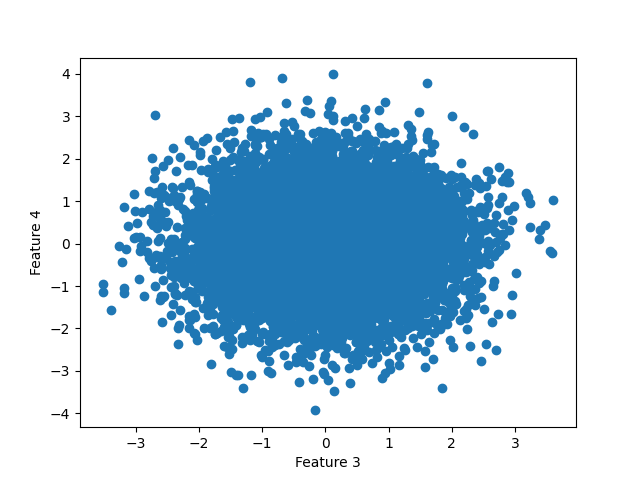
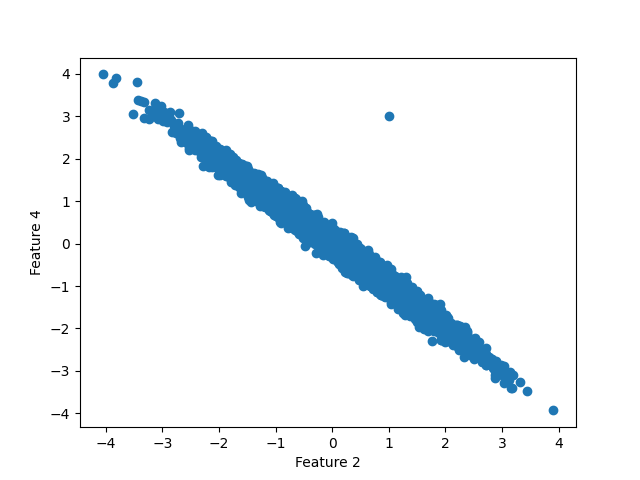
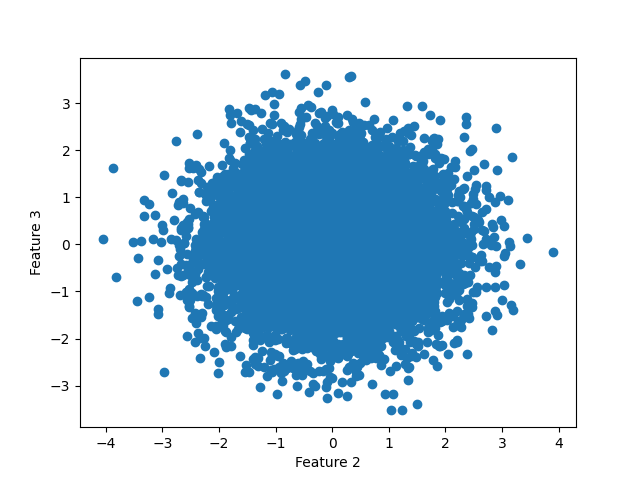
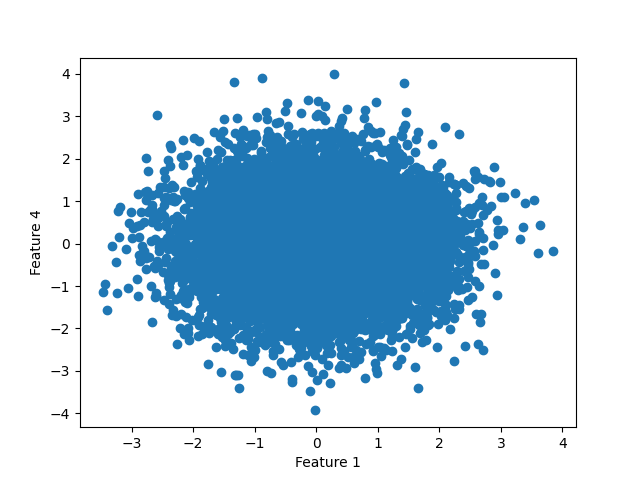
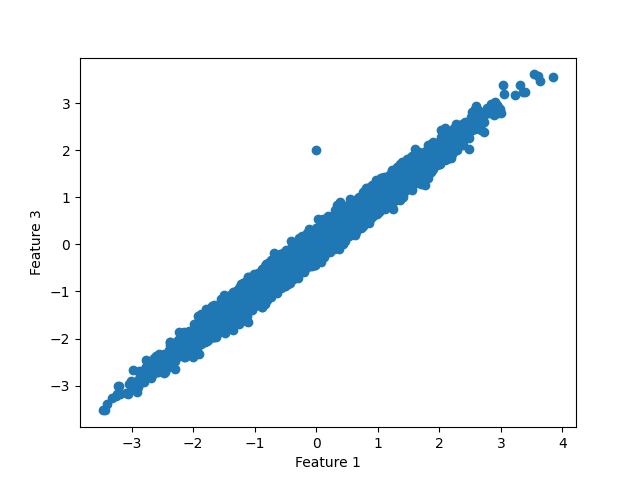
