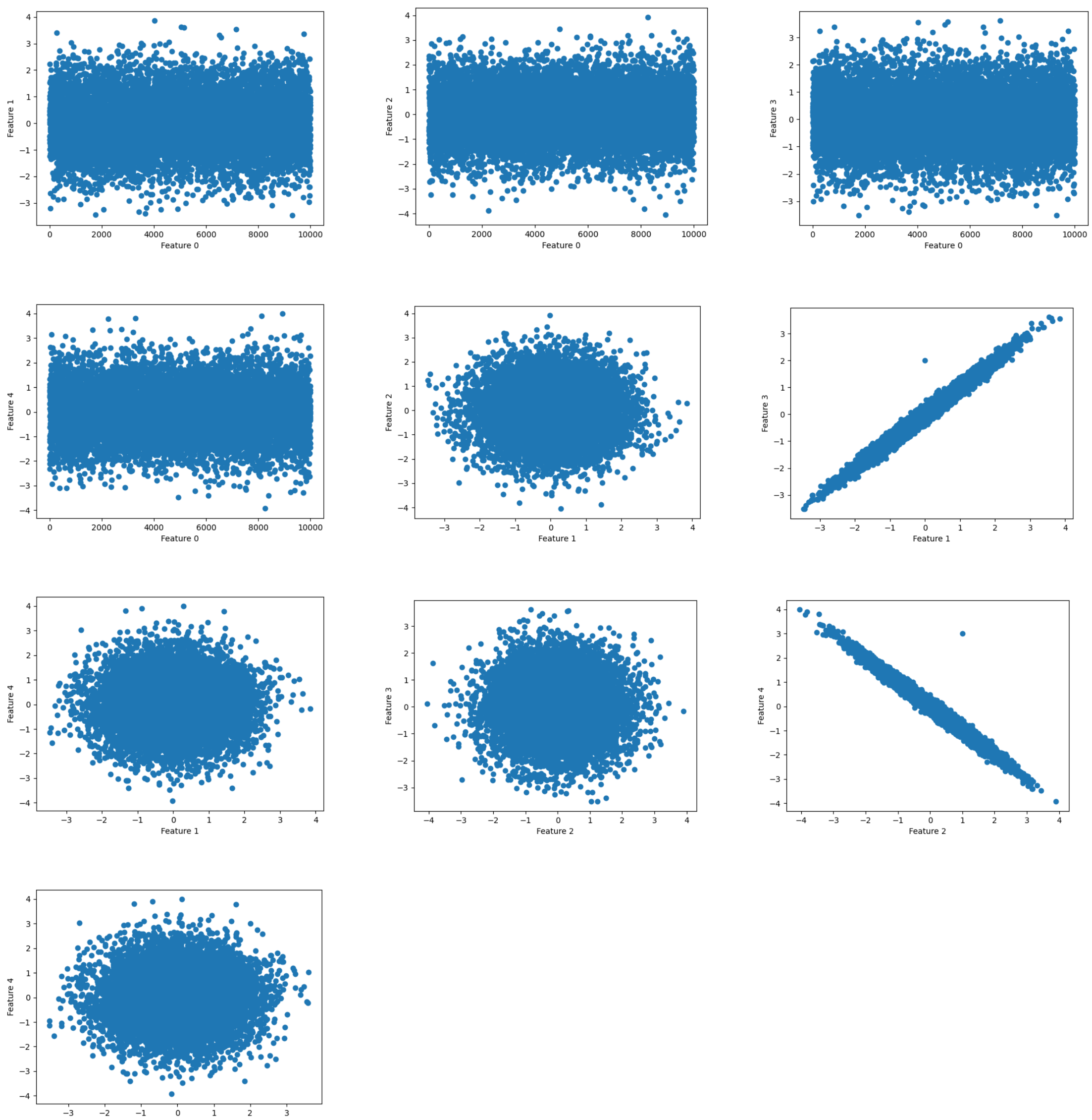
Lab 2

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)

