**MTH 4330 HW 3 - Ely Bardavit**

(1) Upload and plot the data.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

trainingData = pd.read_csv('train4.csv')
plt.figure(figsize=(15,5))
plt.scatter(trainingData.Time, trainingData.Price)
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Plot of Training Data')
```


*The figure above shows the output of the code which plots the training data.

(2) Use the sklearn LinearRegression function to fit a linear model y = β0 + β1x to the data contained intrain.txt .

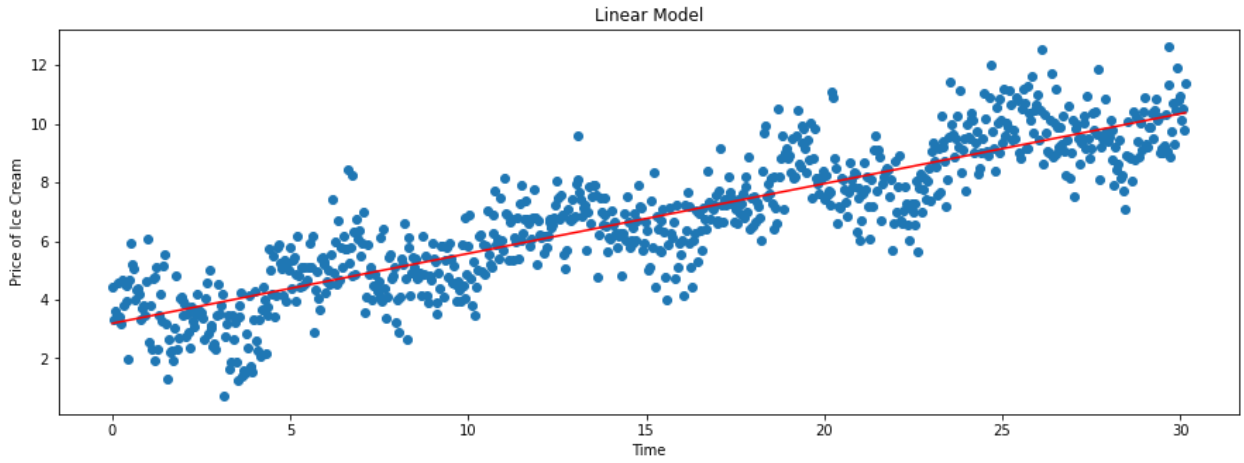
(a) Which estimates βˆ0 and βˆ1 do you get?

(b) Can you conclude that there is a linear trend in the data? This is equivalent to asking whether you can conclude that β1 is equal to zero or not. Justify your answer using statistical inference.

(c) What about β0, can you conclude whether this parameter is zero or not?

```
lineReg = linear_model.LinearRegression()
lineReg.fit(trainingData[['Time']], trainingData.Price)
```

```
intercept = lineReg.intercept_
slope = lineReg.coef_
yPredTrain = intercept + slope * trainingData.Time
plt.figure(figsize=(15,5))
plt.scatter(trainingData.Time, trainingData.Price)
plt.plot(trainingData.Time, yPredTrain, color = 'red')
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Linear Model')
print(intercept, slope)
```




*The figure above shows the output of the code which plots the linear model against the training data.

(a) We get estimates that β0 is 3.191037800253057 and β1 is 0.23839763.
(b) We can conclude that there is a linear trend in the data because the training data looks like a result of some trigonometric functions and for the price of ice cream to keep increasing, there must be a linear parameter.
(c) We can conclude that β0 is not equal to zero because none of the data points shows that at time 0 that the price of ice cream is 0 as well so we can infer that there is an initial price of ice cream.

(3) Improve your model by adding non-linear features (still using only the data contained in train.txt).Suppose your colleague is an economist and she suggests you that, in addition to the linear model you built in the previous question, the following non linear features can help you improve your model:cos(x), log(x), cos(4x), sin(3x), sin(5x) and sin(2x)×cos(2x). Which procedure are you going to use to choose among them? You can come up with your own procedure for feature selection (in this case please don't forget to describe it and justify it) but please always use the the adjusted R2 Coefficient (see the textbook or the wikipedia page https://en.wikipedia.org/wiki/Coefficient_of_determination)to quantify the quality of your

model. Summarize the procedure you are using and conclude which features you should choose to improve your model and why you are choosing these features.

