.164401	7	5	1915	1
4	4.110874	4.442651	9.565284	8	5	2000	2

5 rows × 290 columns

In [168…

```
# Check Ridge train predictions
model_ridge = Ridge(alpha=10).fit(X_train, y)
```

In [175…

```
rmse_cv(model_ridge).mean() #RMSE is improved, earlier it was 0.127
```

Out[175]: 0.12267213549556055

XG Boost

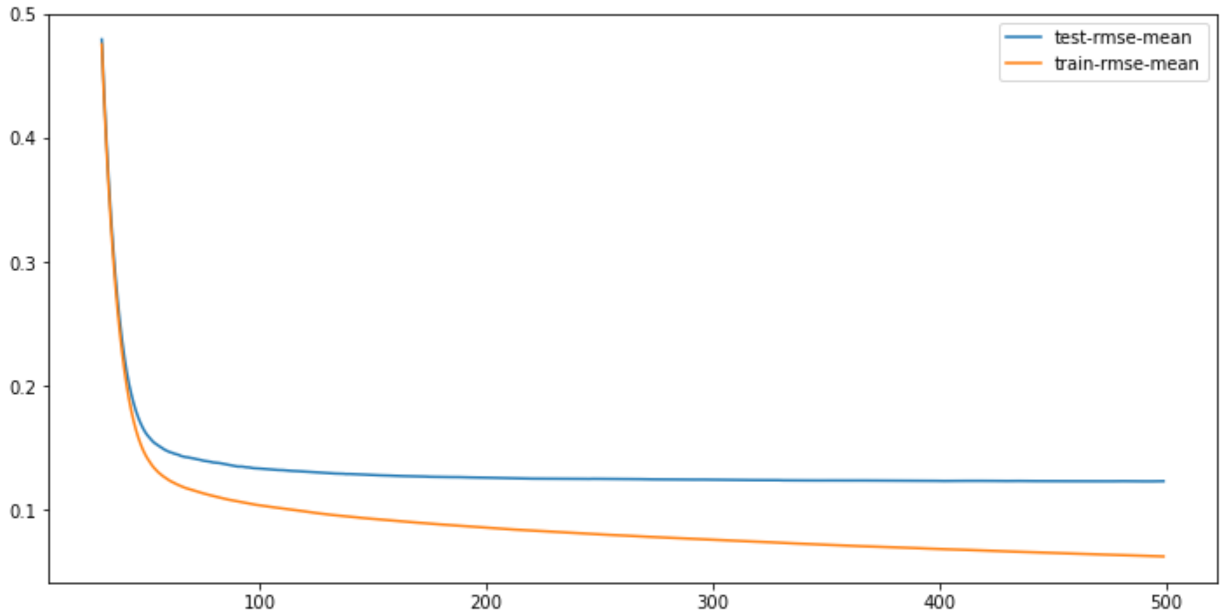
In [198…

```
dtrain = xgb.DMatrix(X_train.loc[:,'MSSubClass':'SaleCondition_Partial'],
dtest = xgb.DMatrix(X_test)

params = {"max_depth":2, "eta":0.1}
model = xgb.cv(params, dtrain,  num_boost_round=500, early_stopping_round
```

