HW4_SP21

May 11, 2021

1 HW4

1.1 GENERAL INSTRUCTIONS:

- **CLEARLY** mark where you are answering each question (all written questions must be answered in Markdown cells, NOT as comments in code cells)
- Show all code necessary for the analysis, but remove superfluous code

---

Using the dataset HW4_1,

- **a) (15 points)** Build a Linear Regression Model to predict `y` from all the `X` variables (`X1` to `X50`).
  - Use TTS with a 90/10 split (since data is large)
  - z-score your predictors
  - record the MSE/R2 for both training/test sets
- **b) (10 points)** Thouroughly discuss the performance of the model built in part a. (*IN A MARKDOWN CELL*)
- **c) (35 points)** Build a NEW Linear Regression Model, but using PCA:
  - first, Use TTS with a 90/10 split (since data is large)
  - z-score your predictors
  - next, apply PCA to the *training set*.
  - make a scree plot
  - (15 of 35 points) **Thouroughly discuss** what the scree plot tells you about the `X` variables and their relationships to each other. (*IN A MARKDOWN CELL*)
  - Figure out how many PCs you need to keep to retain 90% of the original variance.
  - 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.
  - fit your model using these components and record the MSE/R2 for both training/test sets
- **d) (20 points)** Thouroughly discuss how the performance of the model built in part c differs from the model in part a. In your answer, discuss how PCA works, and how that may relate to the change in performance you observed. Also discuss why z-scoring before applying PCA is important.(*IN A MARKDOWN CELL*)
- **e) (20 points)** 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?

(*IN A MARKDOWN CELL*)

[1]:
```
# import necessary packages
import warnings
warnings.filterwarnings('ignore')

import pandas as pd
import numpy as np
from plotnine import *


from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split

from sklearn import metrics
from sklearn.preprocessing import StandardScaler #Z-score variables

from sklearn.model_selection import train_test_split # simple TT split cv
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.linear_model import LinearRegression
from sklearn.decomposition import PCA

from sklearn.model_selection import GridSearchCV
from sklearn import decomposition, datasets
from sklearn.pipeline import Pipeline

%precision %.7g
%matplotlib inline
```

[2]:
```
data = pd.read_csv('https://raw.githubusercontent.com/cmparlettpelleriti/
↪CPSC392ParlettPelleriti/master/Data/HW4_1.csv')
```

[3]:
```
data.head()
# Ensured that there are no missing elements from dataset
# data.isnull().sum(axis=0)
print("There are no missing values from the dataset")
```

```
There are no missing values from the dataset
```

[4]:
```
# data.info()
data.shape
```

[4]:
```
(1000, 51)
```

[5]:
```
# data[data.duplicated()].count()
print("There are no duplicate values from the dataset")
```

```
There are no duplicate values from the dataset
```

[6]:
```
features = []
# put all of the Xs into the features list
for feature in data.columns:
    if(str(feature) != 'y'):
        features.append(str(feature))
len(features)
```

[6]:
```
50
```

[7]:
```
LR_Model = LinearRegression() # init an empty Linear Regression model

# vars mse and r2 scores
LR_mse = 0
LR_r2 = 0

z = StandardScaler()
```

2 A)

2.1 Question: Build a Linear Regression Model to predict y from all the X variables

2.2 Answer:

[8]:
```
# a
X_train, X_test, y_train, y_test = train_test_split(data[features], data["y"],␣
 ↪test_size=0.1)

print("{0:0.2f}% data is in training set.".format((len(X_train)/len(data.
 ↪index))*100))
print("{0:0.2f}% data is in test set.".format((len(X_test)/len(data.
 ↪index))*100))
```

```
90.00% data is in training set.
10.00% data is in test set.
```

[9]:
```
X_train[features] = z.fit_transform(X_train[features]) # z-score and fit bc␣
 ↪model is trained with train data
X_test[features] = z.transform(X_test[features]) # z-score but do not fit bc do␣
 ↪not want to leak test data into model

print("z-scored X Train: ")
X_train.head()
```

```
z-scored X Train:
```

