Emma Mayes (eemayes2)

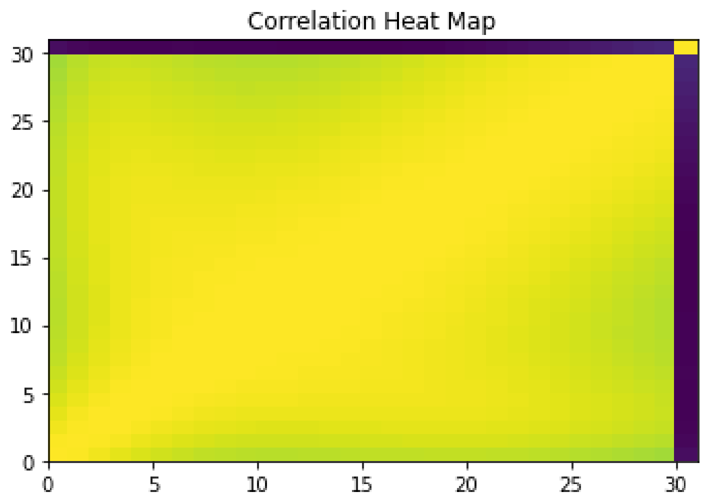
IE598 MLF F18

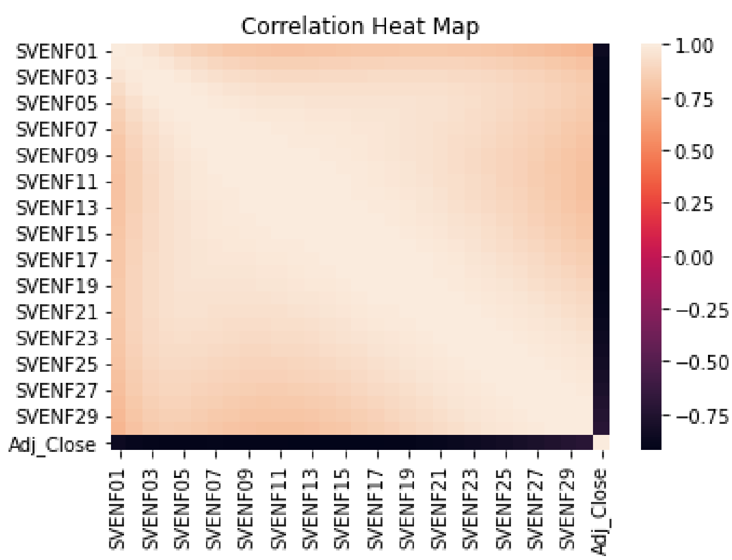
Module 5 Homework (Dimensionality Reduction)