```
trainingData['cos(Time)'] = np.cos(trainingData[['Time']])
trainingData['log(Time)'] = np.log(trainingData[['Time']])
trainingData['cos(4 * Time)'] = np.cos(4 * trainingData[['Time']])
trainingData['sin(3 * Time)'] = np.sin(3 * trainingData[['Time']])
trainingData['sin(5 * Time)'] = np.sin(5 * trainingData[['Time']])
trainingData['sin(2 * Time) * cos(2 * Time)'] = np.sin(2 *
trainingData[['Time']]) *np.cos(2 * trainingData[['Time']])
trainingData.replace([np.inf, -np.inf], 0, inplace=True)
trainingData
```

*The code above adds the features suggested by the economist friend into the dataframe "trainingData".

| | Time | Price | cos(Time) | log(Time) | cos(4 * Time) | sin(3 * Time) | sin(5 * Time) | sin(2 * Time) * cos(2 * Time) |
|---|---|---|---|---|---|---|---|---|
| 0 | 0.000000 | 4.456329 | 1.000000 | 0.000000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 |
| 1 | 0.037737 | 3.301815 | 0.999288 | -3.277118 | 0.988629 | 0.112969 | 0.187567 | 0.075187 |
| 2 | 0.075474 | 3.595955 | 0.997153 | -2.583971 | 0.954775 | 0.224491 | 0.368475 | 0.148665 |
| 3 | 0.113211 | 3.401495 | 0.993599 | -2.178506 | 0.899207 | 0.333140 | 0.536305 | 0.218762 |
| 4 | 0.150947 | 4.571147 | 0.988629 | -1.890824 | 0.823190 | 0.437523 | 0.685097 | 0.283883 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 795 | 30.000795 | 10.928634 | 0.155037 | 3.401224 | 0.812331 | 0.892925 | -0.712091 | 0.291599 |
| 796 | 30.038532 | 10.106108 | 0.192198 | 3.402481 | 0.715395 | 0.836350 | -0.567764 | 0.349360 |
| 797 | 30.076268 | 10.521322 | 0.229085 | 3.403736 | 0.602193 | 0.769069 | -0.403288 | 0.399175 |
| 798 | 30.114005 | 9.814676 | 0.265647 | 3.404990 | 0.475293 | 0.691939 | -0.224493 | 0.439914 |
| 799 | 30.151742 | 11.364662 | 0.301830 | 3.406243 | 0.337584 | 0.605951 | -0.037728 | 0.470648 |

800 rows × 8 columns

*The figure above shows the newly updated dataframe.

In order to find out which features to include in our model, we will conduct an "F test" in which the null hypothesis states that all of the coefficients of the regression features are equal to 0. The alternative hypothesis states that not all of the coefficients of the regression features are equal to 0.

```
from sklearn.feature_selection import SelectKBest, f_regression
fs = SelectKBest(score_func=f_regression, k=7)
fit= fs.fit(trainingData[['Time', 'cos(Time)', 'log(Time)', 'cos(4 *
Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 * Time) * cos(2 *
Time)']], trainingData.Price)
```