[9]:
```
           X1        X2        X3        X4        X5        X6        X7  \
989  1.008033  0.592632  0.818202  0.802991  1.456343  0.972175  1.538542
845  0.480787  0.717563  0.423217 -0.807531  0.865570  0.062041 -0.299143
217  0.436805  0.282049  0.066015  0.083087  0.118407  0.047720  0.192094
32   1.739771  1.342898  1.275219  1.366417  1.135389  1.035608  1.629280
37   0.594005 -0.018501  0.214652  0.664414 -0.022913 -0.177831  0.050234

           X8        X9       X10  …       X41       X42       X43  \
989  1.615854  1.926015  0.313414  …  0.828279  0.889747  1.983928
845  0.122555  0.809809  0.191880  …  0.293223 -0.227618  0.785833
217  0.402682  0.606112  0.989091  … -0.567141  0.330478  0.008890
32   0.998390  1.801668  1.480356  …  0.751025  0.317862  0.969496
37  -0.131150 -0.551854  0.485029  …  0.598010  0.971947  1.021218

          X44       X45       X46       X47       X48       X49       X50
989  2.067698  1.418610  2.184420  1.769049  1.620804  0.715645  1.483737
845 -0.319452 -0.089786 -1.107032  0.119718 -0.295180  0.712850 -0.099317
217 -0.150212 -0.849994  0.313117 -0.443129  0.077430 -0.407556 -0.567156
32  -0.163393  0.569255 -0.090553  0.055813  0.794964  0.750303  0.307946
37   0.888895  1.164739  0.575431  0.887725  0.637626  0.931058  1.027171

[5 rows x 50 columns]
```

[10]:
```
LR_Model.fit(X_train, y_train) # fit the X and y training data to the LR model

y_pred = LR_Model.predict(X_test)

LR_mse = mean_squared_error(y_test, y_pred)
LR_r2 = r2_score(y_test, y_pred)

print("Linear Regression Model ~ Mean Squared Error:\n" + str(LR_mse) + "\n")
print("Linear Regression Model ~ r2 score:\n" + str(LR_r2))
```

```
Linear Regression Model ~ Mean Squared Error:
4.553352194581286

Linear Regression Model ~ r2 score:
0.999023297033239
```

3 B)

3.1 Question: Thouroughly discuss the performance of the model built in part a.

3.2 Answer:

3.2.1 What Mean Squared Error (mse) is and why it is important and used in this context:

- I am using mse as a metric to measure my model's performance because it is a good metric to use to check how close the model's forecasts are to actual results. The mean squared error is sum of squared errors divided by the number of data points and is considered a loss function because it is a measure of well a model is doing. The mean squared error value tells us approximately what error value we can expect to get from any data point on the Linear Regression (LR) model. Like the sum of squared errors, the lower the mean squared error is (relative to the outcome units squared), the better the LR model is at predicting the outcome variable (y).

3.2.2 Interpretation of mse from the Linear Regression Model:

- The mean squared error for the linear regression model is about 3.702 as shown above. As discussed before, the mse is in terms of the outcome units squared. In this LR model, the y-value does not have units and so the error is simply y-units squared. Given the ambiguity of the data and nature of mse, it is difficult to make any conclusions from the mse. The mse will be more helpful later on when we compare the PCA model's mse value to this one because it will provide us insight on how much the error changed from using less components. To help get a better idea of how well our LR model is doing without comparing it to another model (PCA), I calculate r2 next. r2 is generally more insightful since it gives a standardized score (between 0 and 1).

3.2.3 What r2 is and why it is important and used in this context:

- I am using r2 as a metric to measure my model's performance because it helps me understand the strength of the relationship between the predictor variables (Xs) and the outcome (y) in a standard scale (0 - 1). r2 represents the percentage of variance that is explained by the model. The closer the percentage or decimal value of r2 is to 1.0, the more the variation is explained by the model (as opposed to external factors/noises). In constrast, an r2 of 0 or close to 0 is an indicator that the model does a poor job of predicting the outcome because the variance is not explained by the model.

3.2.4 Interpretation of r2 from the Linear Regression Model:

- The r2 value is very high, 0.999, as shown above. This high r2 value indicates the model is performing very well at predicting the outcome variable (y) because the variation in our model's results are being explained from the model itself. We want the variation of a model to be explained by our predictors/features because that implies that the features are great choices for predicting the outcome variable of interest.