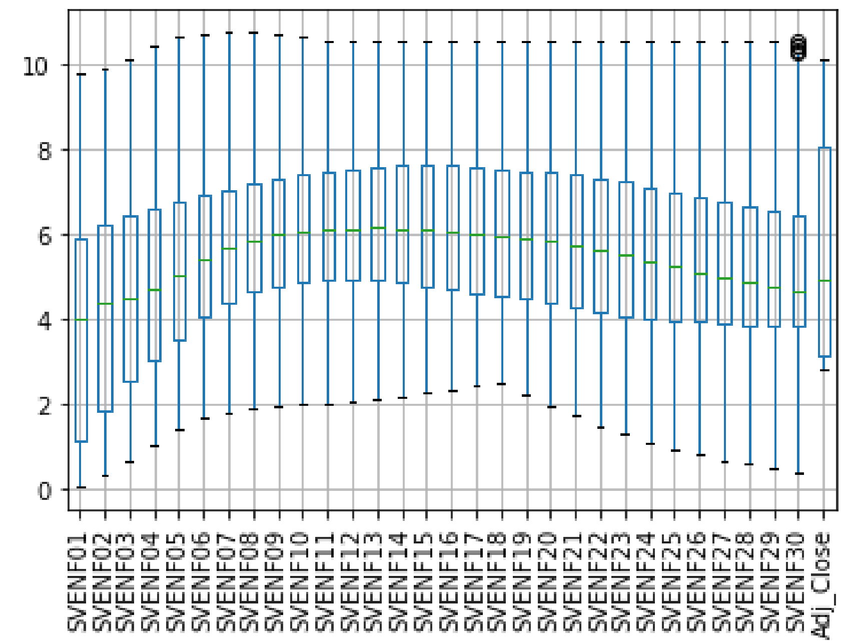
Use the Treasury Yield Curve dataset

Part 1: Exploratory Data Analysis

Describe the data set sufficiently using the methods and visualizations that we used previously. Include any output, graphs, tables, that you think is necessary to represent the data. Label your figures and axes. DO NOT INCLUDE CODE, only output figures!







Statistical descriptions of all features is too large of a table to place here, so it can be found in my printed PDF of my code attached at the end of this document

Split data into training and test sets. Use random_state = 42. Use 85% of the data for the training set. Use the same split for all experiments.

Part 2: Perform a PCA on the Treasury Yield dataset

Compute and display the explained variance ratio for all components, then recalculate and display on n_components=3.

What is the cumulative explained variance of the 3 component version.

`Cumulative explained variance for 3 components: 0.9944059173093O3`

Part 3: Linear regression v. SVM regressor - baseline

Fit a linear regression model to both datasets (the original dataset with 30 attributes and the PCA transformed dataset with 3 PCs.) using SKlearn. Calculate its accuracy R2 score and RMSE for both in sample and out of sample (train and test sets). (You may use CV accuracy score if you wish).

Fit a SVM regressor model to both datasets using SKlearn. Calculate its accuracy R2 score and RMSE for both in sample and out of sample (train and test sets). (You may use CV accuracy score if you wish).

Part 4: Conclusions

Write a short paragraph summarizing your findings. Which model performs best on the untransformed data? Which transformation leads to the best performance increases? How does training time change for the two models. Report your results using the Results worksheet format. Embed the completed table in your report.

Overall, the linear regression model did the best in accuracy on the untransformed data. However, most likely, this model, along with the SVR model for untransformed data, are most likely not generalizable since it is dependent on 30 features. When the data was transformed using PCA, the accuracy was very comparable to what was shown for the full 30 feature model, but the model will most likely be more generalizable as a result of reducing the data to 3 principal components.

	Experiment 1 (Treasury Yields)			
	Linear		SVR	
Baseline (all attributes)	Train Acc:	0.902273	Train Acc:	0.89202
	Test Acc:	0.90413	Test Acc:	0.89246
PCA transform (3 PCs)	Train Acc:	0.86722	Train Acc:	0.86248
	Test Acc:	0.86624	Test Acc:	0.86117

Part 5: Appendix

Link to github repo: https://github.com/eemayes2/IE517_F21_HW5

IE517_HWK5

September 24, 2021

[1]:
```
#Import libraries needed
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')

#Check if any null values we need to change
def num_missing(x):
    return sum(x.isnull())
```

[2]:
```
#Read in Data
df = pd.read_csv('hw5_treasury yield curve data.csv', header=0)
#Reminder: ATT1-13 is noise, MEDV is target variable
df.head()
```

[2]:
```
        Date  SVENF01  SVENF02  SVENF03  ...  SVENF28  SVENF29  SVENF30
Adj_Close
0  5/17/2019   2.1224   2.0266   2.1023  ...   3.6471   3.6970   3.7458
10.130177
1  5/16/2019   2.1239   2.0317   2.1096  ...   3.6660   3.7153   3.7636
10.130177
2  5/15/2019   2.0874   1.9956   2.0844  ...   3.6421   3.6847   3.7257
10.150118
3  5/14/2019   2.1319   2.0559   2.1451  ...   3.7132   3.7630   3.8113
10.130177
4  5/13/2019   2.1051   2.0234   2.1180  ...   3.6655   3.7098   3.7525
10.130177

[5 rows x 32 columns]
```