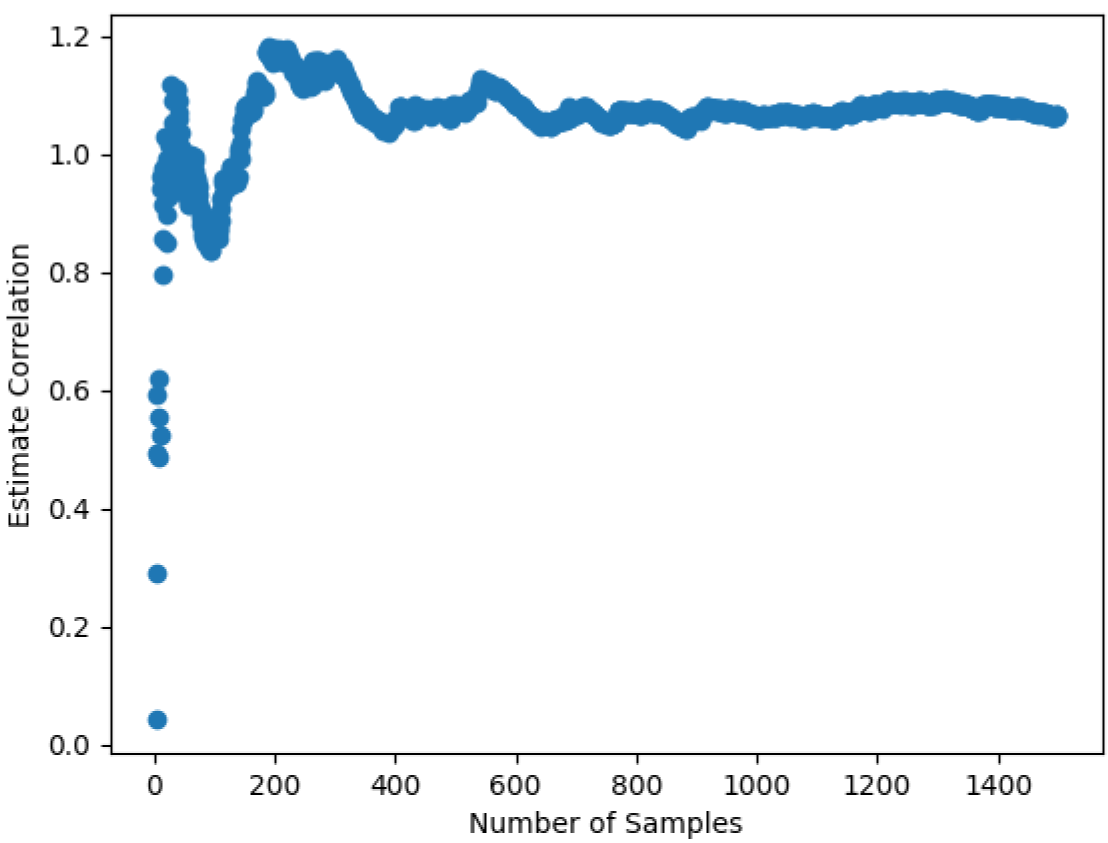
	0	1	2	3
0	var(0)	cov(0,1)	cov(0,2)	cov(0,3)
1	cov(1,0)	var(1)	cov(1,2)	cov(1,3)
2	cov(2,0)	cov(2,1)	var(2)	cov(2,3)

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

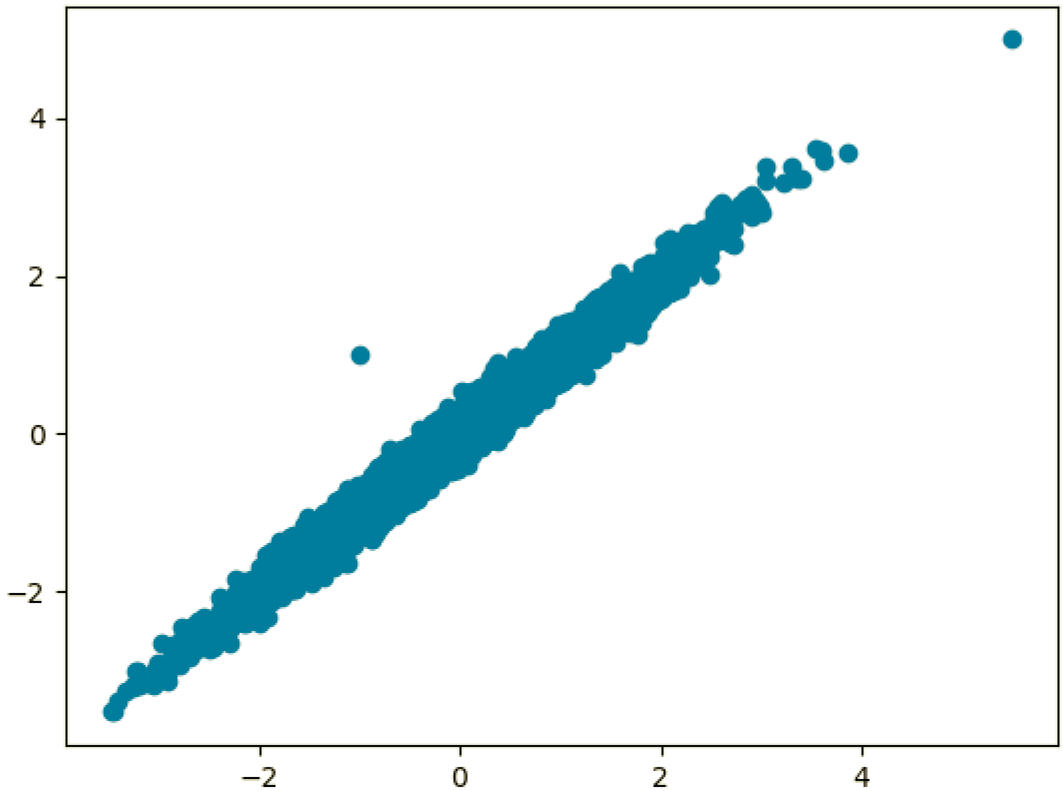
	0	1	2	3	4
0	8.334167e+06	-11.530835	25.440172	-11.682650	-20.510169
1	-1.153084e+01	1.001458	-0.004012	0.991523	0.004122
2	2.544017e+01	-0.004012	1.005376	-0.003901	-0.995059
3	-1.168265e+01	0.991523	-0.003901	1.001885	0.004680
4	-2.051017e+01	0.004122	-0.995059	0.004680	1.005973