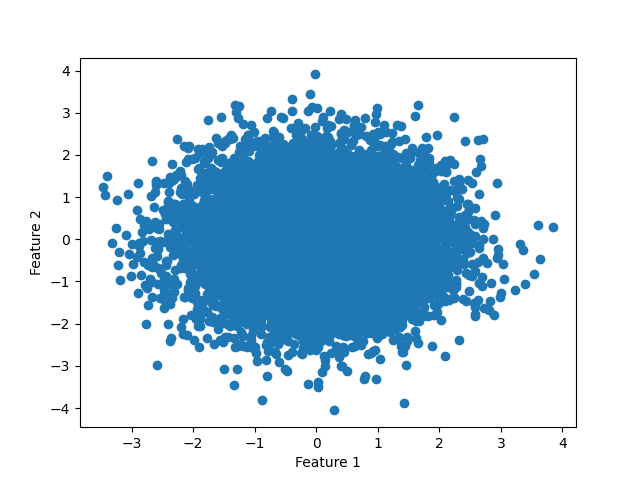
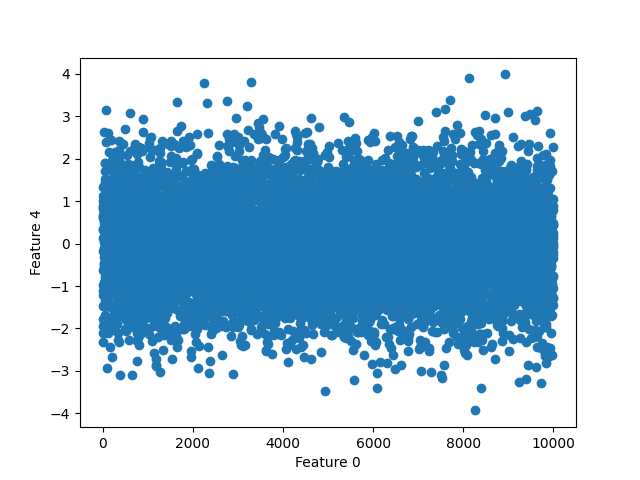
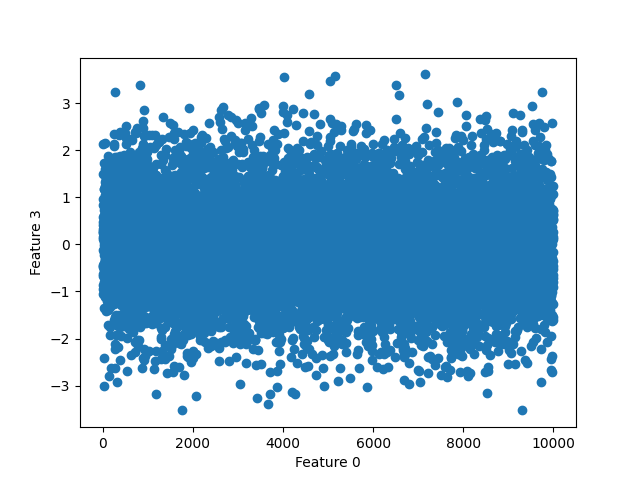
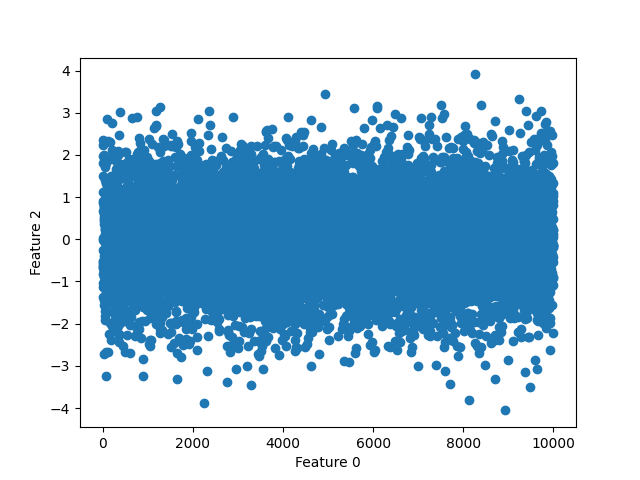
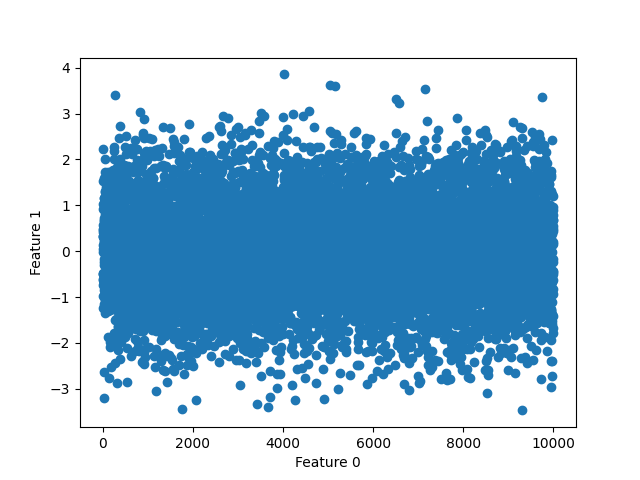
**Lab 2**