[3]:
```
df = df.drop(columns = ['Date'])
print(df.apply(num_missing, axis = 0))
```

```
SVENF01     0
SVENF02     0
SVENF03     0
```

```
SVENF04      0
SVENF05      0
SVENF06      0
SVENF07      0
SVENF08      0
SVENF09      0
SVENF10      0
SVENF11      0
SVENF12      0
SVENF13      0
SVENF14      0
SVENF15      0
SVENF16      0
SVENF17      0
SVENF18      0
SVENF19      0
SVENF20      0
SVENF21      0
SVENF22      0
SVENF23      0
SVENF24      0
SVENF25      0
SVENF26      0
SVENF27      0
SVENF28      0
SVENF29      0
SVENF30      0
Adj_Close    0
dtype: int64
```

0.1 Part 1: EDA

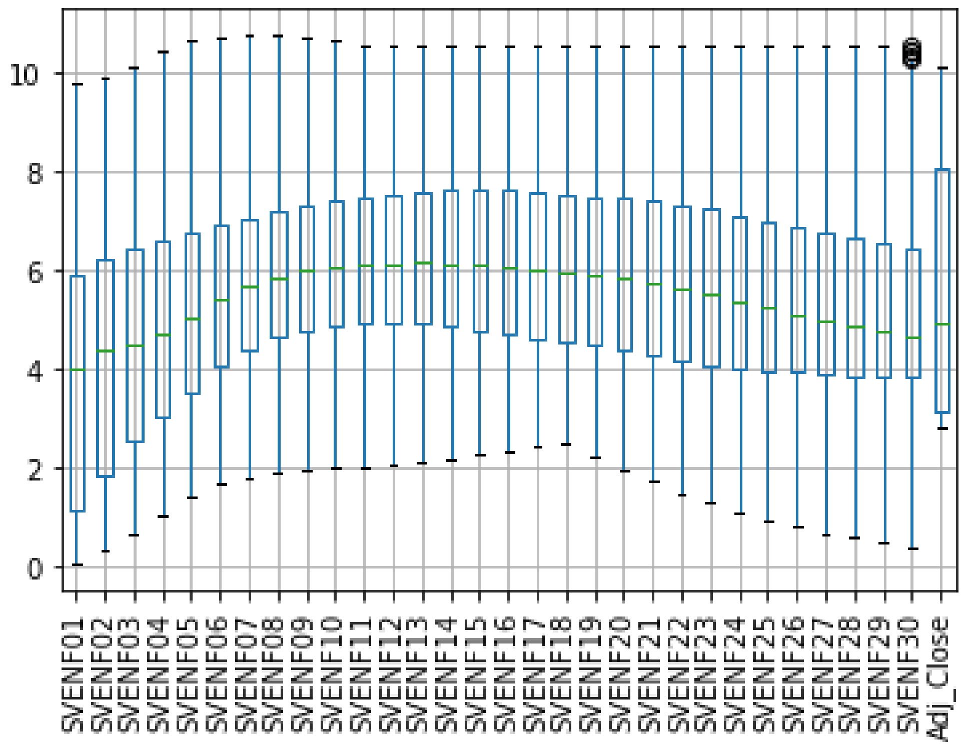
Before Standardizing

[4]:
```
df.describe()
```

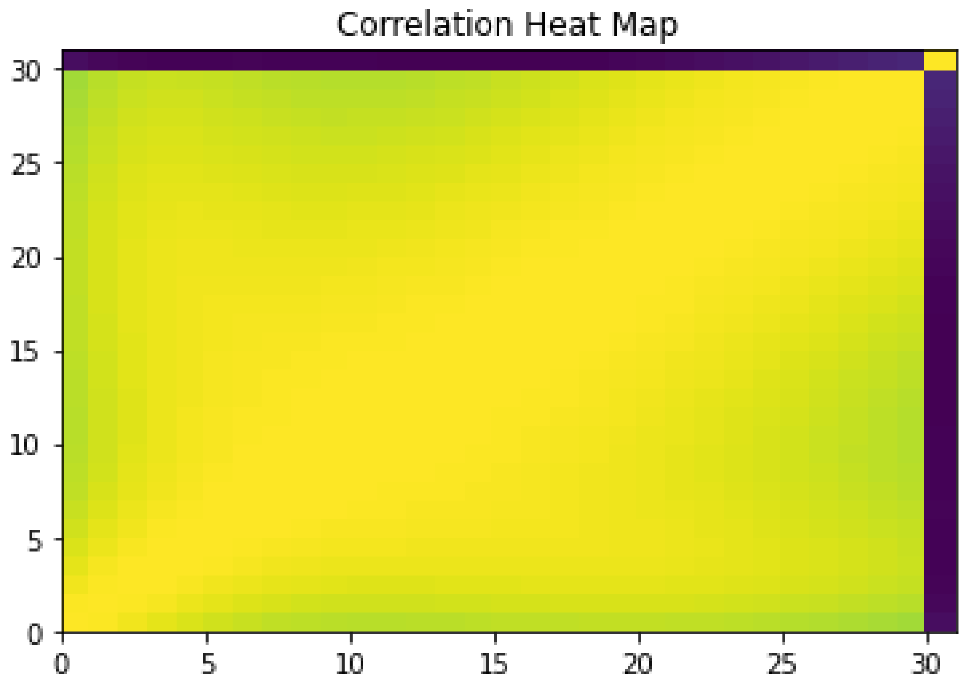
[4]:
```
           SVENF01      SVENF02  ...      SVENF30    Adj_Close