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)



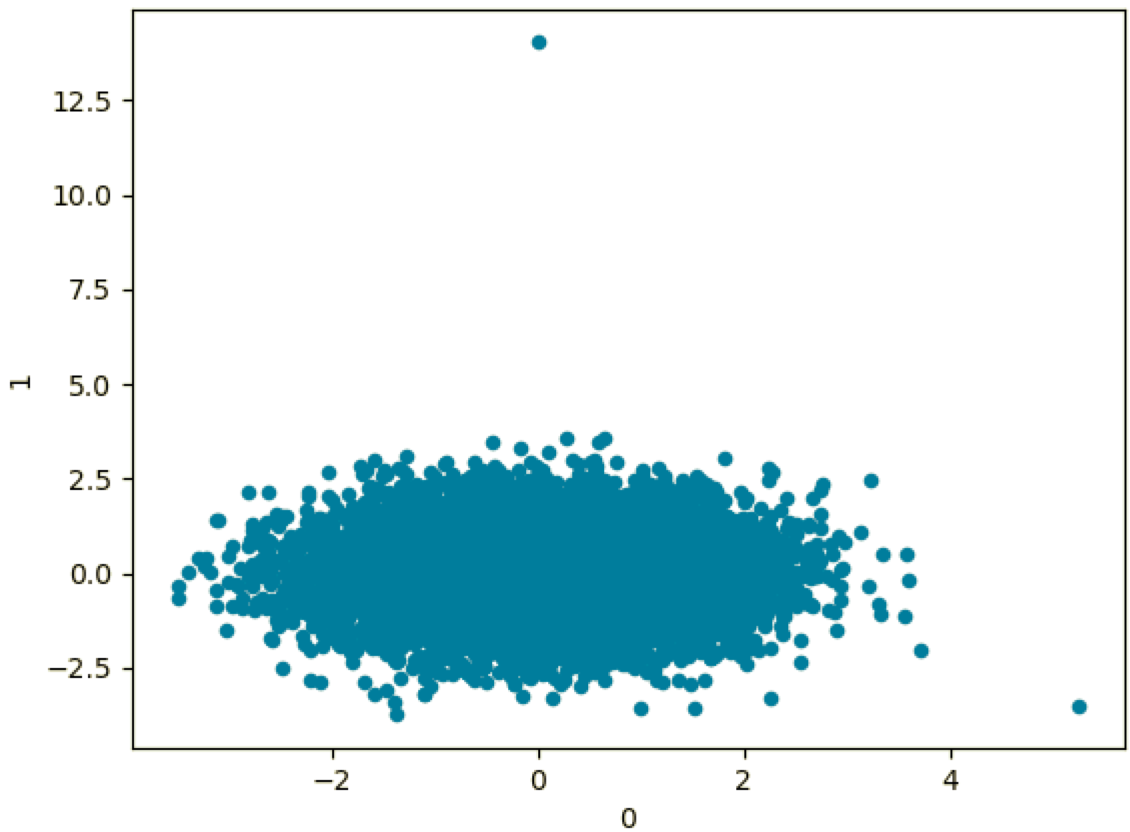
Problem 2



According to hint: The covariance matrix of z = (the covariance matrix of y)(Q)(Q^T) and since we know from the hint that the covariance matrix of y is a 2x2 identity matrix, the covariance matrix of z=(Q)(Q^T). Since if given z then y = $Q^{-1}z$. So we simply need to get Q^{-1} for ourselves using the eigenvalues and eigenvectors of z. Using eigendecomposition we can say that the covariance matrix of z = (the eigenvectors)(the eigenvalues)($the\ eigenvectors^T$) = vλv^T As v is a diagonal matrix and λ is orthogonal then $v^{-1} = v^T$ so since cov(z) = vλv^T = (v$\lambda^{1/2}$)($\lambda^{1/2}v^T$) then $(v\lambda^{1/2})^T = (v\lambda^{1/2})^{-1} = (\lambda^{-1/2}v^T)$ 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)])
```

```
Enter a year: 1892
Enter the number of names: 7
          0  1     2
27    Sarah  F  1799
58   Stella  F   972
70    Sadie  F   724
74    Susie  F   669
83   Sallie  F   582
122   Susan  F   366
147    Sara  F   276
```

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))
```