```
print(fit.pvalues_)
```

```
[4.30400110e-265 1.15876481e-006 1.41798880e-152 7.81855039e-001
 9.08705200e-004 5.33402074e-001 4.57791880e-001]
```

*The code above selects all of the features available at our disposal and conducts the F test via the sklearn python library.

By conducting the F test, we get the following p values for the corresponding features:

| Feature | P Value |
|---|---|
| Time | 4.30400110e-265 |
| cos(Time) | 1.15876481e-006 |
| log(Time) | 1.41798880e-152 |
| cos(4 * Time) | 7.81855039e-001 |
| sin(3 * Time) | 9.08705200e-004 |
| sin(5 * Time) | 5.33402074e-001 |
| sin(2 * Time) * cos(2 * Time) | 4.57791880e-001 |

With a significance level of $\alpha = 0.005$, we can say that features with a high p value are not statistically relevant to our regression model. Thus, the features we should consider are Time, cos(Time), log(Time), and sin(3 * Time) since their p values are less than the significance level $\alpha$. Furthermore, we reject the null hypothesis in favour of the alternative hypothesis.

```
bestReg = linear_model.LinearRegression()
bestReg.fit(trainingData[['Time', 'cos(Time)', 'log(Time)', 'sin(3 *
Time)']],trainingData.Price)
print(bestReg.intercept_, bestReg.coef_)
```

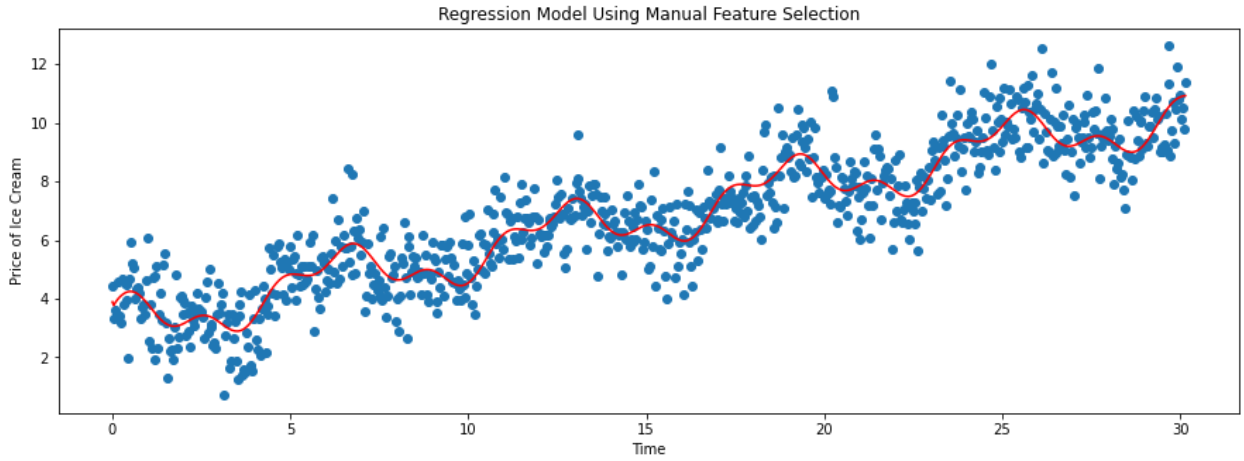
```
3.0776141030639277 [0.23880149 0.81041575 0.05177415 0.37477766]
```

*The code above creates a regression object with the features discussed above and prints out the y intercept and the coefficients of the relevant features.

We obtain the following equation: Price = 3.0776141030639277 + 0.23880149 * Time + 0.81041575cos(Time) + 0.05177415log(Time) + 0.37477766sin(3Time) .

```
y_predict_bestReg = bestReg.predict(trainingData[['Time', 'cos(Time)',
'log(Time)', 'sin(3 * Time)']])
plt.figure(figsize=(15,5))
plt.scatter(trainingData.Time, trainingData.Price)
plt.plot(trainingData.Time, y_predict_bestReg, color = 'red')
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Regression Model Using Manual Feature Selection')
print(r2_score(trainingData.Price, y_predict_bestReg))
```

```
0.8531113462744083
```




*The figure above is a plot of the model created by manually selecting the features using the F test against the training data.

From the plot above, we can see that the model created by choosing 3 additional features fits the training data relatively well with an R2 score of 0.8531113462744083, which is good because the model is not over-fitting the training data.

(4) Use Lasso and Ridge regression with all the features suggested by your economist friend. Do you obtain a comparable result than the one you got at the previous point using feature selection?