count  8071.000000  8071.000000  ...  8071.000000  8071.000000
mean      3.785311     4.258972  ...     5.167371     5.509793
std       2.648060     2.498137  ...     1.847834     2.491110
min       0.072700     0.327300  ...     0.411100     2.801050
25%       1.144050     1.865600  ...     3.831350     3.130587
50%       3.986500     4.393300  ...     4.669000     4.956219
75%       5.901500     6.221250  ...     6.421850     8.051437
max       9.813800     9.887800  ...    10.535100    10.150118

[8 rows x 31 columns]
```

[27]:
```
#Boxplots
df.boxplot()
plt.xticks(rotation = 90)
```

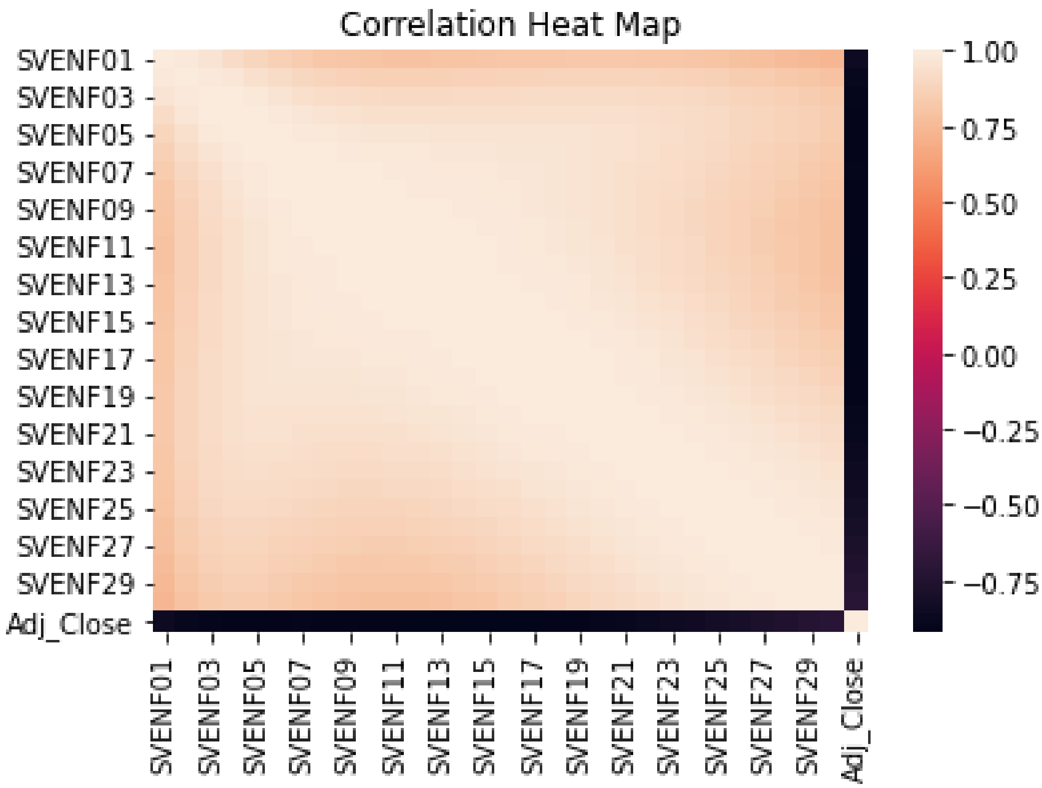
[27]:
```
(array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
        18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31]),
 <a list of 31 Text major ticklabel objects>)
```