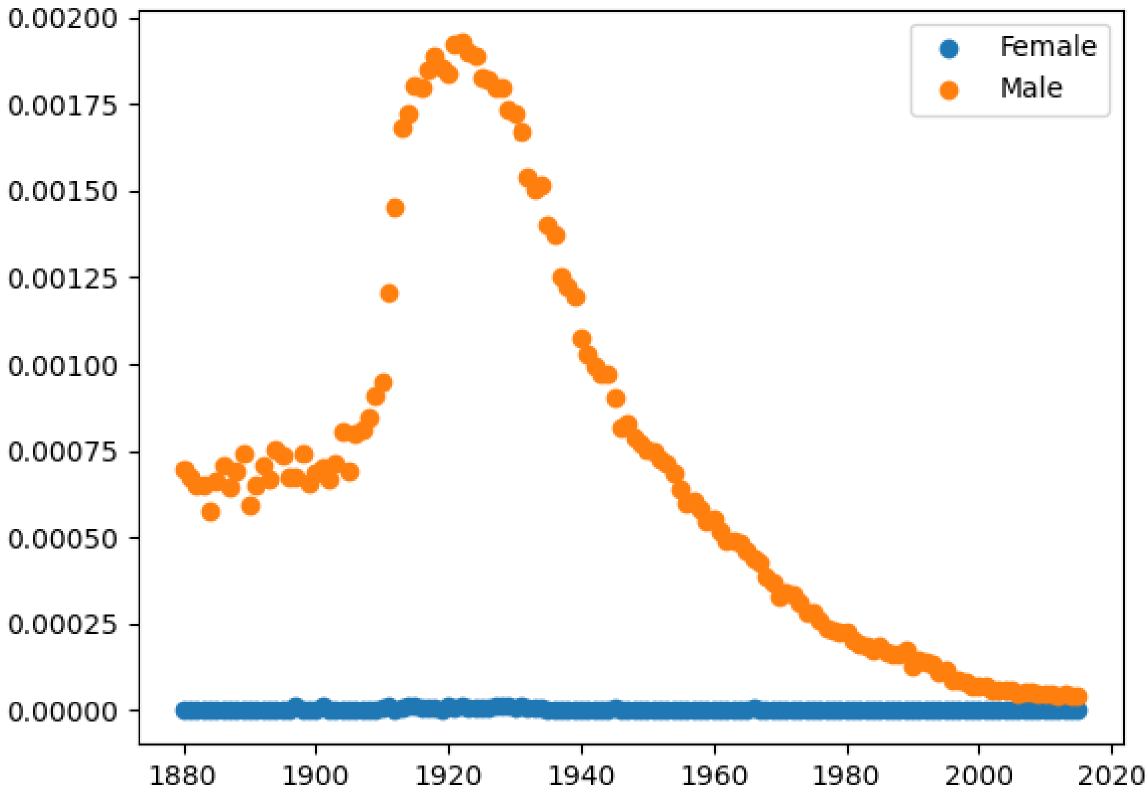
```
Enter a name: Sarah
/home/hprairie/460J/Labs/Lab2/Problem3.py:32: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeError in the future. Use int(ser.iloc[0]) instead
  print("The number of men with the name %s is: %i" % (name,df.iloc[df.index[(df[df.columns[0]] == name) & (df[df.columns[1]] == "M")]][2]))
The number of men with the name Sarah is: 3311
/home/hprairie/460J/Labs/Lab2/Problem3.py:33: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeError in the future. Use int(ser.iloc[0]) instead
  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]))
The number of men with the name Sarah is: 1065265
The most common first letter of all names is J with 43937781 names
```

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"])
```

```
(460J) hprairie@DESKTOP-CAT2SDB:~/460J/Labs/Lab2$ /home/hprairie/miniconda3/envs/460J/bin/python /home/hprairie/460J/Labs/Lab2/Problem3.py
Enter a year: 1881
/home/hprairie/460J/Labs/Lab2/Problem3.py:98: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeError in the future. Use int(ser.iloc[0]) instead
  previous = int(previous[2])
The name with the largest surge is Ethel.
(460J) hprairie@DESKTOP-CAT2SDB:~/460J/Labs/Lab2$ /home/hprairie/miniconda3/envs/460J/bin/python /home/hprairie/460J/Labs/Lab2/Problem3.py
Enter a year: 1921
/home/hprairie/460J/Labs/Lab2/Problem3.py:98: FutureWarning: Calling int on a single element Series is deprecated and will raise a TypeError in the future. Use int(ser.iloc[0]) instead
  previous = int(previous[2])
The name with the largest surge is Robert.
```

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[:,'MSSubClas
s':'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)
```