Problem 1

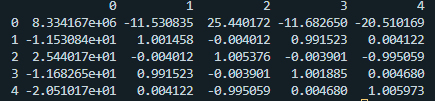
1)



As seen be the figure given columns (0,1,2,3,4) columns 1 and 3 are positively correlated and columns 2 and 4 are negatively correlated.

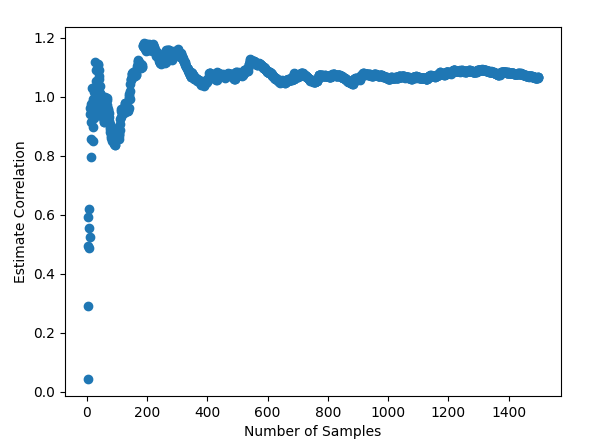
2)

|  | 0 | 1 | 2 | 3 |
| --- | --- | --- | --- | --- |
| 0 | var(0) | cov(0,1) | cov(0,2) | cov(0,3) |
| 1 | cov(1,0) | var(1) | cov(1,2) | cov(1,3) |
| 2 | cov(2,0) | cov(2,1) | var(2) | cov(2,3) |
| 3 | cov(3,0) | cov(3,1) | cov(3,2) | var(3) |

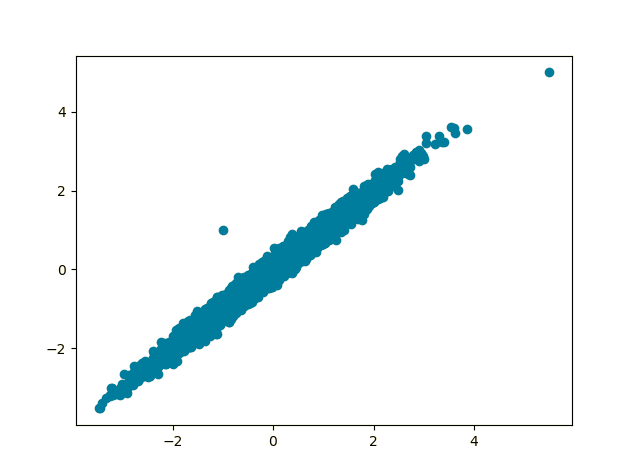


The covariance matrix aligns with the plots above with feature 1 and feature 3 having a strong correlation which is positive and feature 2 and feature 4 having a strong correlation which is negative. The remaining features near zero correlations.

3)



Problem 2



According to hint: The covariance matrix of z = (the covariance matrix of y)(Q)( and since we know from the hint that the covariance matrix of y is a 2x2 identity matrix, the covariance matrix of z=(Q)(. Since if given z then y = . So we simply need to get for ourselves using the eigenvalues and eigenvectors of z. Using eigendecomposition we can say that the covariance matrix of z = (the eigenvectors)(the eigenvalues)( = v*λ* As v is a diagonal matrix and *λ* is orthogonal then so since cov(z) = v*λ* = (v*)(*then= = *(*This is helpful as we want our transformed values to have an identity covariance matrix so that the values are decorrelated. So according to the equations above we multiply the data by the inverse square root of the eigenvalues and the transpose of the eigenvectors in a process known as whitening or sphereing

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.read\_csv("DF2", index\_col = 0)

cov = df.cov()

print(cov)

eigval,eigvec=np.linalg.eig(cov)

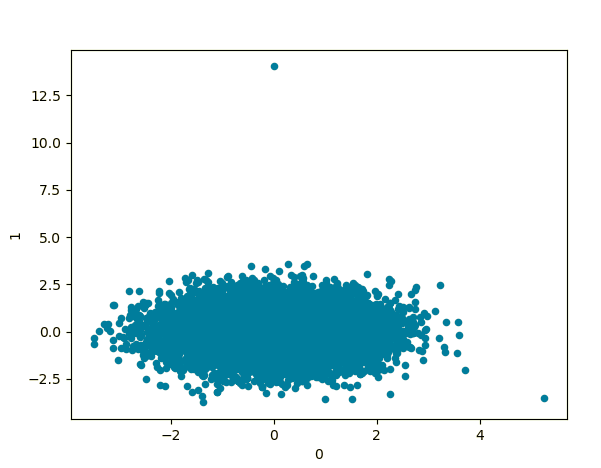
Q=np.dot(np.diag(1/np.sqrt(eigval)),eigvec.T)

white=Q@df.T

white = white.T

white.plot(x = 0, y = 1, kind= "scatter")

plt.show()