[6]:
```
corMat = pd.DataFrame(df.corr())

plt.pcolor(corMat)
plt.title("Correlation Heat Map")
plt.show()
```



[7]:
```
correlation_mat = df.corr()

sns.heatmap(correlation_mat, annot = False)
plt.title("Correlation Heat Map")
plt.show()

#heavy correlation for most features
```



[8]:
```
#Split into training-test sets
from sklearn.model_selection import train_test_split

x = df.drop(columns = ['Adj_Close'])
x.head()

X_train, X_test, y_train, y_test = train_test_split(x, df['Adj_Close'], test_size=0.15, random_state=42)
```

[9]:
```
#Standardize with standard scaler
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

0.2 PCA on dataset

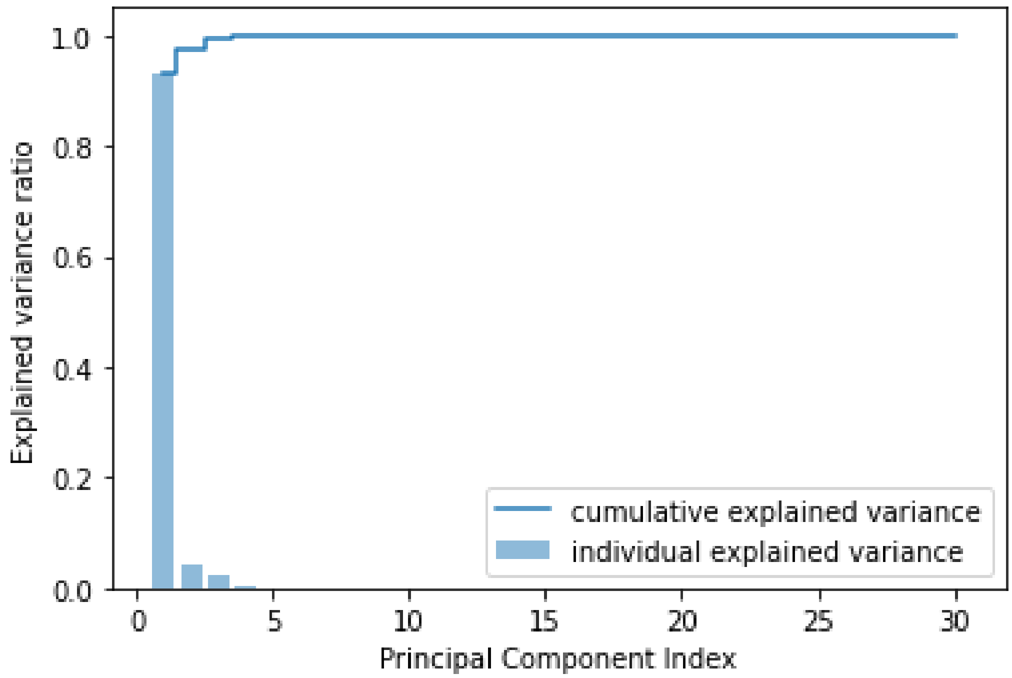
[10]:
```
#Cumulative explained variance
cov_mat = np.cov(X_train_std.T)
eigenvals, eigenvecs = np.linalg.eig(cov_mat)
print('\nEigenvalues \n%s' % eigenvals)
```