first_pred.csv **0.13564**

Complete · 12m ago · simple ridge pred

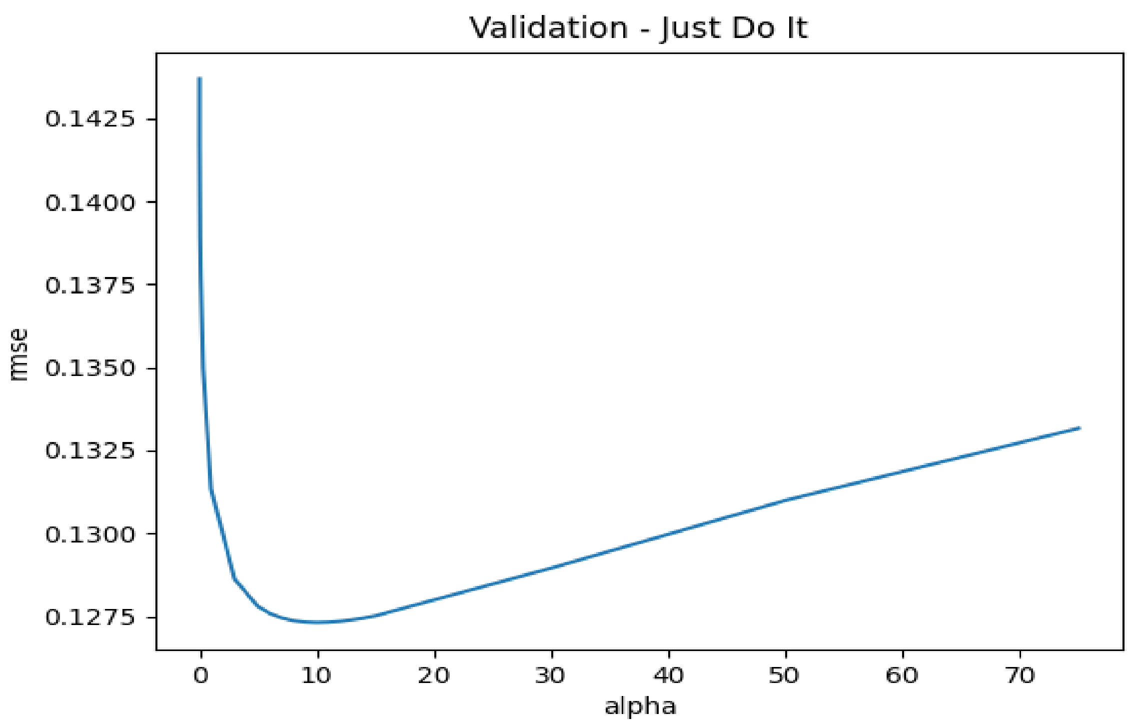
3)

```
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))
```

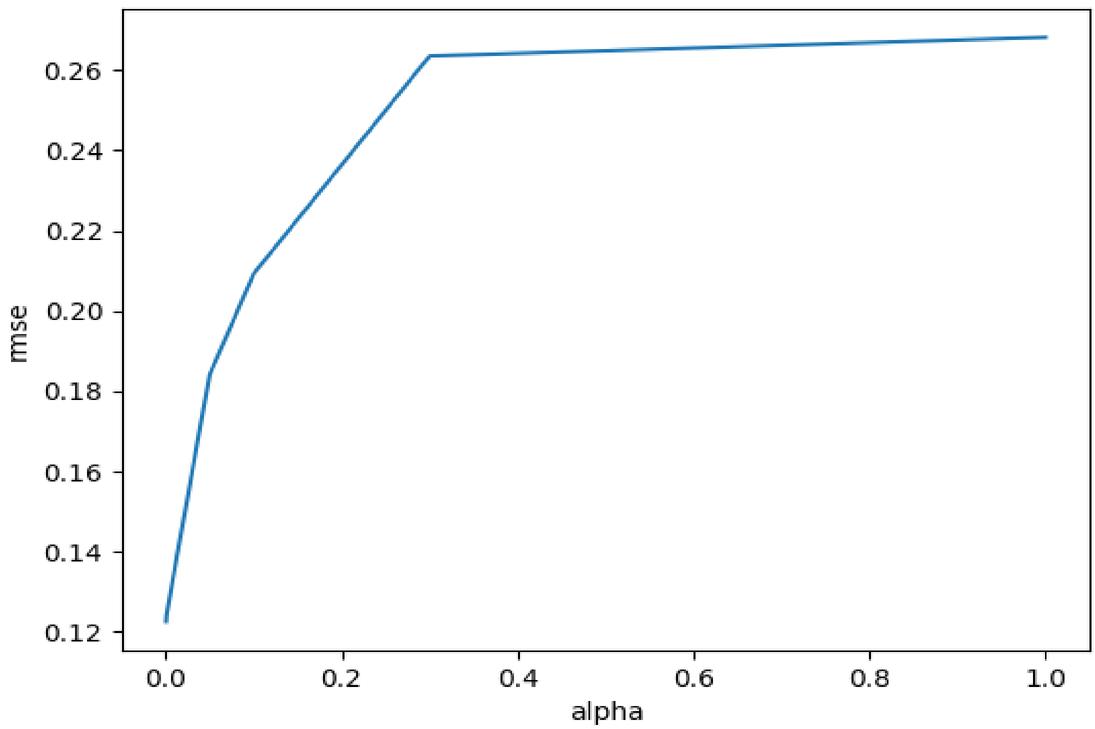
```
cv_ridge = pd.Series(cv_ridge, index = alphas)
cv_ridge.plot(title = "Validation - Just Do It")
plt.xlabel("alpha")
plt.ylabel("rmse")
plt.savefig("Ridge_MSE_Alpha.png")
plt.cla()

# 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)
```

```
(460J) hprairie@DESKTOP-CAT2SDB:~/460J/Labs/Lab2$ /home/hprairie/miniconda3/envs/460J/bin/python /home/hprairie/460J/Labs/Lab2/Problem4.py
  Best Ridge: 0.127312
  Best Lasso: 0.122567
```