```
lassoReg = linear_model.Lasso(alpha=0.0001, max_iter=1000, tol=0.0001)
lassoReg.fit(trainingData[['Time', 'cos(Time)', 'log(Time)', 'cos(4 *
Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 * Time) * cos(2 *
Time)']], trainingData.Price)
print('For Lasso regression:')
print(lassoReg.intercept_, lassoReg.coef_)
lasso_pred = lassoReg.predict(trainingData[['Time', 'cos(Time)',
'log(Time)', 'cos(4 * Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 *
Time) * cos(2 * Time)']])
```

```
print(r2_score(trainingData.Price, lasso_pred))
plt.figure(figsize=(15,5))
plt.scatter(trainingData.Time, trainingData.Price)
plt.plot(trainingData.Time, lasso_pred, color = 'red')
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Lasso Regression Model')
```

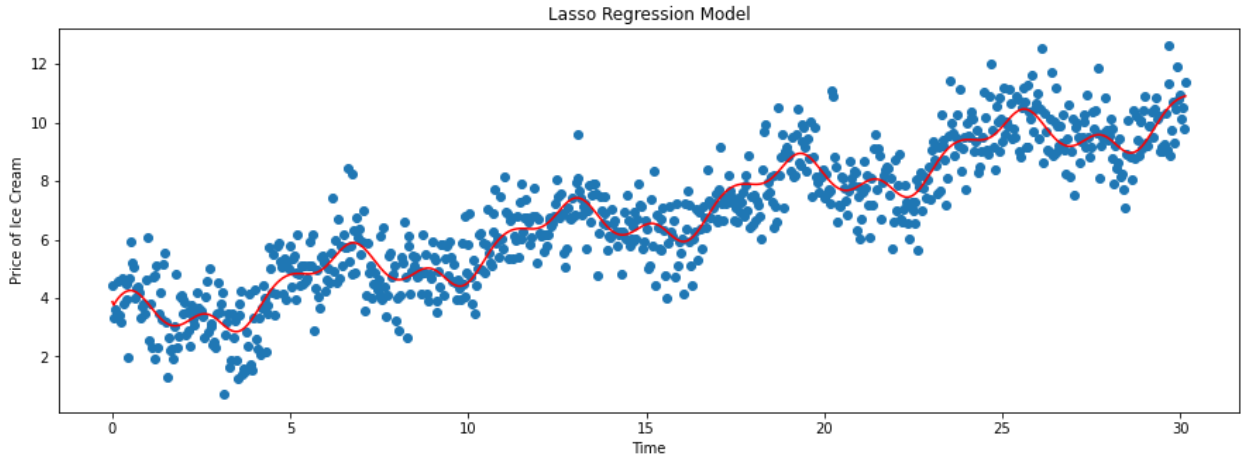
```
For Lasso regression:
3.078371479868208 [ 0.23901748  0.81044556  0.05022364 -0.0209724   0.3740592   0.02250694
 -0.02722501]
0.8532162227688564
```

*The code above creates a lasso regression object with the relevant parameters and all of the features discussed. It then prints out the intercept, respective coefficients, R2 score and plots the regression model against that of the training data.

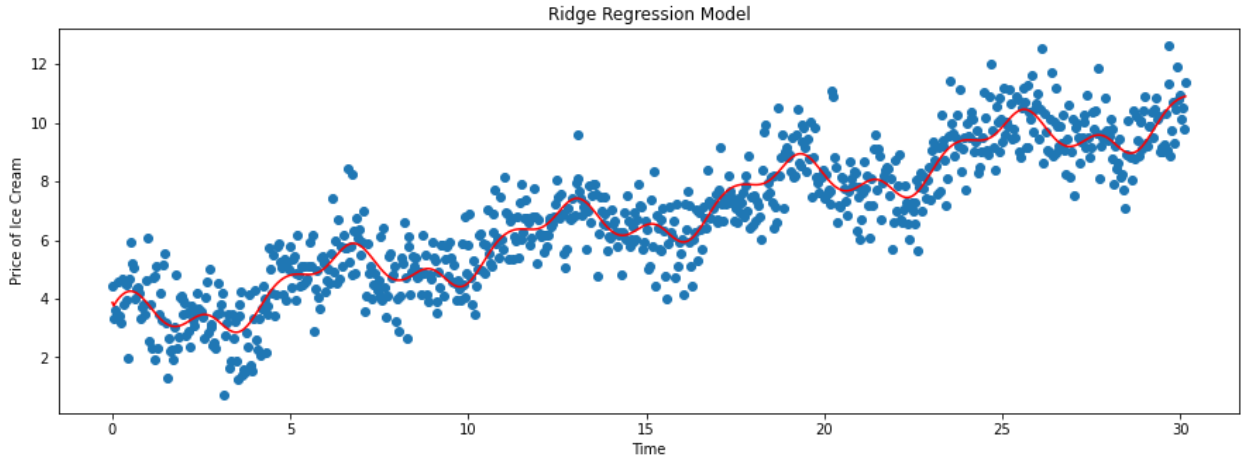
```
from sklearn.linear_model import Ridge
ridgeReg = Ridge(alpha=1.0, max_iter=1000, tol=0.0001)
ridgeReg.fit(trainingData[['Time', 'cos(Time)', 'log(Time)', 'cos(4 *
Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 * Time) * cos(2 *
Time)']], trainingData.Price)
print('For Ridge regression:')
print(ridgeReg.intercept_, ridgeReg.coef_)
ridge_pred = ridgeReg.predict(trainingData[['Time', 'cos(Time)',
'log(Time)', 'cos(4 * Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 *
Time) * cos(2 * Time)']])
print(r2_score(trainingData.Price, ridge_pred))
plt.figure(figsize=(15,5))
plt.scatter(trainingData.Time, trainingData.Price)
plt.plot(trainingData.Time, ridge_pred, color = 'red')
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Ridge Regression Model')
```