```
Eigenvalues
[2.79579848e+01 1.22313057e+00 6.55411544e-01 1.45561136e-01
 1.99031388e-02 2.06243818e-03 2.84492074e-04 3.29679042e-05
 2.55310505e-06 2.02358034e-07 1.41807223e-08 1.14065184e-09
 1.87834406e-10 2.79223564e-10 1.97434888e-10 2.08170843e-10
 2.16456135e-10 2.61581427e-10 2.21833140e-10 2.26975790e-10
 2.24866287e-10 2.31123423e-10 2.57846729e-10 2.56093138e-10
 2.35379808e-10 2.52582442e-10 2.49758675e-10 2.38344430e-10
 2.43769541e-10 2.44978760e-10]
```

[11]:
```python
tot = sum(eigenvals)
var_exp = [(i/tot) for i in sorted(eigenvals, reverse = True)]
cum_var_exp = np.cumsum(var_exp)

print(var_exp)
print(cum_var_exp)

print(len(var_exp))
```

```
[0.9317969749380347, 0.04076507559642957, 0.021843866774838572,
0.004851330579495056, 0.000663341249742568, 6.87379176250734e-05,
9.481686747978828e-06, 1.0987699468249641e-06, 8.509109602390575e-08,
6.744284524055292e-09, 4.726218380051956e-10, 3.8016185315353525e-11,
9.306095353676375e-12, 8.71810986919668e-12, 8.593638070386526e-12,
8.535193547628366e-12, 8.418187396993204e-12, 8.324075570815556e-12,
8.164768289399366e-12, 8.12446687552559e-12, 7.943656188689898e-12,
7.84484986872113e-12, 7.702991064291871e-12, 7.564756762006768e-12,
7.494450263299693e-12, 7.393360094241526e-12, 7.2141527135822344e-12,
6.9380165690743895e-12, 6.580203582839132e-12, 6.260234154352169e-12]
[0.93179697 0.97256205 0.99440592 0.99925725 0.99992059 0.99998933
 0.99999881 0.99999991 0.99999999 1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.
 1.         1.         1.         1.         1.         1.        ]
30
```

[12]:
```python
plt.bar(range(1,31), var_exp, alpha = 0.5, align = 'center', label =␣
 ↪'individual explained variance')
plt.step(range(1,31), cum_var_exp, where = 'mid', label = 'cumulative explained␣
 ↪variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal Component Index')
plt.legend(loc = 'best')
plt.show()
```



[13]:
```
print("Cumulative explained variance for 3 components: " + str(cum_var_exp[2]))
```

```
Cumulative explained variance for 3 components: 0.994405917309303
```

[14]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components = 3)
X_train_pca = pca.fit_transform(X_train_std)
X_test_pca = pca.transform(X_test_std)
```

0.3 Lin Reg v. SVM Reg: Baseline

Linear Regression: Full set

[15]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

lr = LinearRegression()
lreg = lr.fit(X_train_std, y_train)
y_pred = lreg.predict(X_test_std)
y_train_pred = lreg.predict(X_train_std)
```

[16]:
```
print("Train Score: " + str(lreg.score(X_train_std, y_train)))
print("Test Score: " + str(lreg.score(X_test_std, y_test)))
print("Mean Squared Error (MSE): " + str(mean_squared_error(y_test, y_pred)))
print("RMSE: " + str(np.sqrt(mean_squared_error(y_test, y_pred))))
```

```python
print("Train R^2: " + str(r2_score(y_train, y_train_pred)))
print("Test R^2: " + str(r2_score(y_test, y_pred)))
```