Ridge Plot

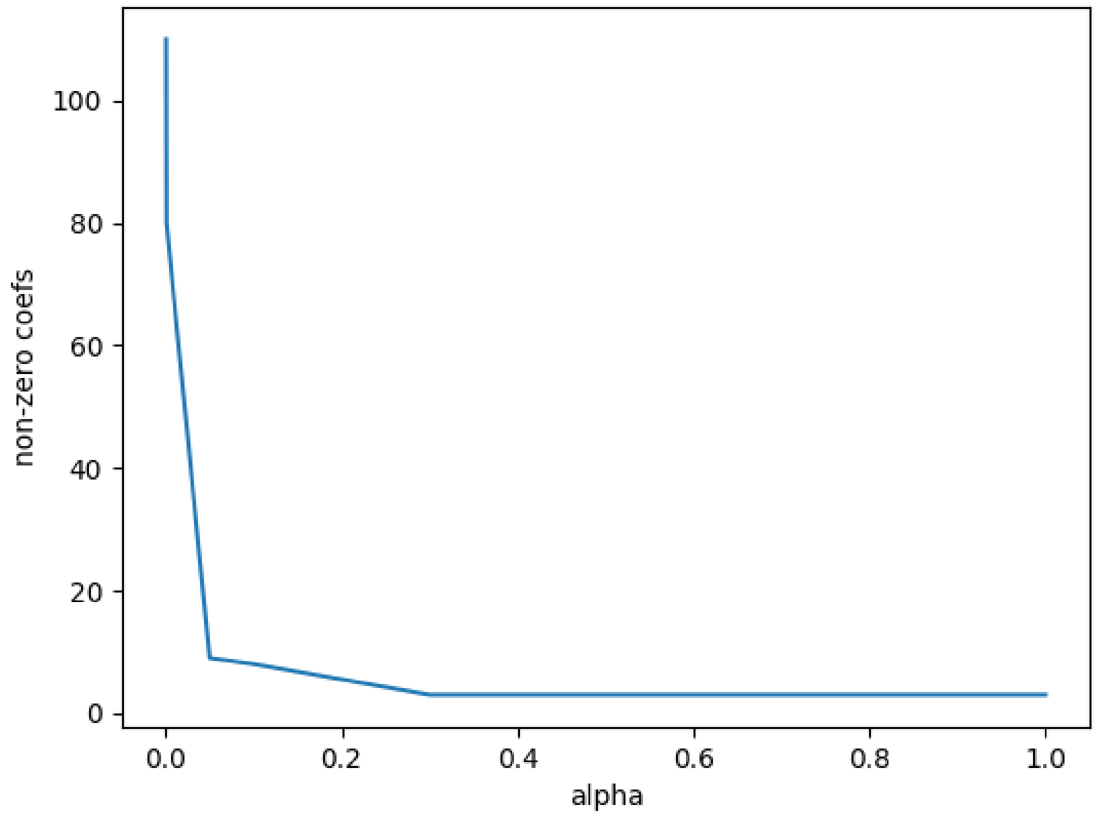


Lasso Plot



4)