Problem 3

1)

year = input("Enter a year: ")

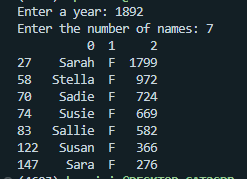
k = input("Enter the number of names: ")

df = pd.read\_csv("Names/yob%i.txt" % int(year),delimiter=",",header=None)

namesWithS = df[df[df.columns[0]].str.startswith("S")]

namesWithS.sort\_values(by=[df.columns[2]])

print(namesWithS[:int(k)])



2)

*# Part 2*

df = pd.read\_csv("Names/yob1880.txt",delimiter=",",header=None)

for index in range(1881,2016):

tempDf = pd.read\_csv("Names/yob%i.txt" % int(index),delimiter=",",header=None)

df = pd.concat([df,tempDf])

df = df.groupby([df.columns[0],df.columns[1]])[df.columns[2]].sum().reset\_index()

name = input("Enter a name: ")

print("The number of men with the name %s is: %i" % (name,df.iloc[df.index[(df[df.columns[0]] == name) & (df[df.columns[1]] == "M")]][2]))

print("The number of men with the name %s is: %i" % (name,df.iloc[df.index[(df[df.columns[0]] == name) & (df[df.columns[1]] == "F")]][2]))

bestLetterCount, bestLetter = 0,"A"

for letter in ascii\_uppercase:

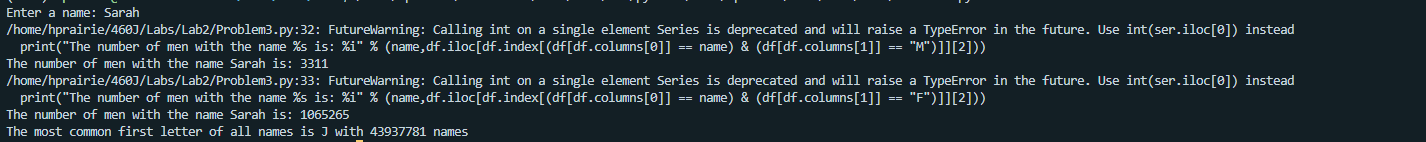
tempTotal = df[df[df.columns[0]].str.startswith(letter)][df.columns[2]].sum()

if(tempTotal > bestLetterCount):

bestLetter = letter

bestLetterCount = tempTotal

print("The most common first letter of all names is %s with %i names" % (bestLetter,bestLetterCount))



3)

name = input("Enter a name: ")

df = pd.read\_csv("Names/yob1880.txt",delimiter=",",header=None)

rel\_freq\_female = []

rel\_freq\_male = []

for index in range(1880,2016):

df = pd.read\_csv("Names/yob%i.txt" % int(index),delimiter=",",header=None)

female\_count = df.iloc[df.index[(df[df.columns[0]] == name) & (df[df.columns[1]] == "F")]]

male\_count = df.iloc[df.index[(df[df.columns[0]] == name) & (df[df.columns[1]] == "M")]]

if len(female\_count) == 0:

female\_count = 0

else:

female\_count = int(female\_count[2])/df[df.columns[2]].sum()

if len(male\_count) == 0:

male\_count = 0

else:

male\_count = int(male\_count[2])/df[df.columns[2]].sum()

rel\_freq\_female.append(female\_count)

rel\_freq\_male.append(male\_count)

years = range(1880,2016)

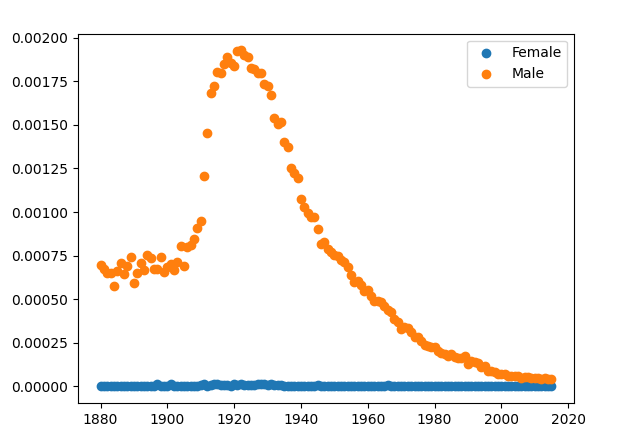
plt.scatter(years,rel\_freq\_female,label="Female")

plt.scatter(years,rel\_freq\_male,label="Male")

plt.legend(loc="upper right")

plt.show()

When running on the name bernard we get the following.