```
For Ridge regression:
3.0786308505363986 [ 0.23900387  0.80861714  0.05018455 -0.02111277  0.37338512  0.02264633
 -0.02771521]
0.8532158024311719
```

*The code above creates a ridge regression object with the relevant parameters and all of the features discussed. It then prints out the intercept, respective coefficients, R2 score and plots the regression model against that of the training data.




*The figure above is a plot of the lasso regression model against the training data.




*The figure above is a plot of the ridge regression model against the training data.

As we can see from the plots of the lasso and ridge regression models and their coefficients, the model we made using the F test is very similar. The difference between the lasso and ridge regression models with the model we made is that instead of removing the statistically insignificant features by making their coefficients 0, lasso and ridge regression made it so their coefficients are very small so that they have minimal impact on the model. For example, the coefficient for cos( 4 * Time) is roughly -0.021 instead of 0 in the model created in part 3. Furthermore, because we are not completely eliminating features in lasso and ridge regression, they have a better R2 score compared to the model in part 3: 0.8532162227688564 and 0.8532158024311719 compared to 0.8531113462744083.

(5) The file test.txt contains additional data you can use to evaluate your model on prediction. Compute the output of your model using as input the times in the first column of test.txt and plot the result. Compare the output of the model with the recorded value contained in the second

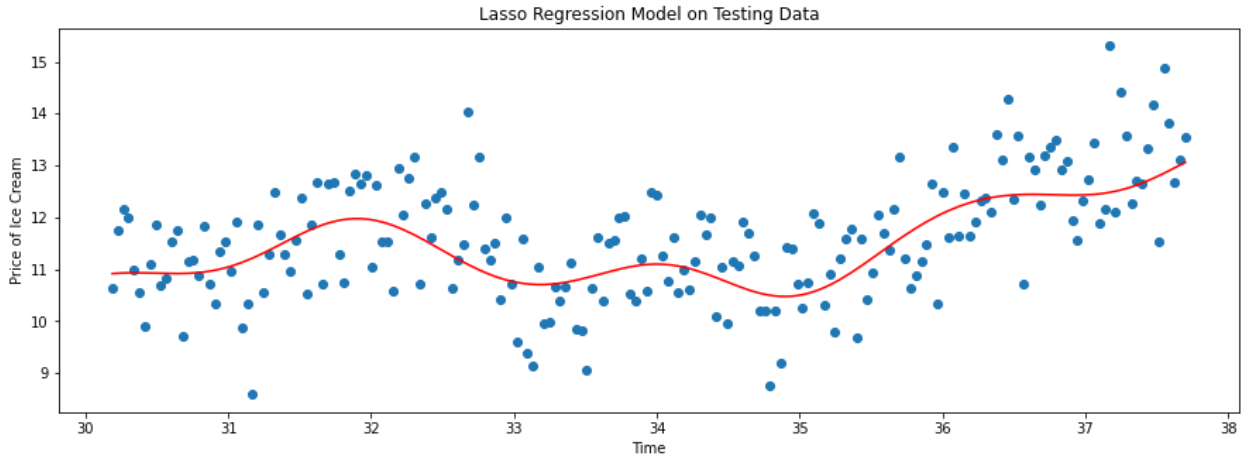
column of text.csv. Do you observe a good agreement between the model and the data in the test.txt file? How do you quantify this agreement? Explain.

Since the model with lasso regression had the highest R2 score on the training data, we will use it to test the model on the testing data.