```
Train Score: 0.9022730353400435
Test Score: 0.9041309535337262
Mean Squared Error (MSE): 0.6121021683244493
RMSE: 0.7823695855057565
Train R^2: 0.9022730353400437
Test R^2: 0.9041309535337262
```

Linear Regression: PCA

[17]:
```python
lr_pca = LinearRegression()
lregpca = lr_pca.fit(X_train_pca, y_train)
y_pred = lregpca.predict(X_test_pca)
y_train_pred = lregpca.predict(X_train_pca)
```

[18]:
```python
print("Train Score: " + str(lregpca.score(X_train_pca, y_train)))
print("Test Score: " + str(lregpca.score(X_test_pca, y_test)))
print("Mean Squared Error (MSE): " + str(mean_squared_error(y_test, y_pred)))
print("RMSE: " + str(np.sqrt(mean_squared_error(y_test, y_pred))))

print("Train R^2: " + str(r2_score(y_train, y_train_pred)))
print("Test R^2: " + str(r2_score(y_test, y_pred)))
```

```
Train Score: 0.8672181160186359
Test Score: 0.8662415053375473
Mean Squared Error (MSE): 0.8540177213873134
RMSE: 0.924130792359671
Train R^2: 0.8672181160186357
Test R^2: 0.8662415053375473
```

SVM Reg: Baseline

[19]:
```python
from sklearn import svm
clf_svr = svm.SVR(kernel = 'linear')
clf_svr.fit(X_train_std, y_train)
y_pred = clf_svr.predict(X_test_std)
y_train_pred = clf_svr.predict(X_train_std)
```

[20]:
```python
print("Train Score: " + str(clf_svr.score(X_train_std, y_train)))
print("Test Score: " + str(clf_svr.score(X_test_std, y_test)))
print("Mean Squared Error (MSE): " + str(mean_squared_error(y_test, y_pred)))
print("RMSE: " + str(np.sqrt(mean_squared_error(y_test, y_pred))))

print("Train R^2: " + str(r2_score(y_train, y_train_pred)))
print("Test R^2: " + str(r2_score(y_test, y_pred)))
```

```
Train Score: 0.8920208361922309
Test Score: 0.8924613825895129
```

```
Mean Squared Error (MSE): 0.686609738198755
RMSE: 0.8286191756161301
Train R^2: 0.8920208361922309
Test R^2: 0.8924613825895129
```

SVM Reg: PCA

[21]:
```
clf_svr = svm.SVR(kernel = 'linear')
clf_svr.fit(X_train_pca, y_train)
y_pred = clf_svr.predict(X_test_pca)
y_train_pred = clf_svr.predict(X_train_pca)
```

[22]:
```
print("Train Score: " + str(clf_svr.score(X_train_pca, y_train)))
print("Test Score: " + str(clf_svr.score(X_test_pca, y_test)))
print("Mean Squared Error (MSE): " + str(mean_squared_error(y_test, y_pred)))
print("RMSE: " + str(np.sqrt(mean_squared_error(y_test, y_pred))))

print("Train R^2: " + str(r2_score(y_train, y_train_pred)))
print("Test R^2: " + str(r2_score(y_test, y_pred)))
```

```
Train Score: 0.8624827979809777
Test Score: 0.8611702699819538
Mean Squared Error (MSE): 0.8863964116075648
RMSE: 0.9414862779709351
Train R^2: 0.8624827979809778
Test R^2: 0.8611702699819538
```

0.4 Statements & Print to PDF

[23]:
```
print("My name is Emma Mayes")
print("My NetID is: eemayes2")
print("I hereby certify that I have read the University policy on Academic␣
 ↪Integrity and that I am not in violation.")
```

```
My name is Emma Mayes
My NetID is: eemayes2
I hereby certify that I have read the University policy on Academic Integrity
and that I am not in violation.
```

[ ]:
```
!wget -nc https://raw.githubusercontent.com/brpy/colab-pdf/master/colab_pdf.py
from colab_pdf import colab_pdf
colab_pdf('IE517_HWK5.ipynb')
```

```
File colab_pdf.py already there; not retrieving.
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
```

```
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.

Extracting templates from packages: 100%
```