4)

5)

year = input("Enter a year: ")

df = pd.read\_csv("Names/yob%i.txt" % int(year),delimiter=",",header=None)

df2 = pd.read\_csv("Names/yob%i.txt" % (int(year) - 1),delimiter=",",header=None)

dfGrouped = df.groupby(df.columns[0])[df.columns[2]].sum().reset\_index()

df2Grouped = df2.groupby(df.columns[0])[df.columns[2]].sum().reset\_index()

results = pd.DataFrame(columns=["Name","Frequency"])

for index, row in dfGrouped.iterrows():

previous = df2Grouped.loc[df2Grouped[df2Grouped.columns[0]] == row[0]]

if len(previous) == 0:

previous = 0

else:

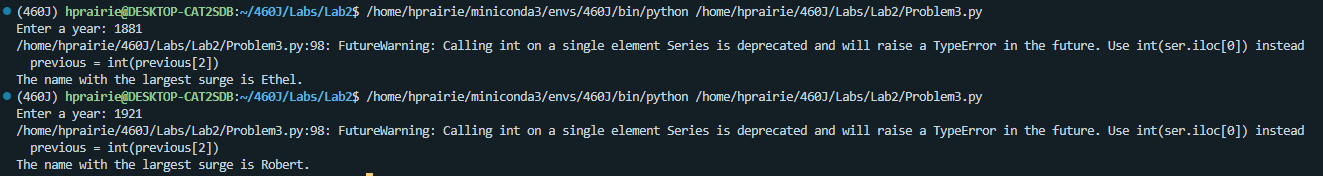
previous = int(previous[2])

results = pd.concat([results,pd.DataFrame([[row[0], row[2]-previous]],columns=["Name","Frequency"])])

results = results.reset\_index()

maxid = results["Frequency"].idxmax()

print("The name with the largest surge is %s." % results.iloc[maxid]["Name"])



Problem 4

2)

*# Import the data*

test = pd.read\_csv("HousePriceData/test.csv",delimiter=',')

train = pd.read\_csv("HousePriceData/train.csv", delimiter=',')

all\_data = pd.concat((train.loc[:,'MSSubClass':'SaleCondition'],test.loc[:,'MSSubClass':'SaleCondition']))

*# Data preprocessing*

*# Log transform the sale price*

train["SalePrice"] = np.log1p(train["SalePrice"])

*# Log transform skewed numeric values*

numeric\_feats = all\_data.dtypes[all\_data.dtypes != "object"].index

skewed\_feats = train[numeric\_feats].apply(lambda x: skew(x.dropna()))

skewed\_feats = skewed\_feats[skewed\_feats > 0.75]

skewed\_feats = skewed\_feats.index

all\_data[skewed\_feats] = np.log1p(all\_data[skewed\_feats])

*# Get dummies variables for data*

all\_data = pd.get\_dummies(all\_data)

*# Fill NaN with mean data*

all\_data = all\_data.fillna(all\_data.mean()) *# Chane this to trian mean.*

*#creating matrices for sklearn:*

X\_train = all\_data[:train.shape[0]]

X\_test = all\_data[train.shape[0]:]

y = train.SalePrice

*# Creating model and running prediction*

*model\_ridge = Ridge(alpha=0.1).fit(X\_train,y)*

*first\_pred = np.expm1(model\_ridge.predict(X\_test))*

*solution = pd.DataFrame({"id":test.Id, "SalePrice":first\_pred})*

*solution.to\_csv("first\_pred.csv", index = False)*



3)

def rmse\_cv(model):

rmse= np.sqrt(-cross\_val\_score(model, X\_train, y, scoring="neg\_mean\_squared\_error", cv = 5))

return(rmse)

*# RIDGE*

alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1, 3, 5,6,7,8,9,10,11,12,13,14,15, 30, 50, 75]

cv\_ridge = [rmse\_cv(Ridge(alpha = alpha)).mean() for alpha in alphas]

print(min(cv\_ridge))

cv\_ridge = pd.Series(cv\_ridge, index = alphas)

cv\_ridge.plot(title = "Validation - Just Do It")

plt.xlabel("alpha")

plt.ylabel("rmse")

plt.savefig("Ridge\_MSE\_Alpha.png")

plt.cla()

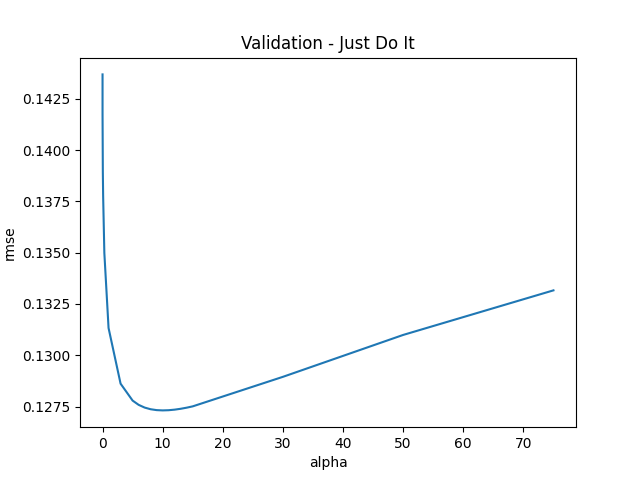
*# LASSO*

model\_lasso = LassoCV(alphas = [1, 0.1, 0.001, 0.0005]).fit(X\_train, y)

