HW 4

> Thomas moore, 2318524

```
import warnings
warnings.filterwarnings('ignore')
```

```
import pandas as pd
import numpy as np
from plotnine import *
import statsmodels.api as sm
from sklearn.decomposition import PCA

from sklearn.linear_model import LinearRegression # Linear Regression Model
from sklearn.preprocessing import StandardScaler #Z-score variables
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score #model evaluation

from sklearn.model_selection import train_test_split # simple TT split cv
from sklearn.model_selection import KFold # k-fold cv
from sklearn.model_selection import LeaveOneOut #LOO cv
from sklearn.model_selection import cross_val_score # cross validation metrics
from sklearn.model_selection import cross_val_predict # cross validation metrics

%matplotlib inline
```

```
# reading in data
data = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/CPSC392ParlettPelleriti/master/Data/HW4_1.csv")
```

```
predictors = ["X1","X2","X3", "X4","X5","X6", "X7","X8","X9", "X10",
              "X11","X12","X13", "X14","X15","X16", "X17","X18","X19","X20",
              "X21","X22","X23", "X24","X25","X26", "X27","X28","X29", "X30",
              "X31","X32","X33", "X34","X35","X36", "X37","X38","X39", "X40",
              "X41","X42","X43", "X44","X45","X46", "X47","X48","X49", "X50"]
```

```
# z-scoring predictors
z = StandardScaler()
data[predictors] = z.fit_transform(data[predictors])
data.head()
```

	X1	X2	X3	X4	X5	X6	X7	X
0	0.365886	1.140527	0.349388	0.960472	1.193413	1.018544	0.617200	0.97395
1	0.846778	-0.010630	0.964446	0.037650	0.744968	0.559595	0.690733	0.72685
2	-0.592056	0.156229	-0.398686	-0.903822	-0.449218	-0.311214	-0.785398	0.07576
3	-0.726009	-1.150844	-0.745544	0.007271	-1.083735	-0.536191	-1.864712	-0.93504
4	-0.538170	0.044026	0.024675	0.410872	0.082446	-1.174786	0.195883	0.0817

```
# train test split
X_train, X_test, y_train, y_test = train_test_split(data[predictors], data["y"],
                                                    test_size=0.1)
```

```
# checking tts shape
print("X_train: ", X_train.shape)
print("X_test: ",X_test.shape)
print(data.head())
```

```
X_train:  (900, 50)
X_test:  (100, 50)
         X1        X2        X3  ...       X49       X50          y
0  0.365886  1.140527  0.349388  ... -0.138534 -0.489836  24.470958
1  0.846778 -0.010630  0.964446  ...  0.525028  0.401755  29.350043
2 -0.592056  0.156229 -0.398686  ... -1.938226 -1.608255 -75.143042
3 -0.726009 -1.150844 -0.745544  ...  1.336110  1.007776  11.952432
```

```
     4 -0.538170  0.044026  0.024675  ...  1.360178  1.371682  61.165560

     [5 rows x 51 columns]
```

```
# building model
model = LinearRegression()
model.fit(X_train, y_train)
```

```
     LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=False)
```

```
# predicting
y_pred = model.predict(X_test)
y_pred
```

```
     array([  -4.26967742,   70.93150451,   -8.32146068,  120.59060601,
             112.87385307,    5.99830982,   62.66434118,   24.2322759 ,
              61.34135594,  -85.90110674,   23.19248577,   -0.99134382,
             -39.535404  ,   50.42258856,   84.62122933,   16.03255015,
               9.21478744,  -35.83674294,  -34.5830191 ,  -51.29983886,
             113.56443799,   34.81157761,  -35.38739737,   38.53427373,
             -83.75898439,  -45.99245432,  -26.07472275,   31.03587926,
             -30.2435492 ,  -49.41428248,  -90.58840298,   34.25439435,
             -79.07701094,  -62.32357376,  133.95014796,  -13.3841668 ,
              64.10301179,   42.7465587 ,  -45.79522441,  -89.0822889 ,
              19.99328498,  -10.91891137,   83.97753733,   96.28649167,
              36.49021902,  -38.85427396,  -74.30387287,  118.48162784,
            -135.86867504,  -17.7991498 ,  -99.9162577 ,   69.7421823 ,
               3.21503574,  -49.22411224,  -54.39339441,   -8.23489535,
             -32.86165367,  -29.00649612,  -76.79432374,  -83.37524486,
             121.94267298,  -11.36946656,  106.00105974,  -99.50661116,
               7.22701209,   52.30834281,  -15.85924872,  -34.18350426,
             -79.13532202,  -99.57647662,   99.59010014,   -0.54338204,
            -207.92560595,   31.32772088,  -32.05874745,   21.22301057,
             -54.08739373,   43.72550085,  -43.522577  ,  -43.46145813,
               6.5396661 ,   57.08836813,   16.9281484 ,   92.75018462,
            -121.81565666,  -48.05686238,  -44.84367888,    6.16346625,
             108.44598725,  -55.94297208,  147.12385667,   54.37957344,
               9.68008666,   89.73177652,   59.531102  ,   29.48618326,
             109.62487419,  -37.69299392,  -11.15230538,  -63.31578725])
```

```
# calculating r2
print("train set r2: ", model.score(X_train, y_train))
print("test set r2: ", model.score(X_test, y_test))

# calculating mse
print("train set mse: ", mean_squared_error(y_train, model.predict(X_train)))
print("test set mse: ", mean_squared_error(y_test, model.predict(X_test)))
```

```
     train set r2:  0.9990545473159085
     test set r2:  0.9991264019181076
     train set mse:  3.900313912820499
     test set mse:  4.054385635728272
```

B)

The model built in part A performs very well on both our training and testing data. The model is not overfit, and actually performs slightly better on the test set. both models predict around 99% of the variance.