4 C)

4.1 Question: Build a NEW Linear Regression Model, but using PCA

4.2 Answer:

[11]:
```
PCA_LR_Model = LinearRegression() # init an empty Linear Regression model

# Use TTS with a 90/10 split (since data is large)
PCA_LR_X_train, PCA_LR_X_test, PCA_LR_y_train, PCA_LR_y_test =␣
↪train_test_split(data[features], data["y"], test_size=0.1)

# z-score predictors
PCA_LR_X_train[features] = z.fit_transform(PCA_LR_X_train[features]) # z-score␣
↪and fit bc model is trained with train data
PCA_LR_X_test[features] = z.transform(PCA_LR_X_test[features]) # z-score but do␣
↪not fit bc do not want to leak test data into model

PCA_Model = PCA()
PCA_Model.fit(PCA_LR_X_train)
```

[11]:
```
PCA()
```

[12]:
```
# mapping of both training and testing set to the PCA Model
PCA_LR_X_train = PCA_Model.transform(PCA_LR_X_train)
PCA_LR_X_test = PCA_Model.transform(PCA_LR_X_test)

# apply PCA to the training set
PCA_LR_Model.fit(PCA_LR_X_train, PCA_LR_y_train) # fit the X and y training␣
↪data to the LR model

PCA_LR_y_pred = PCA_LR_Model.predict(PCA_LR_X_test)

PCA_LR_mse = mean_squared_error(PCA_LR_y_test, PCA_LR_y_pred)
PCA_LR_r2 = r2_score(PCA_LR_y_test, PCA_LR_y_pred)

print("PCA Linear Regression Model ~ Mean Squared Error:\n" + str(PCA_LR_mse) +␣
↪"\n")
print("PCA Linear Regression Model ~ r2 score:\n" + str(PCA_LR_r2))
```

```
PCA Linear Regression Model ~ Mean Squared Error:
4.556310491959998

PCA Linear Regression Model ~ r2 score:
0.9989272147587734
```

[13]:
```
PCA_DF = pd.DataFrame({
    "Explained_Variance": PCA_Model.explained_variance_ratio_,
```

```
    "Principle_Components": range(1, 51),
    "Cumulative_Variance": PCA_Model.explained_variance_ratio_.cumsum()
})

PCA_DF.head()
```

[13]:

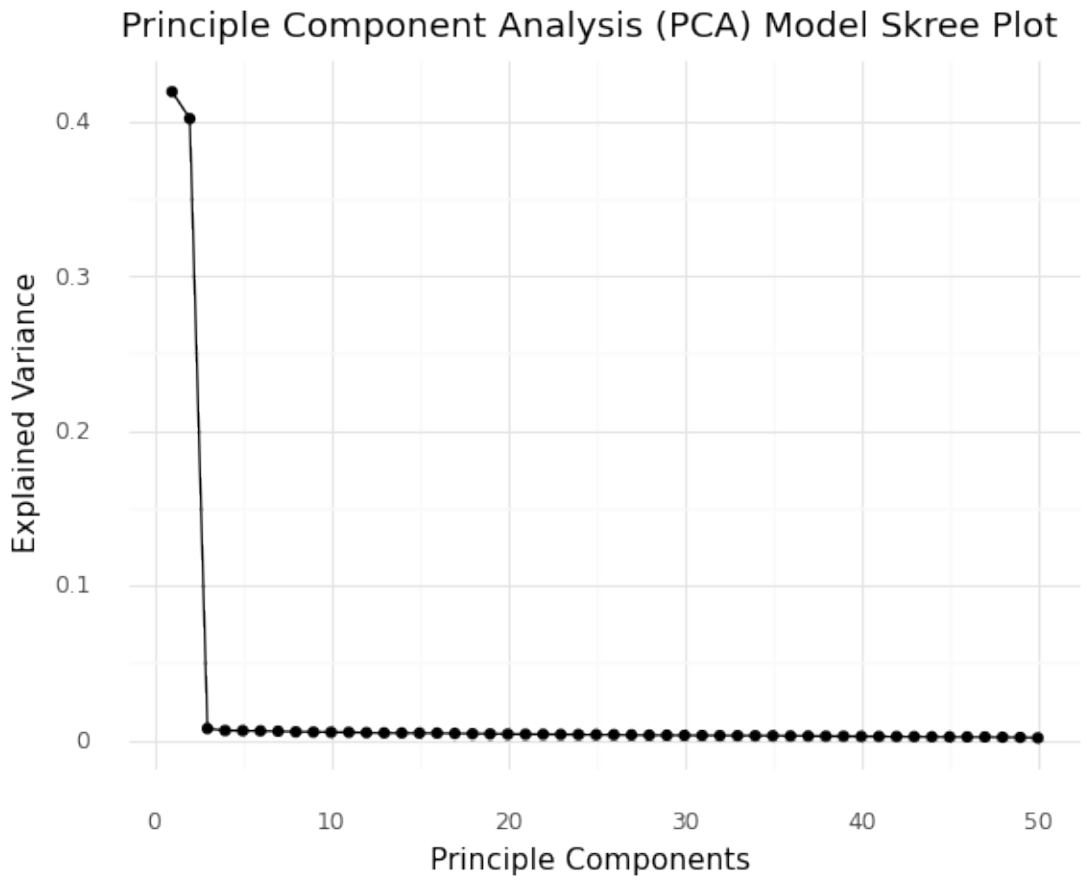
	Explained_Variance	Principle_Components	Cumulative_Variance
0	0.419153	1	0.419153
1	0.401839	2	0.820991
2	0.007579	3	0.828570
3	0.006301	4	0.834871
4	0.006217	5	0.841088