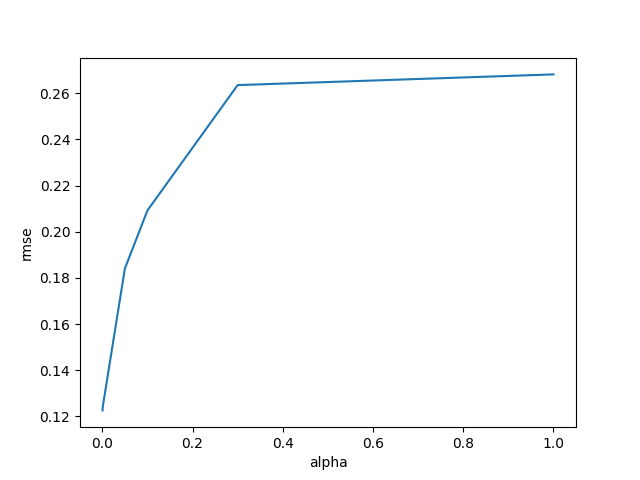
cv\_lasso = rmse\_cv(model\_lasso).mean()

print(cv\_lasso)



Ridge Plot

Lasso Plot



4)

alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1]

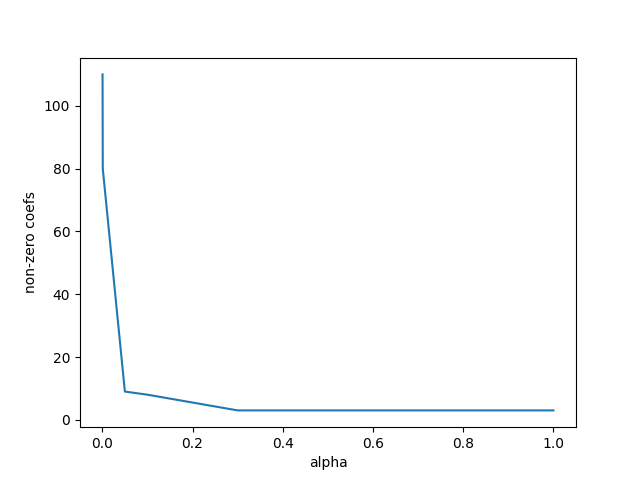
non\_zero = [sum(Lasso(alpha=alpha).fit(X\_train,y).coef\_ != 0) for alpha in alphas]

plt.plot(alphas,non\_zero)

plt.xlabel("alpha")

plt.ylabel("non-zero coefs")

plt.savefig("Lasso\_NZCoef\_Alpha.png")



5)

model\_lasso = Lasso(alpha=0.005).fit(X\_train,y)

model\_lasso\_data = model\_lasso.predict(X\_train)

model\_lasso\_data\_test = model\_lasso.predict(X\_test)

X\_train\_temp = np.column\_stack((X\_train,model\_lasso\_data))

X\_test\_temp = np.column\_stack((X\_test,model\_lasso\_data\_test))

model\_ridge = Ridge(alpha=9).fit(X\_train,y)

model\_ridge\_data = model\_ridge.predict(X\_train)

model\_ridge\_data\_test = model\_ridge.predict(X\_test)

X\_train\_temp = np.column\_stack((X\_train\_temp,model\_ridge\_data))

X\_test\_temp = np.column\_stack((X\_test\_temp,model\_ridge\_data\_test))

def rmse\_cv2(model):

rmse= np.sqrt(-cross\_val\_score(model, X\_train\_temp, y, scoring="neg\_mean\_squared\_error", cv = 5))

return(rmse)

alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1, 3, 5,6,7,8,9,10,11,12,13,14,15, 30, 50, 75]

cv\_ridge = [rmse\_cv(Ridge(alpha = alpha)).mean() for alpha in alphas]

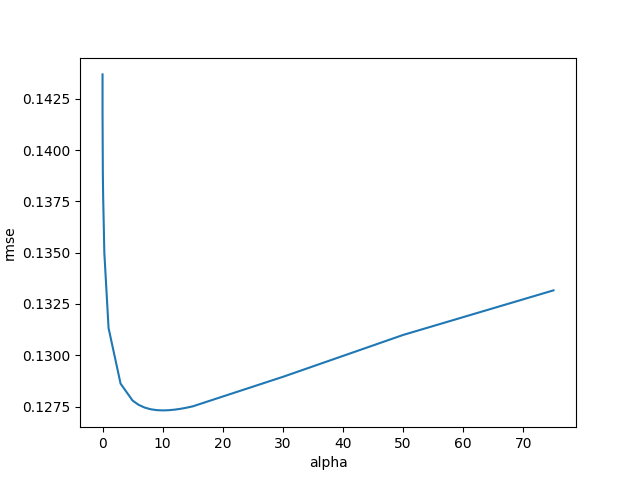
plt.plot(alphas,cv\_ridge)

plt.xlabel("alpha")

plt.ylabel("rmse")

plt.savefig("Ridge\_MSE\_2\_Alpha.png")

print("Best Ridge: %f" % min(cv\_ridge))



model\_ridge = Ridge(alpha=9).fit(X\_train\_temp,y)

ridge\_preds = np.expm1(model\_ridge.predict(X\_test\_temp))

solution = pd.DataFrame({"id":test.Id, "SalePrice":ridge\_preds})

solution.to\_csv("ridge\_stacking\_sol.csv", index = False)