```
testingData = pd.read_csv('test.csv')
testingData['cos(Time)'] = np.cos(testingData[['Time']])
testingData['log(Time)'] = np.log(testingData[['Time']])
testingData['cos(4 * Time)'] = np.cos(4 * testingData[['Time']])
testingData['sin(3 * Time)'] = np.sin(3 * testingData[['Time']])
testingData['sin(5 * Time)'] = np.sin(5 * testingData[['Time']])
testingData['sin(2 * Time) * cos(2 * Time)'] = np.sin(2 *
testingData[['Time']]) *np.cos(2 * testingData[['Time']])
testingData.replace([np.inf, -np.inf], 0, inplace=True)
modelReg = lassoReg.predict(testingData[['Time', 'cos(Time)', 'log(Time)',
'cos(4 * Time)', 'sin(3 * Time)', 'sin(5 * Time)', 'sin(2 * Time) * cos(2
* Time)']])
plt.figure(figsize=(15,5))
plt.scatter(testingData.Time, testingData.Price)
plt.plot(testingData.Time, modelReg, color = 'red')
plt.xlabel('Time')
plt.ylabel('Price of Ice Cream')
plt.title('Lasso Regression Model on Testing Data')
print(r2_score(testingData.Price, modelReg))
```

```
0.42059652414926085
```

*The code above creates a dataframe for the testing data and adds all the possible features from part 3 into the data frame. Then we use the lasso regression model from part 4 to predict the price values of the testing data. A plot is created and displayed and the R2 score is printed out.




*The figure above shows the lasso regression model from part 4 plotted against the testing data.

When looking at the plot, the lasso regression model seems to fit the data relatively well and matches the general shape of the data. However, looks may be deceiving because the R2 score is 0.42059652414926085 which is very low, so there is not much agreement between the model and the data. This is probably due to the fact that the model is trying to extrapolate the price of ice cream after Time 30 when we are only given data points from Time 0 to Time 30. If the data was randomly divided between Time 0 and Time 38 then the model would have been better but instead, the model is left to extrapolate future time points which is the cause of the low R2 score.

Link to Ipython Notebook file:
https://colab.research.google.com/drive/14ULVVpDjlSEG88pDNq3YdVTRvegykGZs?usp=sharing