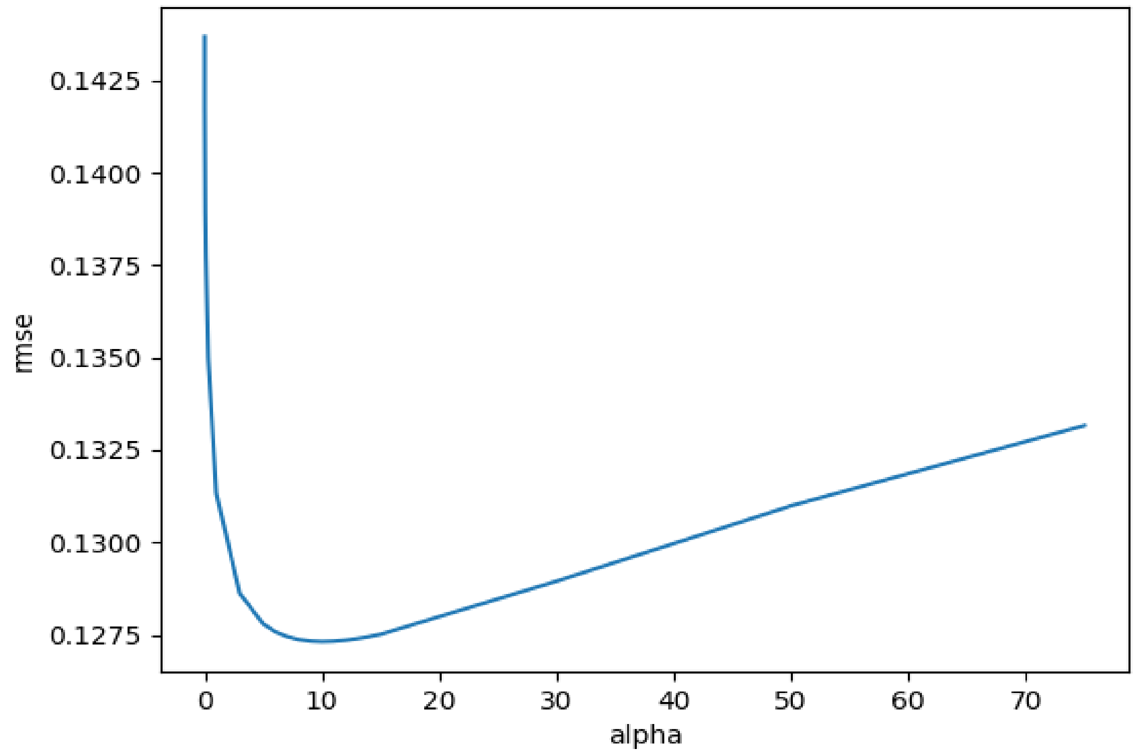
```
alphas = [0.0005, 0.001, 0.05, 0.1, 0.3, 1]

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))
```



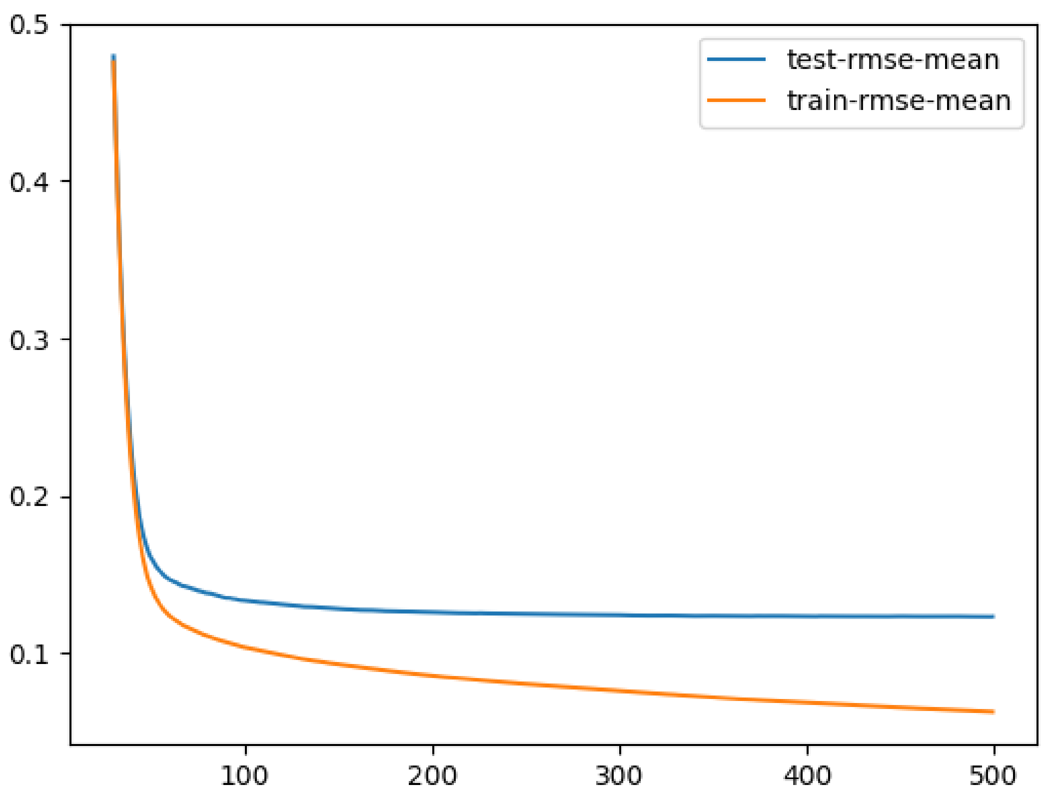

```
model_ridge = Ridge(alpha=9).fit(X_train_temp,y)

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)
```