[14]:
```
PCA_DF.tail()
```

[14]:

	Explained_Variance	Principle_Components	Cumulative_Variance
45	0.002120	46	0.992536
46	0.002091	47	0.994627
47	0.001962	48	0.996589
48	0.001881	49	0.998470
49	0.001530	50	1.000000

[15]:
```
# pca a scree plot
(ggplot(PCA_DF, aes(x = "Principle_Components", y = "Explained_Variance")) +␣
 ↪geom_point() + geom_line() + theme_minimal() + ggtitle("Principle Component␣
 ↪Analysis (PCA) Model Skree Plot") + labs(x = "Principle Components", y =␣
 ↪"Explained Variance"))
```



[15]: <ggplot: (314241290)>

5 C cont)

5.1 Question: Thouroughly discuss what the scree plot tells you about the X variables and their relationships to each other.

5.2 Answer:

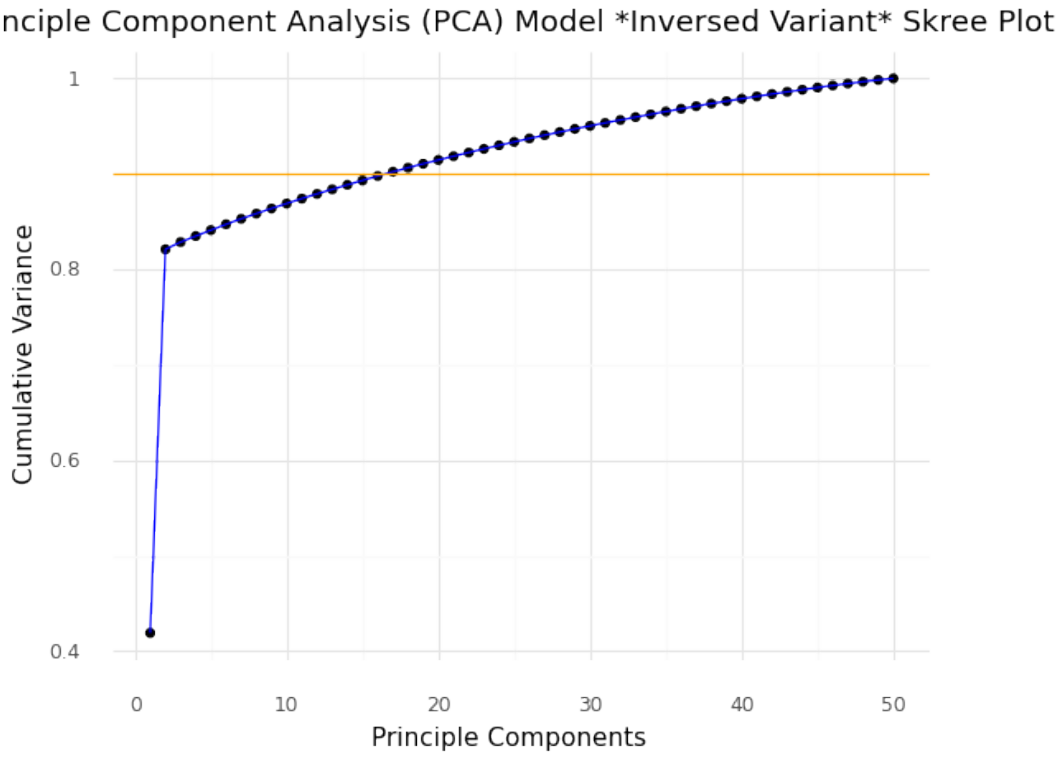
5.3 Explanation principle components and explained variance:

- The priniciple components are the features, in this case it is the Xs (X1 - X50). The explained variance is the percentage of variance that is being explained by the model. It is desirable for the explained variance to be as close to 100% as possible. This is because we want variation from the model to be explained from the predictors (AKA principle components) as opposed to outside noise.

5.4 Explanation and interpretation of Skree Plot:

- The skree plot is a scatter plot that visually represents how much much variation is being explained from the addition of a principle component. For example looking at the first principle component, we can observe that about 42% of the variance is being explained by just the first principle component analysis. In other words if we were to create the PCA model only with that first principle component, the results of the PCA model's predictions would be about 42% explained from that one predictor. The second principle component has an explained variance of about 39%. This indicates that the second component can explain about 40% of the model variance. If we combine the first 2 principle components' explained variances, we get a cumulative variance of about 82%. This technique of principle component analysis is very powerful because it allows us to minimize the number of principle components we use in a model to achieve a desired expected variation percentage. It is important to minimize these principle components because the more principle components that are used in a model, the more computationally expensive it is to get calculations and predictions from a model.

[16]:
```
# C cont)
# Figure out how many PCs you need to keep to retain 90% of the original␣
↪variance.
(ggplot(PCA_DF, aes(x = "Principle_Components", y = "Cumulative_Variance")) +␣
↪geom_point() + geom_line(color = "blue") + geom_hline(yintercept = 0.90,␣
↪color = "orange") + theme_minimal() + ggtitle("Principle Component Analysis␣
↪(PCA) Model *Inversed Variant* Skree Plot") + labs(x = "Principle␣
↪Components", y = "Cumulative Variance"))
```



[16]:
```
<ggplot: (314343138)>
```

[33]:
```
# method used to calculate the min number of principle components to achieve␣
↪the threshold cumulative accuracy
def calc_min_pc(data_frame, col_name, threshold):
    pc_index = 0
    for pc in data_frame[col_name]:
        pc_index += 1
        if pc >= threshold:
            return pc_index
```

[34]:
```
# figuring out how many PCs need to keep to retain 90% of the original variance
min_pc = calc_min_pc(PCA_DF, 'Cumulative_Variance', 0.90)
min_pc
```

[34]:
```
17
```

5.4.1 Showing that the 17th component is the first number of principle components that retains at least 90% of data variability:

[35]:
```
print("BEFORE Minimum PCA component: \n" +␣
↪str(PCA_DF['Cumulative_Variance'][min_pc - 2]) + "\n")
print("Minimum PCA component: \n" + str(PCA_DF['Cumulative_Variance'][min_pc -␣
↪1]) + "\n")
print("AFTER Minimum PCA component: \n" +␣
↪str(PCA_DF['Cumulative_Variance'][min_pc]))
```

```
BEFORE Minimum PCA component:
0.8977788530693844

Minimum PCA component:
0.9021979978561554

AFTER Minimum PCA component:
0.9064729820720643
```

6 Part C cont)

6.1 Question: 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. Fit your model using these components and record the MSE/R2 for both training/test sets

6.2 Answer:

[23]:
```
mod_PCA_Model = PCA(n_components = min_pc)
mod_PCA_Model.fit(PCA_LR_X_train)
```

[23]:
```
PCA(n_components=17)
```

[24]:
```
mod_train_y_pred = PCA_LR_Model.predict(PCA_LR_X_train)

train_mod_mse = mean_squared_error(PCA_LR_y_train, mod_train_y_pred)
test_mod_mse = mean_squared_error(PCA_LR_y_test, PCA_LR_y_pred)

train_mod_r2 = r2_score(PCA_LR_y_train, mod_train_y_pred)
test_mod_r2 = r2_score(PCA_LR_y_test, PCA_LR_y_pred)
```