In [199…

```
model.loc[30:,["test-rmse-mean", "train-rmse-mean"]].plot()
```

Out[199]: <AxesSubplot:>



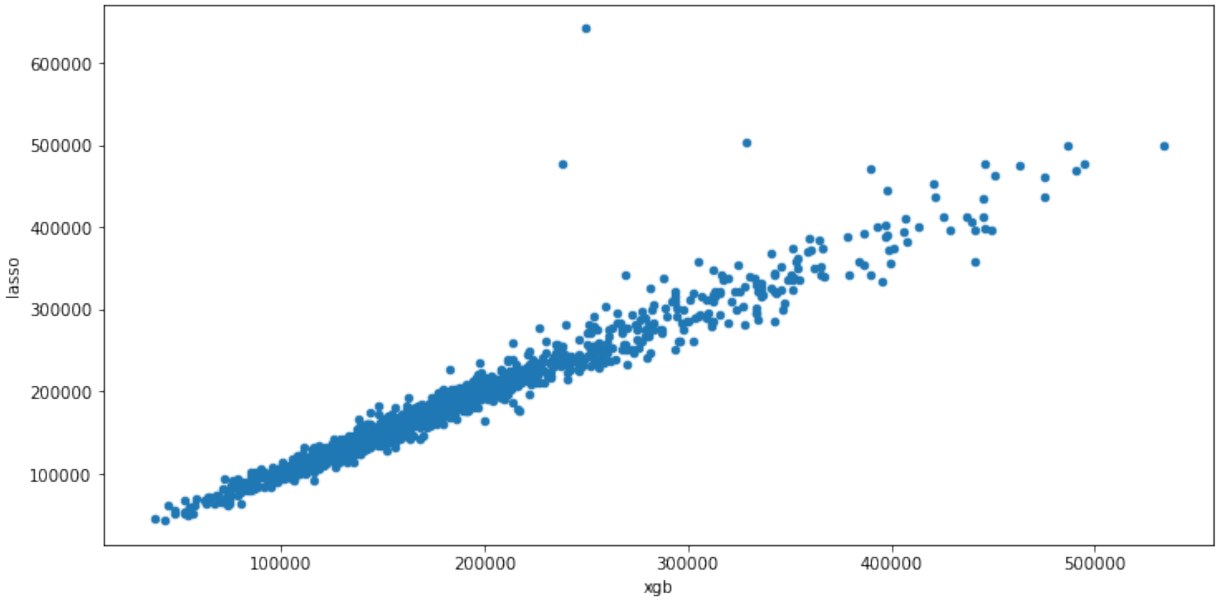
In [200…
```
model_xgb = xgb.XGBRegressor(n_estimators=360, max_depth=2, learning_rate
model_xgb.fit(X_train.loc[:,'MSSubClass':'SaleCondition_Partial'], y)
```

Out[200]:
```
XGBRegressor(base_score=0.5, booster='gbtree', callbacks=None,
             colsample_bylevel=1, colsample_bynode=1, colsample_bytree=1
,
             early_stopping_rounds=None, enable_categorical=False,
             eval_metric=None, gamma=0, gpu_id=-1, grow_policy='depthwis
e',
             importance_type=None, interaction_constraints='',
             learning_rate=0.1, max_bin=256, max_cat_to_onehot=4,
             max_delta_step=0, max_depth=2, max_leaves=0, min_child_weig
ht=1,
             missing=nan, monotone_constraints='()', n_estimators=360, n
_jobs=0,
             num_parallel_tree=1, predictor='auto', random_state=0, reg_
alpha=0,
             reg_lambda=1, ...)
```

In [204…
```
model_lasso = LassoCV(alphas = [0.0005]).fit(X_train.loc[:,'MSSubClass':'
xgb_preds = np.expm1(model_xgb.predict(X_test))
lasso_preds = np.expm1(model_lasso.predict(X_test))
```

In [205…
```
predictions = pd.DataFrame({"xgb":xgb_preds, "lasso":lasso_preds})
predictions.plot(x = "xgb", y = "lasso", kind = "scatter")
```

Out[205]: <AxesSubplot:xlabel='xgb', ylabel='lasso'>



Using only xgb_preds, model is overfitting. On training RMSE is ~0.1, but on test it is 0.13

In [212…
```
rmse_cv(model_xgb).mean() #RMSE is improved, earlier it was 0.127
```

Out[212]: 0.10913110265345634

In [206…

```
preds = 0.7*lasso_preds + 0.3*xgb_preds
```

In [214…

```
solution = pd.DataFrame({"id":test.Id, "SalePrice":preds})
# solution.to_csv("ridge_sol.csv", index = False)
```