6)

dtrain = xgb.DMatrix(X\_train,label=y)

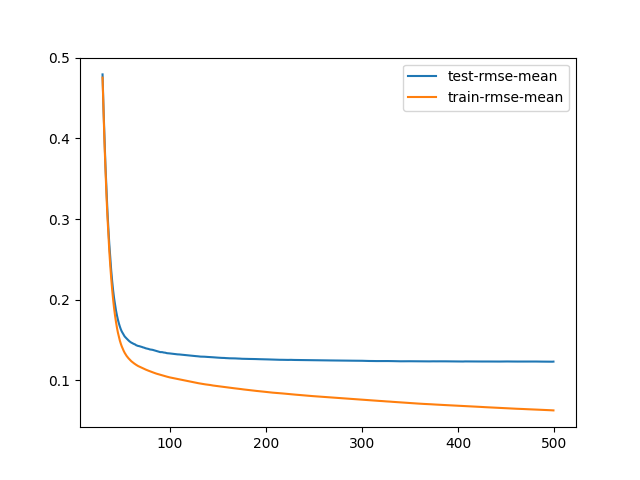
dtest = xgb.DMatrix(X\_test)

params = {"max\_depth":2, "eta":0.1}

model = xgb.cv(params, dtrain, num\_boost\_round=500, early\_stopping\_rounds=100)

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

plt.savefig("XGB\_Boost.png")



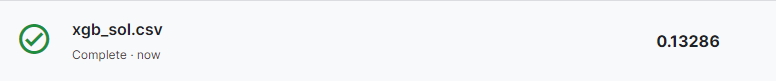
model\_xgb = xgb.XGBRegressor(n\_estimators=360, max\_depth=2, learning\_rate=0.1) *#the params were tuned using xgb.cv*

model\_xgb.fit(X\_train, y)

xgb\_preds = np.expm1(model\_xgb.predict(X\_test))

solution = pd.DataFrame({"id":test.Id, "SalePrice":xgb\_preds})

solution.to\_csv("xgb\_sol.csv", index = False)



7)

First we can attempt stacking the XGB predictions into the model and then running either a lasso or ridge prediction.

model\_lasso = Lasso(alpha=0.005).fit(X\_train,y)

model\_lasso\_data = model\_lasso.predict(X\_train)

model\_lasso\_data\_test = model\_lasso.predict(X\_test)

X\_train\_temp = np.column\_stack((X\_train,model\_lasso\_data))

X\_test\_temp = np.column\_stack((X\_test,model\_lasso\_data\_test))

model\_ridge = Ridge(alpha=9).fit(X\_train,y)

model\_ridge\_data = model\_ridge.predict(X\_train)

model\_ridge\_data\_test = model\_ridge.predict(X\_test)

X\_train\_temp = np.column\_stack((X\_train\_temp,model\_ridge\_data))

X\_test\_temp = np.column\_stack((X\_test\_temp,model\_ridge\_data\_test))

model\_xgb = xgb.XGBRegressor(n\_estimators=360, max\_depth=2, learning\_rate=0.1) *#the params were tuned using xgb.cv*

model\_xgb.fit(X\_train, y)

xgb\_preds\_train = np.expm1(model\_xgb.predict(X\_train))

xgb\_preds\_test = np.expm1(model\_xgb.predict(X\_test))

X\_train\_temp = np.column\_stack((X\_train\_temp,xgb\_preds\_train))

X\_test\_temp = np.column\_stack((X\_test\_temp,xgb\_preds\_test))

def rmse\_cv3(model):

rmse= np.sqrt(-cross\_val\_score(model, X\_train, y, scoring="neg\_mean\_squared\_error", cv = 5))

return(rmse)

alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1, 3, 5,6,7,8,9,10,11,12,13,14,15, 30, 50, 75]

cv\_ridge = [rmse\_cv3(Ridge(alpha = alpha)).mean() for alpha in alphas]

plt.plot(alphas,cv\_ridge)

plt.xlabel("alpha")

plt.ylabel("rmse")

plt.savefig("Ridge\_MSE\_3\_Alpha.png")

plt.show()

print("Best Ridge: %f" % min(cv\_ridge))

alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1, 3, 5,6,7,8,9,10,11,12,13,14,15, 30, 50, 75]

cv\_lasso = [rmse\_cv3(Lasso(alpha = alpha)).mean() for alpha in alphas]

plt.plot(alphas,cv\_lasso)