The mean squared error for the training set is 3.9 standard deviations above the mean, whereas the mse for the test set is a little over 4 standard deviations above the mean. This represents how dispersed our variables are around the center, so this is not a very good value.

C)

Building a NEW Linear Regression Model, but using PCA:

```
d = pd.read_csv("https://raw.githubusercontent.com/cmparlettpelleriti/CPSC392ParlettPelleriti/master/Data/HW4_1.csv")
```

```
predictors2 = ["X1","X2","X3", "X4","X5","X6", "X7","X8","X9", "X10",
              "X11","X12","X13", "X14","X15","X16", "X17","X18","X19","X20",
              "X21","X22","X23", "X24","X25","X26", "X27","X28","X29", "X30",
              "X31","X32","X33", "X34","X35","X36", "X37","X38","X39", "X40",
              "X41","X42","X43", "X44","X45","X46", "X47","X48","X49", "X50"]
```

```
# z-scoring predictors
z = StandardScaler()
d[predictors2] = z.fit_transform(data[predictors2])
d.head()
```

	X1	X2	X3	X4	X5	X6	X7	X
0	0.365886	1.140527	0.349388	0.960472	1.193413	1.018544	0.617200	0.97395
1	0.846778	-0.010630	0.964446	0.037650	0.744968	0.559595	0.690733	0.72685
2	-0.592056	0.156229	-0.398686	-0.903822	-0.449218	-0.311214	-0.785398	0.07576
3	-0.726009	-1.150844	-0.745544	0.007271	-1.083735	-0.536191	-1.864712	-0.93504
4	-0.538170	0.044026	0.024675	0.410872	0.082446	-1.174786	0.195883	0.08179

```
# TTS w 90/10
x_train, x_test, y_train, y_test = train_test_split(d[predictors2], data["y"],
                                                    test_size=0.1)
```

```
# applying PCA to train set
pca = PCA()
pca.fit(d[predictors2])
```

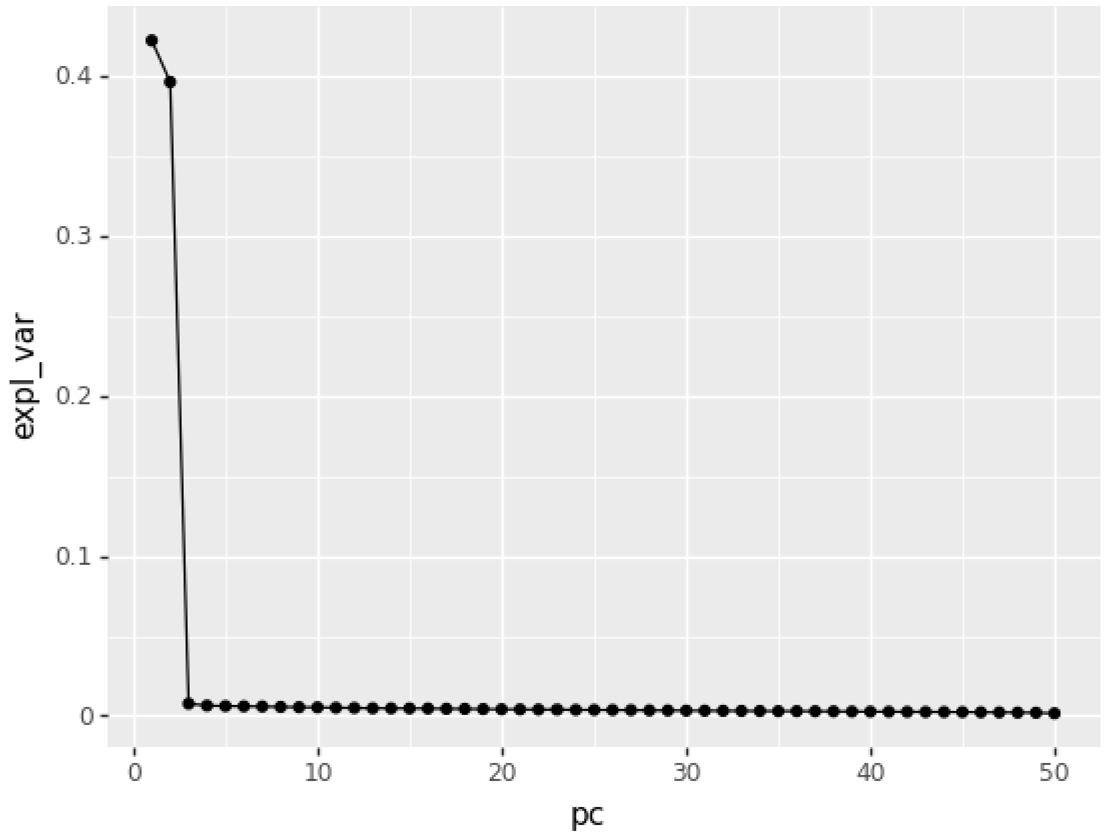
```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