In [215…

```
solution.to_csv('submission.csv') #Score: 0.12299
```

Improve upon this XGboost model

Feature Engineering

In [217…

```
all_data['GarageYrBltn'] = abs(all_data['YrSold'] - all_data['GarageYrBlt
all_data['YearRemodAddn'] = abs(all_data['YrSold'] - all_data['YearRemodA
all_data['YearBuiltn'] = abs(all_data['YrSold'] - all_data['YearBuilt'])
all_data['SF'] = all_data['1stFlrSF']+all_data['2ndFlrSF']+all_data['Tota
```

```
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_86807/35110614
70.py:1: PerformanceWarning: DataFrame is highly fragmented.  This is usu
ally the result of calling `frame.insert` many times, which has poor perf
ormance.  Consider joining all columns at once using pd.concat(axis=1) in
stead. To get a de-fragmented frame, use `newframe = frame.copy()`
  all_data['GarageYrBltn'] = abs(all_data['YrSold'] - all_data['GarageYrB
lt'])
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_86807/35110614
70.py:2: PerformanceWarning: DataFrame is highly fragmented.  This is usu
ally the result of calling `frame.insert` many times, which has poor perf
ormance.  Consider joining all columns at once using pd.concat(axis=1) in
stead. To get a de-fragmented frame, use `newframe = frame.copy()`
  all_data['YearRemodAddn'] = abs(all_data['YrSold'] - all_data['YearRemo
dAdd'])
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_86807/35110614
70.py:3: PerformanceWarning: DataFrame is highly fragmented.  This is usu
ally the result of calling `frame.insert` many times, which has poor perf
ormance.  Consider joining all columns at once using pd.concat(axis=1) in
stead. To get a de-fragmented frame, use `newframe = frame.copy()`
  all_data['YearBuiltn'] = abs(all_data['YrSold'] - all_data['YearBuilt']
)
/var/folders/6p/fvv3r7rj335gs5y3bm1f750m0000gn/T/ipykernel_86807/35110614
70.py:4: PerformanceWarning: DataFrame is highly fragmented.  This is usu
ally the result of calling `frame.insert` many times, which has poor perf
ormance.  Consider joining all columns at once using pd.concat(axis=1) in
stead. To get a de-fragmented frame, use `newframe = frame.copy()`
  all_data['SF'] = all_data['1stFlrSF']+all_data['2ndFlrSF']+all_data['To
talBsmtSF']+all_data['GrLivArea']+all_data['HalfBath']+all_data['FullBath
']
```

In [218…

```
all_data
```

Out[218]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRer
0	4.110874	4.189655	9.042040	7	5	2003	
1	3.044522	4.394449	9.169623	6	8	1976	
2	4.110874	4.234107	9.328212	7	5	2001	
3	4.262680	4.110874	9.164401	7	5	1915	
4	4.110874	4.442651	9.565284	8	5	2000	
...	...	...	...	...	...	...	
1454	5.081404	3.091042	7.568896	4	7	1970	
1455	5.081404	3.091042	7.546974	4	5	1970	
1456	3.044522	5.081404	9.903538	5	7	1960	
1457	4.454347	4.143135	9.253591	5	5	1992	
1458	4.110874	4.317488	9.172431	7	5	1993	

2919 rows × 292 columns

In [223...

```
scaler = StandardScaler()
scaler.fit(all_data)
all_data = pd.DataFrame(scaler.transform(all_data), index = all_data.inde
```

In [224...

```
all_data
```

Out[224]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearR
0	0.419418	-0.020358	-0.103719	0.646183	-0.507284	1.046258	
1	-1.120845	0.619103	0.146544	-0.063185	2.188279	0.154764	-
2	0.419418	0.118440	0.457629	0.646183	-0.507284	0.980221	
3	0.638691	-0.266348	0.136301	0.646183	-0.507284	-1.859351	-
4	0.419418	0.769612	0.922662	1.355551	-0.507284	0.947203	
...	...	...	...	...	...	...	
1454	1.821276	-3.450727	-2.993401	-1.481920	1.289758	-0.043346	-
1455	1.821276	-3.450727	-3.036401	-1.481920	-0.507284	-0.043346	-
1456	-1.120845	2.764091	1.586172	-0.772552	1.289758	-0.373528	
1457	0.915540	-0.165615	0.311255	-0.772552	-0.507284	0.683057	
1458	0.419418	0.378796	0.152052	0.646183	-0.507284	0.716075	