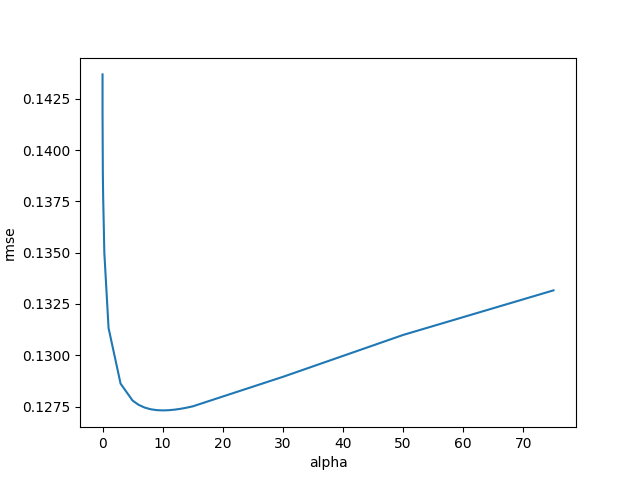
plt.xlabel("alpha")

plt.ylabel("rmse")

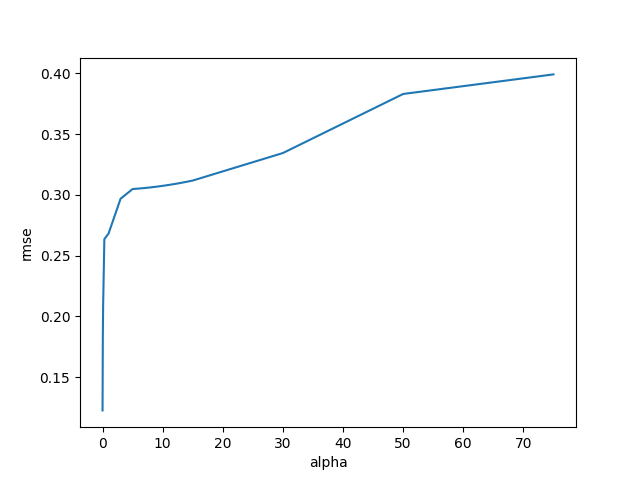
plt.savefig("Lasso\_MSE\_2\_Alpha.png")

plt.show()

print("Best Ridge: %f" % min(cv\_lasso))

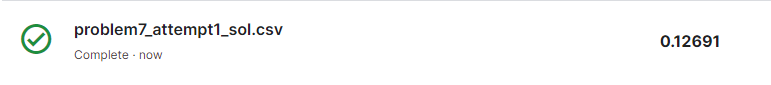
The graph for Ridge alphas is below and we see that an alpha of 8 is the best with a MSE of 0.127312.

The graph for Lasso is below and we can see that with an alpha of 0.0005 we get a better MSE.

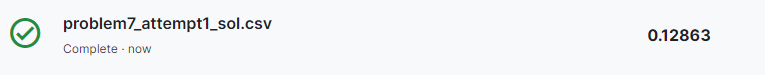




When we run the predictions on kaggle we get the following when using a Lasso Regression Model.



And we get the following when running a Ridge Regression model.



Not bad, but maybe we can get better. We got better results when stacking just the ridge and lasso before. Maybe instead of fitting it to another ridge we can fit it to a Lasso.

With a better ridge alpha we get the following.



And with the best lasso, we get the following.



These are slightly better. Now since these are independent, what if we took an average of their predictions.

model\_ridge = Ridge(alpha=8).fit(X\_train\_temp,y)

model\_lasso = Lasso(alpha=0.0005).fit(X\_train\_temp,y)

lasso\_preds = np.expm1(model\_lasso.predict(X\_test\_temp))

ridge\_preds = np.expm1(model\_ridge.predict(X\_test\_temp))



What if we also averaged in XGBoost?

model\_ridge = Ridge(alpha=8).fit(X\_train\_temp,y)

model\_lasso = Lasso(alpha=0.0005).fit(X\_train\_temp,y)

lasso\_preds = np.expm1(model\_lasso.predict(X\_test\_temp))

ridge\_preds = np.expm1(model\_ridge.predict(X\_test\_temp))

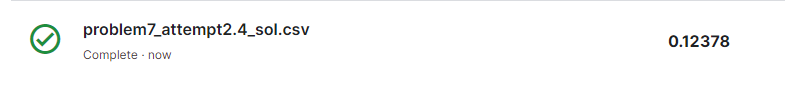
model\_xgb = xgb.XGBRegressor(n\_estimators=360, max\_depth=2, learning\_rate=0.1) *#the params were tuned using xgb.cv*

model\_xgb.fit(X\_train\_temp, y)

xgb\_preds = np.expm1(model\_xgb.predict(X\_test\_temp))

preds = (lasso\_preds+ridge\_preds+xgb\_preds)/3

With that we get our best score yet.



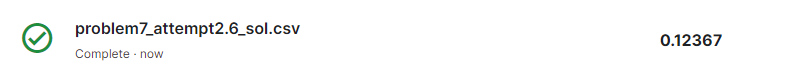
What if we weight it differently?

preds = (0.7)\*(lasso\_preds+ridge\_preds)/2 + xgb\_preds\*(0.3)



The weights of each prediction didnt change much but got worse when lasso and ridge took over. So what if we gave more weight to xgb?

preds = (0.6)\*(lasso\_preds+ridge\_preds)/2 + xgb\_preds\*(0.4)



With that we get the best prediction so far.