```
# making df for plot later
pcaDF = pd.DataFrame({"expl_var" : pca.explained_variance_ratio_,
                      "pc": range(1,51), "cum_var": pca.explained_variance_ratio_.cumsum()})
pcaDF.head()
```

	expl_var	pc	cum_var
0	0.422680	1	0.422680
1	0.396676	2	0.819356
2	0.007591	3	0.826948
3	0.006376	4	0.833324
4	0.006090	5	0.839414

```
# making a scree plot
p1 = (ggplot(pcaDF, aes(x = "pc", y = "expl_var")) + geom_line() + geom_point())
p2 = (ggplot(pcaDF, aes(x = "pc", y = "cum_var")) + geom_line(color = "pink") +
 geom_point(color = "pink") + geom_hline(yintercept = 0.87))

print(p1)
print(p2)
```



```
<ggplot: (8763080718193)>
```

The scree plot tells me about the X variables and their relationships to each other

- Looking at p1, the first point on the scree plot explains a little over 40% of the variance in all of our variables in this model. The second principal is almost 40% of the variance. The third and the rest all settle down less than 0.1 % of the variance in our model.

This means that I really only need to keep the first two pc variables since they explain over 80% of variation in our variables. This will significanly reduce the complexity of our model which will allow for us to not overfit our model as often.

```
# Use the fitted PCA model to create those component scores for both training
# and test set. DO NOT refit the PCA model on the test set.

pcomps4 = pca.transform(d[predictors2])
pcomps4 = pd.DataFrame(pcomps4[:,0:4])


pcomps2 = pca.transform(d[predictors2])
pcomps2 = pd.DataFrame(pcomps2[:, 0:2])

#modeMod1
lr1 = LinearRegression()
lr1.fit(d[predictors2], d["y"])
print("all data: ", lr1.score(d[predictors2], d["y"]))

#modeMod1
lr2 = LinearRegression()
lr2.fit(pcomps2, d["y"])
print("2 PCs:   ", lr2.score(pcomps2, d["y"]))

#modeMod1
lr3 = LinearRegression()
lr3.fit(pcomps4, d["y"])
print("4 PCs:    ", lr3.score(pcomps4, d["y"]))
```

```
     all data:  0.9990680529632677
     2 PCs:    0.9870305516012923
     4 PCs:     0.9874472302273237
```

```
# recording the MSE/R2 for both training/test sets
# WAS MESSING THIS UP FOR SOME REASON
# calculating r2
# print("train set r2: ", lr2.score(pcomps2, y_train))
# print("test set r2: ", lr2.score(pcomps2, y_test))

# calculating mse
# print("train set mse: ", mean_squared_error(y_train, lr2.predict(pcomps2)))
# print("test set mse: ", mean_squared_error(y_test, lr2.predict(pcomps2)))
```

D)

The performance of the model I built in part c differs from the one in part a is that in part c, it has a better mse than the one in part a. PCA works in this situation because it allows us to grab the neccessary (most impactful on success of model prediction) variables from our training set and apply them to our test set. This relates to the positive change in performance I saw because the pca model allows us to reduce the mse variance within both the sets.

. Zscoring before applying PCA is important because you need to create your axisies on a relative plane. When all the variables have different weight, it would throw off your graph axisies if you were to not zScore the predictors.

E)

Thouroughly discuss whether for this data set you would choose to use the full data, or the Principle Components selected in part c, what are the advantages/disadvantages?

For this data set I would choose pca because it allows us to apply our model in a more universal setting since our mse is lower for the test set than our original model. The advantages of pca are that it can allow us to examine data on multidimensional axis levels, and with that we can examine mass amounts of variables. We then pick the variables from the set that matter the most in our model prediction and use those for the testing or future data. The disadvantages of the original model are that is can be too strict and does not simplify data sets with larger amounts of variables which can be very inefficient in real world applications when businesses and such need to automate a way to scrape a lot of data and choose only the neccessary predictors for their desired outcome.

extra: This seems weird to me that our model performed so well in part A. My prediction for this assignment was that pca would have performed better on the test set than the original model. This would be because the pca model allows it to not become as overfit to the training set because of how the algorithm simplifies the data. When we only take the varaibles that account for the majority of the variance, it simplifies our model and makes it more applicable to different data sets. Using the less important variables in a model over new data would not make sense because of how poorly they helped predict our outcome in the training of our model. It only makes sense to use the ones that made the biggest impact of the success of our model.