2919 rows × 292 columns

In [225…
```
df_train = all_data.iloc[:1460,:]
df_test = all_data.iloc[1460:,:]
final = pd.concat([df_train,y], axis = 1)
```

In [226…
```
final
```

Out[226]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearR
0	0.419418	-0.020358	-0.103719	0.646183	-0.507284	1.046258	
1	-1.120845	0.619103	0.146544	-0.063185	2.188279	0.154764	-
2	0.419418	0.118440	0.457629	0.646183	-0.507284	0.980221	
3	0.638691	-0.266348	0.136301	0.646183	-0.507284	-1.859351	-
4	0.419418	0.769612	0.922662	1.355551	-0.507284	0.947203	
...	...	...	...	...	...	...	
1455	0.419418	-0.165615	-0.231508	-0.063185	-0.507284	0.914184	
1456	-1.120845	0.806133	0.767440	-0.063185	0.391237	0.220801	
1457	0.638691	0.026597	0.029092	0.646183	3.086800	-1.000876	
1458	-1.120845	0.118440	0.170303	-0.772552	0.391237	-0.703711	
1459	-1.120845	0.420154	0.214215	-0.772552	0.391237	-0.208437	-

1460 rows × 293 columns

In [231…
```
from sklearn.linear_model import BayesianRidge, HuberRegressor, Ridge, Or
from sklearn.ensemble import GradientBoostingRegressor
from catboost import CatBoostRegressor
```

In [270…
```
baseline_model = GradientBoostingRegressor()
baseline_model.fit(df_train, y)
```

Out[270]: GradientBoostingRegressor()

In [287…
```
br_params = {
    'n_iter': 304,
    'tol': 0.16864712769300896,
    'alpha_1': 5.589616542154059e-07,
    'alpha_2': 9.799343618469923,
    'lambda_1': 1.7735725582463822,
    'lambda_2': 3.616928181181732e-06
}


ridge_params = {
    'alpha': 10
}
```

In [295…
```
models = {'gbr':GradientBoostingRegressor(),
          'br':BayesianRidge(**br_params),
          'ridge':Ridge(**ridge_params),
          'catboost':CatBoostRegressor(loss_function='RMSE',n_estimators=
```

In [274…
```
for name, model in models.items():
    model.fit(df_train, y)
```

In [275…
```
results = {}

kf = KFold(n_splits=10)

for name, model in models.items():
    result = np.exp(np.sqrt(-cross_val_score(model, df_train, y, scoring=
    results[name] = result
```

In [276…
```
for name, result in results.items():
    print("----------\n" + name)
    print(np.mean(result))
    print(np.std(result))
```

```
----------
gbr
1.1334937793062636
0.02041514019070494
----------
br
1.136720968542448
0.02708918872877562
----------
ridge
1.1377247226437417
0.025916296541081737
----------
catboost
1.1237199881446363
0.02114096999987419
```

In [291…
```
y_pred = (
    0.0 * np.exp(models['gbr'].predict(df_test)) +
    0.0 * np.exp(models['br'].predict(df_test)) +
    0.0 * np.exp(models['ridge'].predict(df_test))+
    1 * np.exp(models['catboost'].predict(df_test)))
```

In [292…
```
solution = pd.DataFrame({"id":test.Id, "SalePrice":y_pred})
# solution.to_csv("ridge_sol.csv", index = False)
```

In [293…
```
solution.to_csv('submission.csv',index=False) #Score: 0.12299
```

In [294…
```
solution
```

Out[294]:

	id	SalePrice
0	1461	125953.825089
1	1462	161424.964322
2	1463	187908.591855
3	1464	196804.694788
4	1465	183519.623195
...	...	...
1454	2915	80918.325435
1455	2916	78242.653140
1456	2917	160394.307425
1457	2918	113555.787541
1458	2919	224401.442934

1459 rows × 2 columns

Using ensemble model with some feature engineering gives us better result at public leaderboard 0.124, than by using just xgboost 0.132