ridge_stacking_sol.csv
Complete · now

0.12513

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)
```

```python
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)
```

xgb_sol.csv — Complete · now — **0.13286**

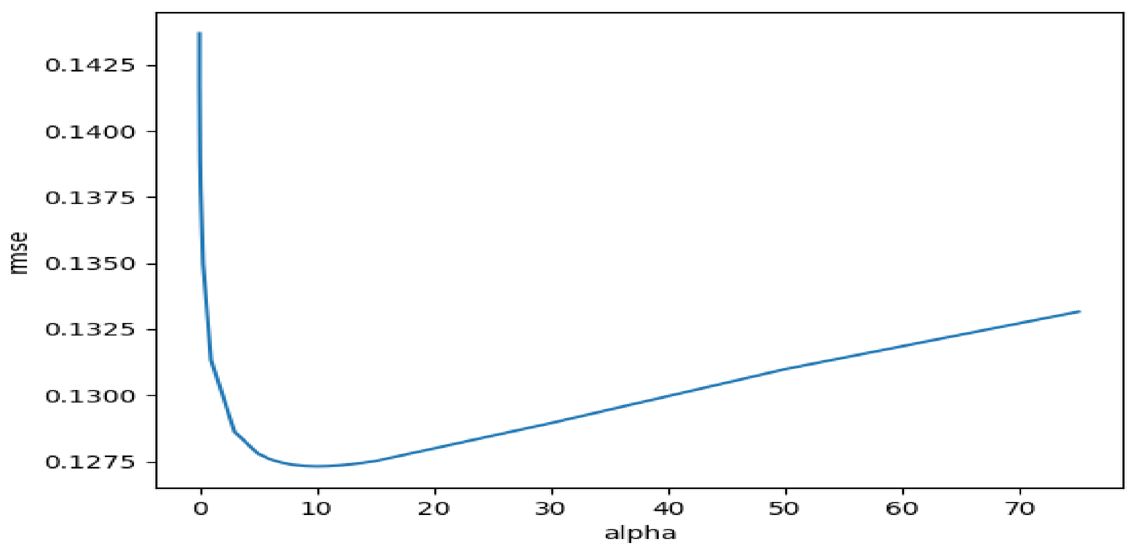
7)

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

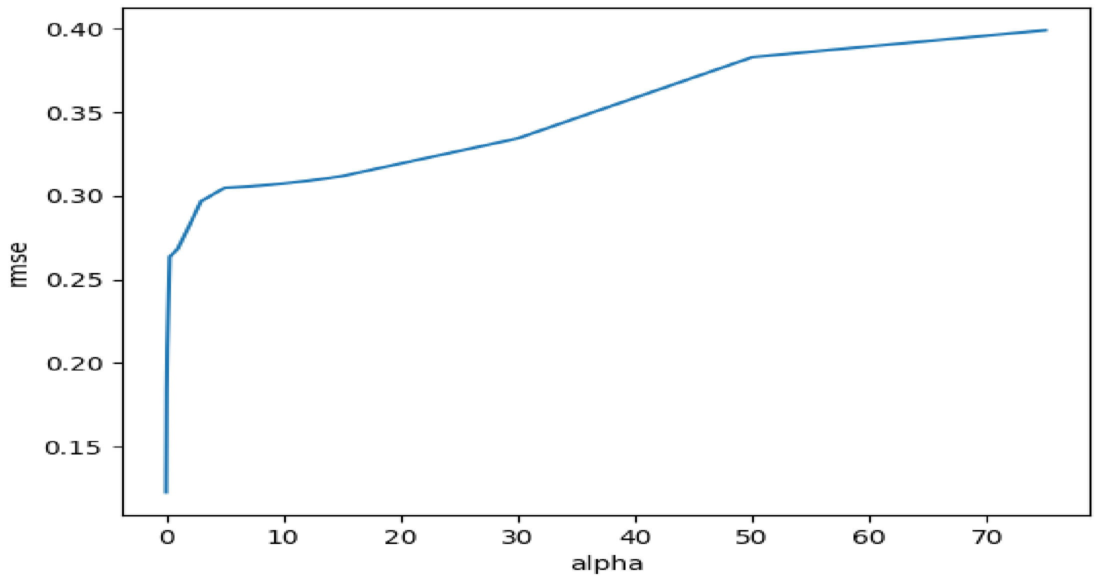
```python
model_lasso = Lasso(alpha=0.005).fit(X_train,y)
model_lasso_data = model_lasso.predict(X_train)
model_lasso_data_test = model_lasso.predict(X_test)
X_train_temp = np.column_stack((X_train,model_lasso_data))
X_test_temp = np.column_stack((X_test,model_lasso_data_test))

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.




```
Best Ridge: 0.127312
Best Ridge: 0.122567
```

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

problem7_attempt1_sol.csv 0.12691
Complete · now

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

problem7_attempt1_sol.csv 0.12863
Complete · now

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.

problem7_attempt2_sol.csv 0.12533
Complete · now

And with the best lasso, we get the following.

problem7_attempt2.2_sol.csv 0.12582
Complete · now

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))
```

problem7_attempt2.3_sol.csv
Complete · now
0.12552

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.

problem7_attempt2.4_sol.csv
Complete · now
0.12378

What if we weight it differently?

```
preds = (0.7)*(lasso_preds+ridge_preds)/2 + xgb_preds*(0.3)
```

problem7_attempt2.5_sol.csv
Complete · now
0.12387

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)
```

problem7_attempt2.6_sol.csv **0.12367**

Complete · now

With that we get the best prediction so far.