[25]:
```
print("PCA Model MSE (Train): " + str(train_mod_mse))
print("PCA Model MSE (Test): " + str(test_mod_mse) + "\n")

print("PCA Model r2 (Train): " + str(train_mod_r2))
print("PCA Model r2 (Test): " + str(test_mod_r2))
```

```
PCA Model MSE (Train): 3.850324009717197
PCA Model MSE (Test): 4.556310491959998

PCA Model r2 (Train): 0.9990763521921641
PCA Model r2 (Test): 0.9989272147587734
```

7 Part D)

7.1 Question: Thouroughly discuss how the performance of the model built in part c differs from the model in part a. In your answer, discuss how PCA works, and how that may relate to the change in performance you observed.

7.2 Answer:

- The models of Part A and Part C are very similar except the Part C (PCA LR Model) only uses 17 principle components to create a Linear Regression Model and Part A (non-PCA LR Model) uses all 50 principle components. In order to make the PCA LR Model, I used the LinearRegression() constructor from sklearn to create the model. I supplied the parameter 'n_components' to be equal to 17 which tells the sklearn LR class to create the Linear Regression Model but only with 17 principle components. I chose 17 components because that is the minimum number of principle components to achieve a 90%+ cumulative

accuracy score. I was able to determine that 17 principle components are needed by leveraging a Principle Component Analysis (PCA) Model.

- Principle Component Analysis is a method of dimensionality reduction that processes data that is in the form of features/predictors into principle components. The principle components are the features that are determined in the order of the most variability of data. And so the first principle component determined by the PCA Model is the component that can explain the most variability of data in comparison to the other components. The second principle component has the second most variability of data, the third has the third most variability of data, and so on. Behind the scenes, PCA Models are determining the principle components by using the mathematical technique of eigendecomposition. The result of using PCA Models is that we can gain insight in which principle components are most important and then we can selectively decide which and how many principle components we want to use in a model.
- For the question above, we wanted to create a Linear Regression model with the provided dataset that has at least a 90% cumulative explained variance. I determined that 17 principle components are required to achieve that 90% requirement and so I am able to create a modified version of my Linear Regression Model that only utilizes those 17 principle components. There are huge performance benefits to minimizing the number of principle components we use, especially when working with large/dense datasets. We minimize the computational expenses computers have when we create or simulate models with their minimum number of principle components. This is because the computer has a lot less calculations and computations to perform when it only has to account for a given number of principle components as opposed to all of the principle components.

7.3 Question: Also discuss why z-scoring before applying PCA is important.

7.4 Answer:

- It is important to z-score data before applying PCA because we want to ensure that the different features/predictors are standardized. Being standardized means that the different features are transformed into a form in which all of the features can be compared to one another. This is especially important when dealing with features of different unit types. For example, if one feature is time in seconds and another feature is distance in miles, we want to z-score those features so that we can appropriately compare the two features of different units. After z-scoring, we can create a PCA Model because the model then has features that can be appropriately compared to one another. We want to ensure that these different features can be fairly compared when the PCA model is performing eigendecomposition to determine the principle components.

8 Part E)

8.1 Question: 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?

8.2 Answer:

- I would choose to use the principle components selected in Part C because those pinciple components explain 90% of the variance of data while having a much lower computational cost. The computationally cost is much lower because the Linear Regression Model only has

to process 17 principle components when it is being simulated or used. The computer would have to perform far more calculations and computations to process 50 principle components compared to 17 principle components. Thus, the 17 principle components model in Part C is far more efficient with a small loss of explained variance.

- Advantages of using the Part C Model:
  - A lot less computationally expensive for the computer to process/use the model
  - Retains 90% of the explained variability from the dataset
  - Reduces overfitting because overfitting tends to happen when there are too many different variables in a dataset
  - Removes repetitive features that may be providing the same information to the model (would cause overfitting if not removed)
- Disadvantages of using the Part C Model:
  - Must be careful with what number of principle components we use, otherwise will lose valuable information in the model
  - Principle components are often considered not as easily interpretable as the original features because they are linear combinations of the original features.
  - Have to remember to standardize the data before applying data to PCA